

Treelike and Chordal Graphs: Algorithms and Generalizations

Dissertation
for the Degree of
Doctor of Natural Sciences
(Dr. rer. nat.)

FRANK KAMMER



University of Augsburg
Faculty of Applied Computer Science
Theoretical Computer Science Group

May 2010

Referees:

Prof. Dr. Torben Hagerup
Prof. Dr. Rolf Niedermeier
Prof. Dr. Walter Vogler

Day of Defense:

October 22, 2010

Acknowledgements

The wish to write a thesis on treelike graphs arose from the point I took a course called *Algorithmen für NP-harte Probleme* given by Torben Hagerup in the summer of 2002. It is not hard to see that problems on trees can be solved by using a divide-and-conquer approach. However, it was very surprising to me that this approach can be generalized to general graphs by making them treelike. Even if this generalization is known for about two decades, there are still new results on this topic, which fascinate me and which—from my point of view—are important in finding new algorithms for NP-hard problems. I want to thank my supervisor, Torben Hagerup, for that inspiring course, for the guidance in the first steps of my own research, the introduction of several open problems, and the fruitful and interesting discussions.

I also want to thank my parents for making me curious such that I try to get to the bottom of everything. Finally, special thanks go to Sabine Eschenhof and Torsten Tholey. They supported me and helped me in countless ways to write this thesis.

Contents

1	Introduction	1
1.1	Abstract	1
1.2	Algorithms on Trees and Extensions	3
1.3	Trees versus Tree Decompositions	13
1.4	Large treewidth	19
2	Tree Decompositions	24
2.1	An $O(n \log n)$ -Time Algorithm	24
2.2	Finding an X -separator	27
2.3	Clique Trees	35
3	Tree Decompositions on Planar Graphs	41
3.1	Introduction	41
3.2	Peelings, Mountains, and Connectivity	42
3.3	Decomposition into mountains	50
3.4	Separators in Planar Graphs	69
3.5	Special Tree Decomposition for Mountains	70
3.6	Shortcuts	78
3.7	A Linear-Time Algorithm	101
4	Outerplanarity Index	113
4.1	Another Complexity Parameter	113
4.2	Treewidth of ℓ -Outerplanar Graphs	114
4.3	Ideas of the Algorithm	116
4.4	Extended Peelings	117
4.5	Biconnected Graphs	122
4.6	General Graphs	124
5	Generalization of Trees	127
5.1	A Sketch of Monadic Second-Order Logic	127
5.2	Three Convex Coloring Variants	130
5.3	Hardness Results for Convex Coloring	134
5.4	MCRP on Trees	139
5.5	MCRP on Graphs of Bounded Treewidth	141
5.6	MBRP and MRRP	151
5.7	Approximation of MCRP and MRRP	156

6	VDPP on Chordal Graphs	158
6.1	The ℓ -Vertex-Disjoint Paths Problem	158
6.2	A Simple Approach for the ℓ -VDPP	159
6.3	Shortest ℓ -Vertex-Disjoint Paths	161
6.4	A Speedup for the ℓ -VDPP	162
6.5	Hardness of the VDPP	166
7	Generalization of Chordal Graphs	169
7.1	Motivation	169
7.2	New Complexity Parameters	174
7.3	The 3 Complexity Parameters in Relation	177
7.4	Recognition Problems	179
7.5	Algorithms on Intersection Graphs	180
	Index	185
	Bibliography	199
	Curriculum Vitae	206

Chapter 1

Introduction

1.1 Abstract

Many graph problems that are NP-hard on general graphs have polynomial-time solutions on trees. Usually, one traverses a tree bottom-up in linear time and thereby collects all necessary information such that an optimal solution is found at the root of the tree. A key fact is that in a tree every vertex v is the only connection between its descendants and the remaining vertices; in other words, every vertex v defines a so-called separator $\{v\}$.

In 1984, Robertson and Seymour [77] generalized this technique on trees to general graphs by introducing so-called tree decompositions, which are—intuitively speaking—packings of a graph into a tree such that almost every node of the tree guarantees a separator in the graph. For a fast generalization of an algorithm on trees, it is not enough to have several separators since the performance of an algorithm depends strongly on the size of the separators used: the smaller the separators are, the better the algorithm is. For measuring the maximal size of a separator guaranteed by a tree decomposition, Robertson and Seymour introduced a parameter called the width of a tree decomposition. The so-called treewidth of a graph G measures the best possible packing of G into a tree; to be more exact, it is the smallest width of a tree decomposition of G .¹

The result of Robertson and Seymour was followed by a series of papers with faster and faster algorithms for computing a tree decomposition [4, 5, 16, 79]. In 1996, Bodlaender [14] finally solved the problem of finding a tree decomposition on n -vertex graphs with treewidth k in $\Theta(f(k) \cdot n)$ time for some function f . However, his algorithm is not practical already for very small values of k because of an exponential factor depending on k in the running time. The author states only that f is a very fast growing function; Röhrig [80] shows that $f(k) = 2^{\Theta(k^3)}$. The problem of determining whether the treewidth of a given graph is at most a given integer k is NP-complete [4]. Thus, it is not very surprising that the running time is exponential with respect to the parameter treewidth.

One of the most efficient algorithms that computes for each graph G with treewidth k a tree decomposition for G of width $O(k)$ is Reed's algorithm of 1992 [76], whose running time is $O(c^k n \min(n, k! \log n))$ for some large $c \in \mathbb{N}$.

¹For the exact definitions see page 7.

After the introduction of tree decomposition and treewidth many results were published describing a polynomial-time or even a linear-time algorithm for solving an NP-hard graph problem on every graph class whose treewidth is bounded by a constant. Linear-time solvable problems are for example maximum 3-coloring, maximum independent set, maximum triangle matching, minimum edge dominating set, minimum dominating set, minimum maximal matching, and minimum vertex cover. Thus, from the theoretical point of view, using Bodlaender’s algorithm [14], we can solve many NP-hard problems in linear time on every graph class with bounded treewidth. However, practical linear-time algorithms for these problems on graphs of bounded treewidth do not exist because there are no practical algorithms for computing the tree decomposition. But for planar graphs with n vertices and treewidth k , this thesis shows how to find a tree decomposition of width $O(k)$ in time $O(k^{O(1)}n)$ (Kammer and Tholey [56])—moreover, the constants hidden in the O -notation are of moderate size. On planar graphs all the problems mentioned above remain NP-hard [36]. Thus, many problems that are NP-hard on planar graphs can be solved in linear time if the given graph is planar and has constant treewidth. More details about computing a tree decomposition on general and special graphs can be found in Chapters 2 and 3, respectively.

In Chapter 4, we show a simple-to-implement algorithm to compute the so-called outerplanarity index k and a so-called k -outerplanar embedding [53], thereby improving the running time $O(k^3n^2)$ of an algorithm of Bienstock and Monma [12] on n -vertex planar graphs with outerplanarity index k to $O(n^2)$. Moreover, we also describe a linear-time algorithm for computing a 4-approximation of the outerplanarity index and a $4k$ -outerplanar embedding. Given such an embedding, a well known linear-time algorithm can be used to find a tree decomposition for G of width $12k - 1$. This is a second approach to find a tree decomposition on planar graphs.

Note that we are still not able to solve NP-hard problems in general. The parameter treewidth allows us to measure the complexity of a given graph $G = (V, E)$ not only in the size of the input $|V| + |E|$, but additionally in the treewidth k of G . For many NP-hard graph problems, we obtain a so-called fixed-parameter algorithm, i.e., for some function f , an algorithm with a running time of $O(f(k)n^{O(1)})$. Then k is also called a complexity parameter. Note that we already used the treewidth as a complexity parameter. The outerplanarity index can also be used as a complexity parameter. Baker [7] presented a general technique to solve each of the NP-hard problems mentioned above and many others on k -outerplanar n -vertex graphs G in $O(c^k n)$ time for a small constant c if a k -outerplanar embedding of G is given. Combining [53] and [7] we can solve all the NP-hard problems mentioned above and many others on n -vertex graphs with outerplanarity index k in $O(c^k n)$ time.

Many problems considered on graph classes of bounded treewidth have something in common: They can be formulated in so-called monadic second order logic (MSOL). Courcelle [26] showed that all MSOL problems can be solved in polynomial time on graphs with treewidth bounded by a constant. However, his approach only works if the graph problem under consideration can be solved on a tree in polynomial time. In Chapter 5, we consider three similar coloring problems, where two of which can not be solved on trees up to now in polynomial time. Thus, we introduce as a new complexity parameter the maximum number of occurrences of one color and show that the same condition holds for

all three problems: Polynomial-time solvability with respect to this new parameter on trees generalizes to polynomial-time solvability on graphs of bounded treewidth (Kammer and Tholey [54]). Bar-Yehuda, Feldman, and Rawith [9] used a so-called local ratio technique to approximate one of the three coloring problems on trees. By generalizing their approach, we present an approximation algorithm for the same problem on graphs of bounded treewidth [54].

In Chapter 6, we consider so-called chordal graphs, which do not have small treewidth in general, but they have a special tree decomposition called a clique tree. Many NP-hard problems can be solved on chordal graphs by techniques usually applied to graphs of bounded treewidth using this special tree decomposition, e.g., Okamoto, Uno, and Uehara showed that the number of independent sets in a chordal graph can be counted in linear time [72]. We combine the techniques on graphs of bounded treewidth with a so-called sparsification technique and present, for fixed $k \in \mathbb{N}$, the first linear time algorithm for the so-called k -disjoint path problem on chordal graphs (Kammer and Tholey [55]). For general k , there is not much hope to solve the disjoint path problem on chordal graphs since we also show that the general disjoint path problem is NP-hard on chordal graphs [55].

Although chordal graphs are already generalizations of trees by their clique tree, new graph classes generalizing chordal graphs are introduced in Chapter 7. More precisely, three partly new complexity parameters measuring the similarity to chordal graphs are used to construct new and simple polynomial-time approximation algorithms with constant approximation ratios for many NP-hard problems if they are restricted to graphs bounded with respect to one of the new complexity parameters. One of these algorithms is a polynomial-time approximation algorithm with constant approximation ratio for the dominating set problem on the intersection graphs of a restricted set of possibly rotated r -regular polygons. For further information, see also the paper of Kammer, Tholey, and Voepel [57] as well as that of Ye and Borodin [95].

Parts of this dissertation are from publications written jointly with T. Tholey and H. Voepel.

1.2 Algorithms on Trees and Extensions

Before defining trees exactly, we have to introduce some terminology on graphs, which shall be more-or-less standard. A *graph* G is a tuple (V, E) , where V is a finite set of elements called *vertices* or *nodes* and E a finite set containing so-called *edges*. We say that a vertex v (an edge e) *belongs to* G , is *contained in* G , or is *part of* G if $v \in V$ ($e \in E$). The *size* of G is then $|V| + |E|$. If G is a so-called *undirected* graph, an *edge* e is a set $\{u, v\}$ of two vertices of G , which are called *adjacent* or *endpoints* of e ; we also say that e *connects* u with v as well as v with u . If G is a so-called *directed* graph, e is a tuple (u, v) of two vertices of G ; we then say that e *connects* u with v , where u is the *tail* and v the *head* of e . Then u is called *adjacent* to v and v is called *adjacent* to u . Unless explicitly stated otherwise, we assume that a graph has no edge $\{v, v\}$ or (v, v) . Let us define (\emptyset, \emptyset) as the *empty graph*.

The *neighbors* $N_G(v)$ of a vertex v in a graph G are defined by the set of vertices that are adjacent to v and the *degree* $\deg_G(v) = |N_G(v)|$. The index G of N_G and \deg_G is omitted, if it is clear from the context. A *subgraph* of

$G = (V, E)$ is a graph (V', E') with $V' \subseteq V$ and $E' \subseteq E$. Then, G is the *supergraph* of (V', E') . For a graph $G = (V, E)$ and a set of vertices V' , the subgraph of G *induced* by V' —also denoted by $G[V']$ —consists of the vertices in V' and all edges in E with both endpoints in V' . A graph G' is *contained* in a graph G if G' is a subgraph of G .


Moreover, a *path* P is a graph G whose vertex set can be ordered in a list $L = (v_1, \dots, v_\ell)$ ($\ell \in \mathbb{N}$) and whose edge set connects exactly the consecutive vertices of L , i.e., the edges of L are $\{v_1, v_2\}, \dots, \{v_{\ell-1}, v_\ell\}$ if G is an undirected graph, and $(v_1, v_2), \dots, (v_{\ell-1}, v_\ell)$ otherwise. We then also say that v_1, \dots, v_ℓ is the *order* of the vertices of P . The *tail* and the *head* of P are the first and last vertex of L , respectively, and P is called a *path from v_1 to v_ℓ* or a *v_1 - v_ℓ -path*. v_1 and v_ℓ are also called the *endpoints* of P . The vertices of a path different from its endpoints are called *internal*. An *undirected cycle* is an undirected graph that consists of a path P with at least 3 vertices and an edge connecting the endpoints of P . A *directed cycle* is a directed graph that consists of a path P and an edge connecting the head of P with the tail of P . The *length* of a path or a cycle is given by the number of its edges. Two vertices u and v are *connected* if they are endpoints of a path contained in G . Otherwise, they are *disconnected*. A graph G is *connected* if each pair of vertices is connected. Otherwise, G is *disconnected*. Two edges are called *incident* in a graph G if they are part of a path that has length 2 and that is contained in G . The *distance* of two vertices is the length of a shortest path that contains both. A *weighted* graph G is a graph with a so-called *weight-function* that assigns a rational value to the vertices or the edges of G . A weight-function is called *positive* if the image set of it contains only positive numbers. Unless denoted otherwise, a graph is always an undirected graph in this thesis. A *tree* is a connected graph that contains no cycle. Consequently, between each pair of vertices in a tree there exists a unique path. A *forest* is a graph consisting of several trees.

Asymptotic O-notations, complexity classes (mainly P and NP), standard notions from elementary set theory, and propositional logic are used without further explanation; necessary information for understanding this thesis are obtainable from almost any textbook on elementary computer science or mathematics. For clarity, in this thesis a *collection* is similar to a set, $\mathbb{N} = \{1, 2, 3, \dots\}$ are the *natural numbers*, and $\mathbb{N}_0 = \mathbb{N} \cup \{0\}$.

For describing a bottom-up traversal on a tree, some further notation is necessary. The *root* r in a tree T is a designated vertex of T usually drawn as the top-most vertex—denoting the top vertex as root is the usual name, but admittedly from the biological point of view a confusing name. If T is a tree and if r is the *root* of T , we call T also an (*r -*)*rooted* tree. Let $v \neq r$ be a vertex in an r -rooted tree T . Then, the *parent* of v is the only vertex u with being both adjacent to v and part of the path P from v to r . All vertices of P are called *ancestors* of v , and v is called *child* of u . If v is an ancestor of a vertex w , then w is a *descendant* of v . If $v \neq w$, v is also called a *proper* ancestor of w , and w is called a *proper* descendant of v . The *leaves* of T are the vertices in T with no children; the remaining vertices of T are the *inner vertices* of T . The *depth* of a vertex v in a tree T with root r is the length of the path from v to r , and the *height* of v is the length of a longest path from v to one of its descendants. A *subtree* of T is a subgraph of T being a tree. For a vertex w of T , the *subtree* T_w is the subtree of T induced by all descendants of w . T is a *binary tree* if each vertex has at most 2 children: possibly a *left* and possibly a

right one. Finally, let us define the *union* of $\ell \in \mathbb{N}$ given graphs $G_i = (V_i, E_i)$ ($i = 1, \dots, \ell$) as the graph $(\bigcup_{i=1}^{\ell} V_i, \bigcup_{i=1}^{\ell} E_i)$.

How can we traverse a tree *bottom-up*? For answering this question let us consider the following example: Assume that we are interested in a biggest set of soccer teams that do not have played pairwise against each other. The Summer Olympics event in 1912 was the first international soccer contest with several teams such that each team represented a different country. The winner of the gold medal was decided by a knockout tournament that is shown in Fig. 1.2.1. For our concern, the representation of the tournament by a graph T —as shown in Fig. 1.2.2—is more useful, where each node corresponds to a participant and a pair of vertices is adjacent if and only if the corresponding participants played a match against each other. Note that T is a tree. Searching for the biggest set of teams not having played against each other during the knockout tournament in 1912 is then equivalent to finding a so-called maximum independent set in T . Given a graph $G = (V, E)$ an *independent set* in G is a set $V' \subseteq V$ such that no pair of vertices in V' is adjacent. A *maximum independent set*—abbreviated by MIS—is an independent set that among all these sets has maximum cardinality. Practical applications of MIS are for example scheduling [38], information retrieval [6], signal transmission [8], clustering in pattern recognition [27], and stereo vision correspondence [48]. See also [18] for further applications of MIS.

In the following, a description of an algorithm is given for finding an MIS on a tree T . In addition, the algorithm is illustrated on the tree of Fig. 1.2.2. First, choose as the root r of T an arbitrary node of T , say . For each node w of T , compute for the subtree T_w two different biggest independent sets S_w^+ and S_w^- such that w is contained in S_w^+ but not in S_w^- . A computation for our

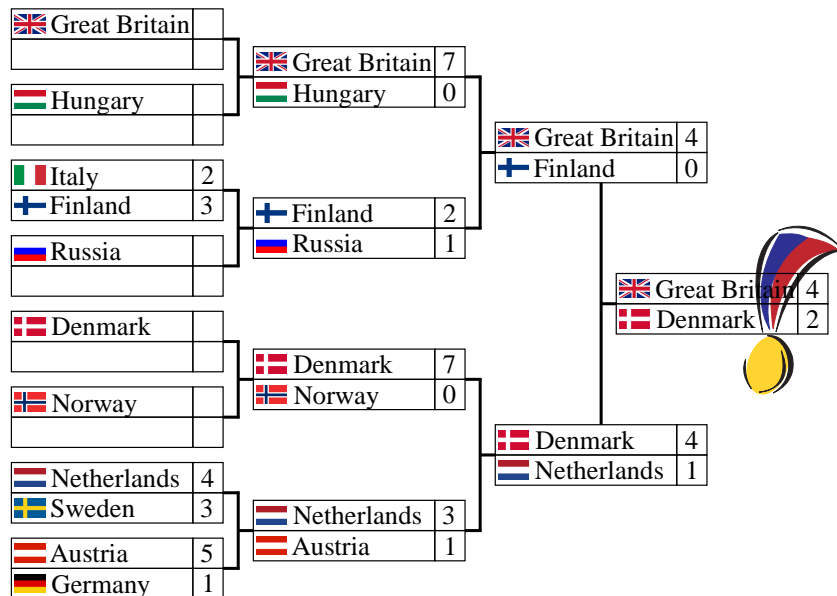


Figure 1.2.1: Knockout tournament in 1912.

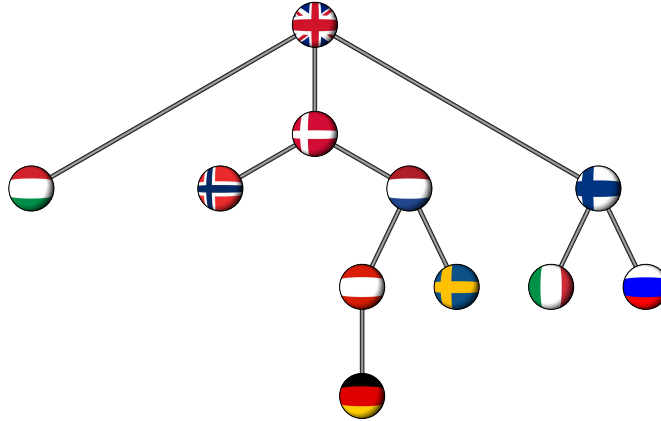


Figure 1.2.2: Knockout tournament in 1912 represented by a graph.

example is shown in Table 1.2.3. Starting with the leaves of T , for each leaf w , set $S_w^+ = \{w\}$ and $S_w^- = \emptyset$. Then continue stepwise with a vertex w whose children are already processed. Compute S_w^+ as the union of $\{w\}$ and of $S_{\tilde{w}}^-$ over all children \tilde{w} of w . Compute S_w^- by first defining, for each child \tilde{w} of w , $Q_{\tilde{w}}$ as either $S_{\tilde{w}}^+$ or $S_{\tilde{w}}^-$ such that $Q_{\tilde{w}}$ has as many vertices as possible. Then, compute S_w^- as the union of $Q_{\tilde{w}}$ over all children \tilde{w} of w . Note that by the construction the computed sets S_w^+ and S_w^- are indeed independent and as big as possible under the condition that $w \in S_w^+ \setminus S_w^-$. At the end, return as an MIS a biggest set among the two sets S_r^+ and S_r^- computed at the root of T . For our example, a possible MIS is $\{ \text{Italy}, \text{Denmark}, \text{Germany}, \text{Sweden}, \text{Italy}, \text{Russia} \}$.

w	S_w^+	S_w^-	s_w^+	s_w^-
		/	1	0
		/	1	0
		/	1	0
		/	1	0
		/	1	0
		/	1	0

w	S_w^+	S_w^-	s_w^+	s_w^-
			1	1
			2	2
			3	3
			1	2
			6	6

Table 1.2.3: Computation of an MIS in the graph of Fig. 1.2.2.

If we want to make the algorithm above more efficient, we do the following: For each node w of the tree T , compute during the bottom-up traversal only the cardinalities s_w^+ and s_w^- of S_w^+ and S_w^- , respectively. If w is a leaf, $s_w^+ = 1$ and $s_w^- = 0$. Otherwise, if $C(w)$ denotes the set of children of w , we simply have to compute $s_w^+ = 1 + \sum_{\tilde{w} \in C(w)} s_{\tilde{w}}^-$ and $s_w^- = \sum_{\tilde{w} \in C(w)} \max(s_{\tilde{w}}^+, s_{\tilde{w}}^-)$. The cardinality of an MIS in T is then $\max(s_r^+, s_r^-)$. Since the time for computing the cardinalities of a leaf is constant and of a non-leaf w is linear in the number of children of w , the whole computation of the cardinality of an MIS can be done in a time linear in the number of nodes of T .

An MIS can be obtained, if, for each node w of T and each child \tilde{w} of w , we keep track during the computation of s_w^- , which of the two values $s_{\tilde{w}}^+$ and $s_{\tilde{w}}^-$ is the larger one. Processing the nodes finally in reverse order with respect to the

bottom-up traversal, i.e., traversing the tree in a so-called *top-down* direction, we can combine—again in linear time—all sets that make a contribution to the obtained cardinality. In our example, we obtain 6 as the size of an MIS. We can now find MIS by computing the 6 'backwards' as follows:

$$\begin{aligned}
6 &= s_{\text{UK}}^- = 0 + 1 + 3 + 2 = 0 + s_{\text{UK}}^+ + s_{\text{DK}}^+ + s_{\text{FI}}^- \\
&= 0 + |\text{UK}| + [|\text{DK}| + s_{\text{UK}}^- + s_{\text{FI}}^-] + [0 + s_{\text{UK}}^+ + s_{\text{FI}}^+] \\
&= 0 + |\text{UK}| + [|\text{DK}| + 0 + (0 + s_{\text{UK}}^- + s_{\text{FI}}^+)] + [0 + |\text{UK}| + |\text{FI}|] \\
&= 0 + |\text{UK}| + [|\text{DK}| + 0 + (0 + (0 + s_{\text{UK}}^+) + |\text{FI}|)] + [0 + |\text{UK}| + |\text{FI}|] \\
&= 0 + |\text{UK}| + [|\text{DK}| + 0 + (0 + (0 + |\text{UK}|) + |\text{FI}|)] + [0 + |\text{UK}| + |\text{FI}|],
\end{aligned}$$

i.e., also by this approach we are able to obtain the already known MIS $\text{UK}, \text{DK}, \text{UK}, \text{FI}, \text{UK},$ and FI .

If we additionally take the soccer matches of 1912 under consideration that are played to figure out the rankings of the remaining countries, the graph of Fig. 1.2.2 modifies to the graph shown in Fig. 1.2.4, which is not a tree.

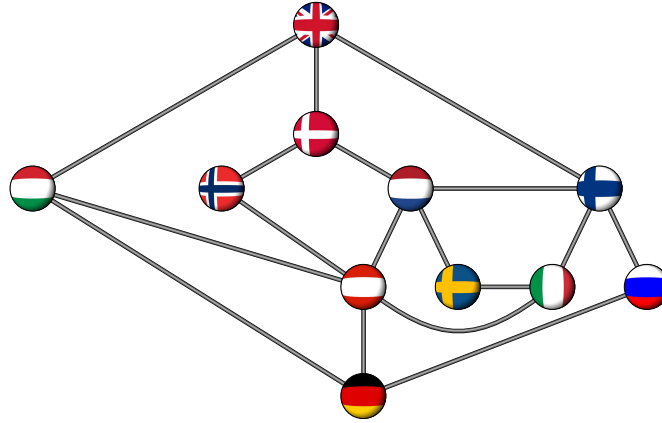


Figure 1.2.4: All matches of 1912 represented by a graph.

Let us now state how a graph G can be packed into a tree by defining tree decomposition exactly. Note that we want to traverse G somehow bottom-up—as if G is a tree—however, we have to find a way to handle the cycles of G . The idea is to map subgraphs of G to nodes of T .

Definition 1.2.1 (tree decomposition, bag). A *tree decomposition* for a graph $G = (V, E)$ is a pair (T, B) with $T = (W, F)$ being a tree and B being a mapping $W \rightarrow \{V' \mid V' \subseteq V\}$ such that the following properties are satisfied:

(TD1) $\bigcup_{w \in W} G[B(w)] = G$.

(TD2) For each vertex $v \in V$, (TD2) $_v$ holds:

The nodes w with $v \in B(w)$ induce a subtree in T .

For each node $w \in W$, $B(w)$ is called the *bag* of w .

The *width* and the *size* of a tree decomposition is the maximal size of a bag minus one and the sum of all cardinalities of its bags, respectively. The *treewidth* of a graph G is the smallest width of any tree decomposition for G , and the *treewidth* of a graph class \mathcal{G} is the smallest existent $k \in \mathbb{N} \cup \{\infty\}$ such that each $G \in \mathcal{G}$ has treewidth at most k . A class \mathcal{G} of graphs is of *bounded treewidth* if there exists a constant c such that the treewidth of G is at most c for every $G \in \mathcal{G}$. A tree decomposition (T, B) is called *rooted* and *binary* if T is a rooted and binary tree, respectively. For a better understanding, if (T, B) is a tree decomposition, the vertices of T are always called nodes and are usually denoted by the character w . Moreover, for set W' of some nodes of T and a subtree T' of T , let us define $B(W')$ and $B(T')$ as the union of $B(w)$ over all nodes w in W' and T' , respectively. For an arbitrary set Z , let us end these definitions by calling two sets X and Y to be *Z-conform* if $X \cap Z = Y \cap Z$.

An MIS in a graph G can be found as follows—we can choose G for an example as the graph shown in Fig. 1.2.4. First construct a rooted, binary tree decomposition (T, B) for G such that the number of nodes of T is bounded by the number of vertices of G ; for details how to find such a tree decomposition see the end of Section 2.2. Similar to our last algorithm, we traverse the tree T bottom-up. For the graph in Fig. 1.2.4, a tree decomposition (T, B) is shown in Fig. 1.2.5. As the root we choose an arbitrary node of T , say node B. Property (TD1) guarantees that all vertices and all edges are considered. We next compute, for each node w of T , the subgraph $G[B(T_w)]$ and, for each independent set $Y \subseteq B(w)$ —a non independent set can not be extended to an independent set in G —an MIS S_w^Y such that $Y = S_w^Y \cap B(w)$. This construction might seem to be complicated, however, at a second glance this is the canonical generalization of the algorithm on trees. Having a simpler notation we define, for each node w of T , $I(B(w))$ as the collection of all sets $Y \subseteq B(w)$ being independent in G .

Starting with the leaves of T , for each leaf w and each $Y \subseteq B(w)$, check if Y is an independent set in G . Only if this is the case, i.e., if $Y \in I(B(w))$, define $S_w^Y = Y$. (Otherwise leave S_w^Y undefined.) Then continue stepwise with a node w whose children are already processed. For each $Y \subseteq B(w)$, check if $Y \in I(B(w))$. Only in this case continue with the computation of S_w^Y as

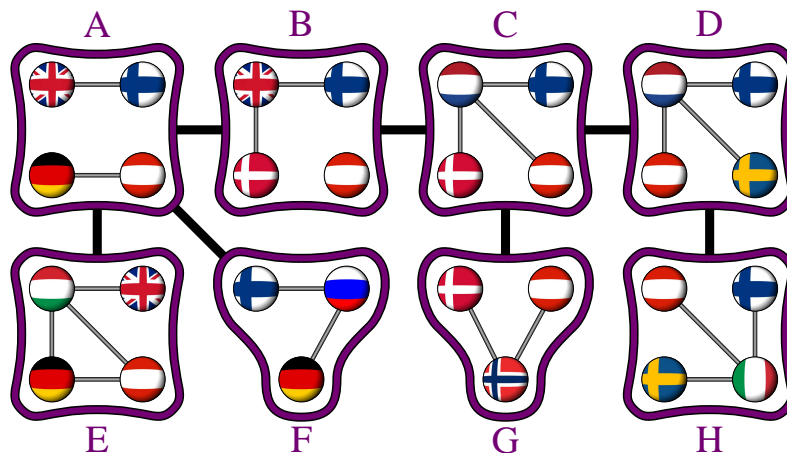



Figure 1.2.5: A tree decomposition for the graph of Fig. 1.2.4.

described next. (Otherwise leave S_w^Y again undefined.) Note that property (TD2) guarantees that $B(w)$ contains all vertices that are contained in two or more graphs $G[B(T_{\tilde{w}})]$ with \tilde{w} being a child of w . Thus, as long as we fix the vertices of $B(w)$ that are part of S_w^Y , for each child \tilde{w} of w , we can independently search for a largest contribution $Q_{\tilde{w}}$ to S_w^Y of independent vertices in $G[B(T_{\tilde{w}})]$ as follows.

After setting initially $Q_{\tilde{w}} = \emptyset$, consider all sets $X \in I(B(\tilde{w}))$. For the vertices in $Z = B(\tilde{w}) \cap B(w)$, check if X is Z -conform with Y . If so and if furthermore $|S_{\tilde{w}}^X| > |Q_{\tilde{w}}|$, update $Q_{\tilde{w}} = S_{\tilde{w}}^X$.

After the iteration over all children $C(w)$ of w define $S_w^Y = Y \cup \bigcup_{\tilde{w} \in C(w)} Q_{\tilde{w}}$. After processing the root r of T , return finally as an MIS of G a biggest set among all sets S_r^Y with $Y \in I(B(r))$. The computation on the graph of Fig. 1.2.4 is shown in Table 1.2.6—however, the computation on the leaves of T is skipped. As a solution, we can take for example .

For a more efficient algorithm—similar to the algorithm on trees—we compute during the bottom-up traversal only the cardinalities s_w^Y of S_w^Y for each node w of T and each $Y \in I(B(w))$ as follows. If w is a leaf, set $s_w^Y = |Y|$. If w has children $C(w)$, compute:

$$s_w^Y = |Y| + \sum_{\tilde{w} \in C(w)} \max_{\substack{X \in I(B(\tilde{w})) \text{ being} \\ (B(w) \cap B(\tilde{w}))\text{-conform with } Y}} (s_{\tilde{w}}^X - |X \cap Y|)$$

The cardinality s of an MIS of G can be obtained at the root r of T by taking the maximum of s_r^Y over all $Y \in I(B(r))$. It is easy to see that the size s_w^Y for each node w and each set $Y \in I(B(w))$ can be computed by using only the cardinalities computed at the children of w . If k is the width of (T, B) , the time to spend at a node w with $\ell \in \mathbb{N}$ children is thus bounded by $O(2^{k+1} \cdot ((k+1)^2 + \ell 2^{k+1}))$, where the first factor bounds the number of subsets $Y \subseteq B(w)$ and the second factor bounds the time for checking if Y is an independent set and the time for the computation of the cardinalities $|Q_{\tilde{w}_1}|, \dots, |Q_{\tilde{w}_\ell}|$ for the children $\tilde{w}_1, \dots, \tilde{w}_\ell$ of w .

By a subsequent top-down traversal of T a maximum independent set I of G can be found as follows: Fix $v \in I$ or $v \notin I$ for each $v \in V$ in such a way that these restrictions do not decrease the cardinality of I . Since T has no more nodes as G has vertices and since each node of T has at most two children, i.e., $\ell \leq 2$, the algorithm above runs on n -vertex graphs in time $O(n \cdot f(k))$ for some simple-exponential function f . In other words, if k is a small constant, the algorithm runs in time linear in the number of vertices of the given graph.

As good as the algorithm above is for small k , the algorithm is not practicable for large k : Let X be the $k+1$ elements of a bag of largest cardinality. Then we have to check the independence of each set $Y \subseteq X$ in G , i.e., we have to check 2^{k+1} sets. As a consequence, our algorithm described at last runs in polynomial time if we use it only on a graph class \mathcal{G} where every n -vertex graph G in \mathcal{G} has treewidth bounded by $O(\log n)$, in other words, on a graph class with *logarithmically bounded* treewidth.

For some special graph classes, we can obtain a polynomial or even a linear time algorithm for the MIS problem even if the graph class has no logarithmically bounded treewidth. If we return to our examples of finding a biggest set of teams that did not play against each-other during a soccer competition, e.g.,

w	Y	$S_w^Y \setminus Y$	contribution of the children	s_w^Y
A	/		$Q_E = \{\text{Italian}\}, Q_F = \{\text{Russian}\}$	2
A			$Q_E = \{\text{UK}\}, Q_F = \{\text{Russian}\}$	2
A			$Q_E = \{\text{Italian}\}, Q_F = \{\text{French}\}$	2
A		/	$Q_E = \{\text{German}\}, Q_F = \{\text{German}\}$	1
A			$Q_E = \{\text{Russian}\}, Q_F = \{\text{Russian}\}$	2
A		/	$Q_E = \{\text{UK}, \text{German}\}, Q_F = \{\text{German}\}$	2
A			$Q_E = \{\text{UK}, \text{Russian}\}, Q_F = \{\text{French}, \text{Russian}\}$	3
A		/	$Q_E = \{\text{German}\}, Q_F = \{\text{German}\}$	2
A		/	$Q_E = \{\text{Russian}\}, Q_F = \{\text{French}\}$	2
D	/		$Q_H = \{\text{Italian}\}$	1
D			$Q_H = \{\text{Italian}\}$	2
D		/	$Q_H = \{\text{French}\}$	1
D		/	$Q_H = \{\text{Russian}\}$	1
D		/	$Q_H = \{\text{German}\}$	1
D		/	$Q_H = \{\text{French}, \text{Russian}\}$	2
D		/	$Q_H = \{\text{French}, \text{German}\}$	2
D		/	$Q_H = \{\text{Russian}, \text{German}\}$	2
D		/	$Q_H = \{\text{French}, \text{Russian}, \text{German}, \text{Italian}\}$	3
C	/		$Q_G = \{\text{UK}\}, Q_D = \{\text{Italian}\}$	2
C			$Q_G = \{\text{UK}\}, Q_D = \{\text{Russian}, \text{Italian}\}$	3
C			$Q_G = \{\text{UK}\}, Q_D = \{\text{French}, \text{German}\}$	3
C			$Q_G = \{\text{Russian}\}, Q_D = \{\text{Italian}\}$	2
C			$Q_G = \{\text{Russian}\}, Q_D = \{\text{Russian}, \text{German}\}$	2
C			$Q_G = \{\text{Russian}\}, Q_D = \{\text{French}, \text{German}\}$	3
C			$Q_G = \{\text{Russian}\}, Q_D = \{\text{French}, \text{Russian}, \text{German}\}$	3
C			$Q_G = \{\text{Russian}, \text{German}\}, Q_D = \{\text{Russian}, \text{German}, \text{Italian}\}$	3
C			$Q_G = \{\text{Russian}, \text{German}\}, Q_D = \{\text{French}, \text{Russian}, \text{German}, \text{Italian}\}$	4
B	/		$Q_A = \{\text{Italian}, \text{Russian}\}, Q_C = \{\text{Russian}, \text{UK}, \text{Italian}\}$	5
B			$Q_A = \{\text{UK}, \text{Russian}\}, Q_C = \{\text{Russian}, \text{UK}, \text{Italian}\}$	5
B			$Q_A = \{\text{French}, \text{Italian}\}, Q_C = \{\text{French}, \text{UK}, \text{German}\}$	4
B			$Q_A = \{\text{Italian}, \text{Russian}\}, Q_C = \{\text{UK}, \text{Italian}\}$	4
B			$Q_A = \{\text{Russian}, \text{German}\}, Q_C = \{\text{Russian}, \text{German}\}$	3
B			$Q_A = \{\text{UK}, \text{Russian}, \text{Russian}\}, Q_C = \{\text{Russian}, \text{German}\}$	4
B			$Q_A = \{\text{French}, \text{Russian}\}, Q_C = \{\text{French}, \text{Russian}, \text{German}\}$	3
B			$Q_A = \{\text{French}, \text{Italian}\}, Q_C = \{\text{UK}, \text{Italian}\}$	4
B			$Q_A = \{\text{Russian}, \text{Russian}\}, Q_C = \{\text{Russian}, \text{German}, \text{Italian}\}$	4
B			$Q_A = \{\text{French}, \text{Russian}\}, Q_C = \{\text{French}, \text{UK}, \text{German}, \text{Italian}\}$	4

Table 1.2.6: Computation of an MIS in the graph of Fig. 1.2.4.

the first Soccer World Cup in 1930, then we have a graph of such a special graph class, where the running time does not depend exponentially on the treewidth.

In 1930, the teams were drawn into four groups. All teams in a group played pairwise against each other, and the winner of each of the four groups qualified for a knockout tournament. See the left side of Fig. 1.2.7 for the representation of the tournament in 1930 by a graph.

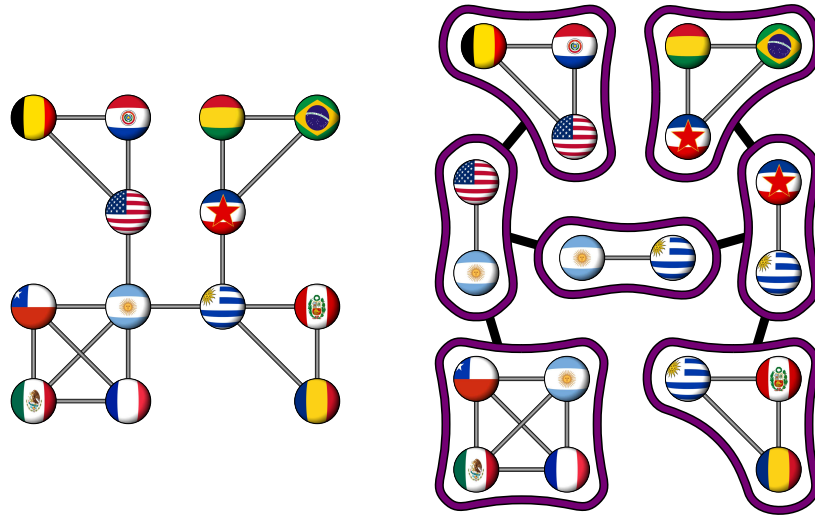















Figure 1.2.7: Soccer World Cup in 1930 represented by a graph G (left side) and a tree decomposition for G (right side). The participants of the tournament in 1930 are Argentina , Belgium , Bolivia , Brazil , Chile , France , Jugoslavia , Mexico , Paraguay , Peru , Rumania , Uruguay , and U.S.A. .

Definition 1.2.2 (chord, chordless, chordal graph). A *chord* of a path X or a cycle X in a graph G is an edge e of G connecting two vertices of X such that e does not belong to X . Moreover, X is called *chordless* in G if no edge of G is a chord of X . A graph G is *chordal* if each of its cycles of length at least 4 has a chord.

Definition 1.2.3 (clique, clique tree). A *clique* in a graph is a set of vertices that are all pairwise adjacent. A clique C in a graph G is *maximal* if no other vertex of G is adjacent to all vertices of C . Moreover, let us call an ℓ -*clique* a clique with ℓ vertices. A *clique tree* for a graph G is a tree decomposition (T, B) for G with the additional property:

(TDC) The vertices contained in each bag induce a maximal clique in G different to all maximal cliques induced by other bags.

It is easy to verify that the graph G on the left side of Fig. 1.2.7 is chordal and that the tree decomposition shown on the right side is a clique tree for G . An example of a clique is shown in Fig. 1.2.8.



Figure 1.2.8: In every tournament of the South American Championship of Nations all participants play against each other. The number of participants varies between 3 and 8. In 1922, the tournament had 5 participants, and its process is shown by a 5-clique.

It is well known [17, 40, 92] that chordal graphs are exactly the graphs for which a clique tree exists—see also Lemma 2.3.9. By Theorem 2.3.8, a clique tree for a chordal graph $G = (V, E)$ can be found having at most $|V|$ nodes and size $O(|V| + |E|)$. Further details can be found in Section 2.3.

On a chordal graph G , we can find an MIS by using a clique tree (T, B) of G similar to the algorithm using tree decompositions. The only modification is that, for a node w of T , we do not have to compute the cardinalities s_w^Y of up to $2^{|B(w)|}$ different independent sets, where each set is $B(w)$ -conform to a set $Y \subseteq B(w)$. The reason for this is that Y has to be an independent set in G , i.e., $|Y| \leq 1$. Consequently, we have to compute at a node w only the cardinalities of $B(w) + 1 = O(B(w))$ independent sets in $G[B(T_w)]$. Thus, for the whole computation of a maximum independent set in G , we have to compute only $O(|V| + |E|)$ cardinalities of independent sets. For an efficient computation at the parent w' of w , we additionally compute at w the maximum cardinality s_w^{\max} of an independent set that does not contain a vertex of $B(w')$; in other words, $s_w^{\max} = \max_{X \in I(B(w) \setminus B(w'))} s_w^X$. The computation of s_w^{\max} can be done during the computations of the cardinalities of independent sets of w and takes asymptotically no extra time. Of course, we can compute the cardinalities s_w^Y as on general tree decompositions; however, the same result can be obtained faster for a non-leaf w with children $C(w)$ as follows.

$$s_w^Y = |Y| + \sum_{w'' \in C(w)} \begin{cases} s_{w''}^Y - |Y| & \text{if } \emptyset \neq Y \subseteq B(w'') \\ s_{w''}^{\max} & \text{otherwise} \end{cases}$$

The time for computing the cardinality of an independent set at a node w only depends on the number of children of w since, for each child w'' of w , we have to add one summand. As a consequence, we can bound the time for computing the cardinality of an independent set by $O(\deg(w) + 1) = O(n)$.

Recall that, for each graph G with n vertices and m edges, the sum over all cardinalities of the bags of a clique tree for G is linear in $n + m$, and therefore we can solve the MIS problem on G in $O((n + m)n)$ time. Indeed, we have roughly estimated the running time for computing the cardinality of an independent set, but the algorithm above needs for some chordal graphs more than linear time. For example we can take the graphs obtained, for all $n \in \mathbb{N}$, from an $\lfloor \sqrt{n} \rfloor$ -clique C by connecting $n - \lfloor \sqrt{n} \rfloor$ vertices to one vertex of C .

For improving the running time note that the algorithm on a chordal graph G does not use the fact that each bag induces a maximal clique in G ; it is only

important that each bag induces a clique in G . Let us define a *weak clique tree* as a tree decomposition with the additional property:

(TDC') The vertices of each bag induce a clique in G .

Since each clique tree is also a weak clique tree, each chordal graph has also a weak clique tree. As we see in Section 2.3, every chordal graph $G = (V, E)$ has a weak clique tree (T, B) such that T is binary tree with $O(|V|)$ nodes and such that (T, B) is of size $O(|V| + |E|)$. Using such a weak clique tree we can compute each cardinality s_w^Y in $O(1)$ time and solve the MIS problem on chordal graphs in linear time.

1.3 Trees versus Tree Decompositions

As we know from the last section, efficient algorithms for many NP-hard problems exist if a given graph is a tree or if its treewidth is logarithmically bounded. Why do both graph classes have efficient algorithms?

If we consider Fig. 1.3.1, a tree and a graph with a small treewidth—see the tree decomposition in Fig. 1.3.2—seem to have not much in common. However, this is only the case at first sight.

Trees and graphs with a small treewidth have similar so-called separation properties. Before showing that we need two further definitions.

Definition 1.3.1 (connected component). A *connected component* C of a graph $G = (V, E)$ is a connected subgraph of G induced by some vertices $V^* \subseteq V$ such that no real superset of V^* induces in G a connected graph.

Definition 1.3.2 (separator). In a graph $G = (V, E)$, a *separator* for a subgraph $G' = (V', E')$ of G and for a set $V' \subseteq V$, respectively, is a subset S of V such that at least two vertices in $V' \setminus S$ are not connected in $G[V - S]$. If no set V' is given in the definition above, we assume implicitly $V' = V$. In addition, the *size* of a separator is equal to its cardinality.

In a tree T with at least 3 vertices every vertex of T with degree strictly greater than 1 is a separator. In particular, we can find in every subtree consisting of at least 3 vertices a separator of size 1. Therefore, we can divide a tree again and again into smaller parts and solve a given problem first on these small parts. Again because of the small separator we can combine quickly the solutions of the small parts to a solution of a ever bigger part. In other words, we can use a so-called *divide-and-conquer approach* for solving a given problem on a tree.

We next want to show that if (T, B) is a tree decomposition of width k for a graph G , then every node w of T defines a separator in G of size $k + 1$ if w is a separator in T . Unfortunately, this is not true in general. Therefore, we next define a special kind of tree decompositions.

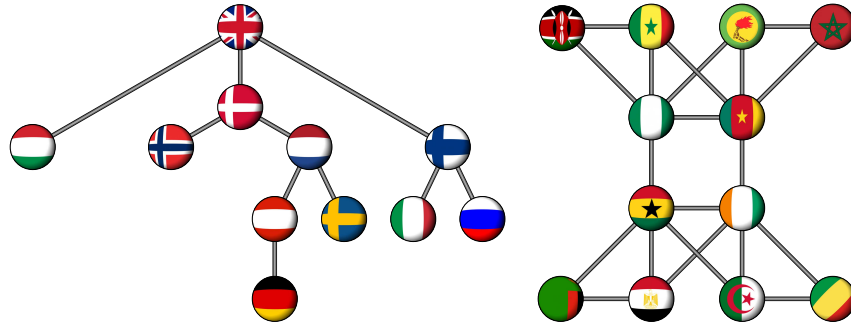














Figure 1.3.1: If we represent each match by an edge, we see on the left side again the knockout tournament of 1912 and on the right side the matches of the African Cup of Nations in 1992 with participants Algeria , Cameroon , Congo , Côte d'Ivoire , Egypt , Ghana , Kenya , Morocco , Nigeria , Senegal , Zaire , and Zambia .

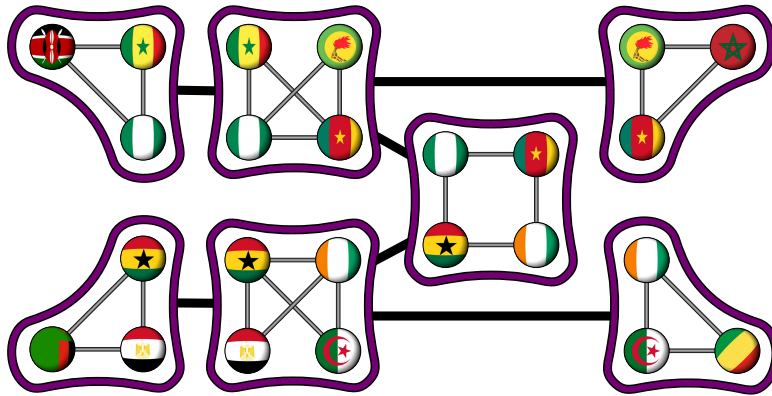


Figure 1.3.2: A tree decomposition for the right graph of Fig. 1.3.1.

Definition 1.3.3 ((k) -normal tree decomposition). Let $k \in \mathbb{N}$. A tree decomposition (T, B) is called k -normal if the following two properties hold:

- $|B(w)| = k + 1$ for all nodes w of T and
- $|B(w_1) \setminus B(w_2)| = 1$ for all adjacent nodes w_1 and w_2 in T .

The next lemma shows that considering only normal tree decompositions does not impose any restriction on our comparison between trees and graphs with small treewidth.

Lemma 1.3.4. *Every graph $G = (V, E)$ with treewidth $k \in \mathbb{N}$ has a k -normal tree decomposition. Additionally, if a tree decomposition (T, B) for G of width $\ell \in \{k, k + 1, \dots\}$ is given and if T has $O(|V|)$ nodes, an ℓ -normal tree decomposition for G can be found in time $O(\ell|V|)$.*

Proof. Note that G has at least $\ell + 1$ vertices since otherwise (T, B) can not have width ℓ . We stepwise modify (T, B) to make it ℓ -normal:

1. Choose as root of T a node r whose bag $B(r)$ has maximal cardinality. Thus, $|B(r)| = \ell + 1$.
2. Traverse T top-down starting from r .
 - (a) If we visit during the traversal a node w with parent w' whose bag contains strictly fewer than $\ell + 1$ vertices of G , copy $\ell + 1 - |B(w)|$ vertices of $B(w') \setminus B(w)$ to $B(w)$.
 - (b) Moreover, if now $B(w) = B(w')$, remove w from T and connect all children of w to w' . Otherwise, if $|B(w) \setminus B(w')| > 1$, replace in T the edge $\{w', w\}$ by a path of length $|B(w) \setminus B(w')|$ and assign bags to the new nodes such that bags of adjacent nodes differ only by one vertex.

Let (T', B') be the rooted tree decomposition obtained. Concerning the running time of the algorithm above we can observe that there is the following injective function from the edges of T' to the vertices of G : Assign to each edge $\{w, w'\}$ with w child of w' the vertex v of G with $v \in B(w) \setminus B(w')$. Consequently, T' has at most $|V|$ edges and $|V| + 1$ nodes. The time we spend during the computation of (T', B') at each node is $O(\ell)$. Thus, the ℓ -normal tree decomposition can be found in $O(\ell|V|)$ time. \square

For the graph shown on the right side of Fig. 1.3.1, if we take the tree decomposition shown in Fig. 1.3.2 and choose the upper node in the middle as root, then we obtain the 4-normal tree decomposition shown in Fig. 1.3.3 by applying the algorithm of the proof of Lemma 1.3.4.

Let us return to our compensation of a tree and of a graph G with logarithmically bounded treewidth k —now taking a normal tree decomposition for

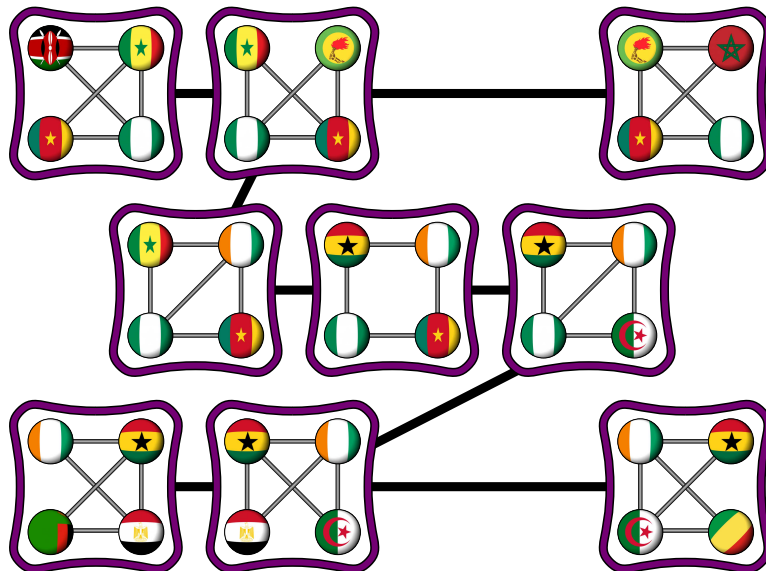


Figure 1.3.3: A 3-normal tree decomposition for the right graph of Fig. 1.3.1.

G into consideration. Property (TD1) ensures intuitively that all vertices and edges of G are part of a bag of (T, B) . More interesting is property (TD2), which guarantees—similar to trees—that for each node w in T with degree strictly greater than 1, $B(w)$ is a separator of size $k + 1$ in G . Moreover, also for each edge $\{w_1, w_2\}$ in T , $B(w_1) \cap B(w_2)$ is a separator of size k in G . As a consequence of these observations, we can prove the following lemma.

Lemma 1.3.5. *If a graph $G = (V, E)$ has treewidth $k \in \mathbb{N}$, there exists for each set $V' \subseteq V$ with $|V'| \geq k + 2$ a separator of size k in G .*

Proof. Due to the fact that G has treewidth k , Lemma 1.3.4 implies that there exists a k -normal tree decomposition (T, B) for G . Choose an arbitrary node r of T as its root. Let w be a node of T with the largest depth such that a vertex $v \in V'$ exists that is contained in the bag of w but not contained in the bag of the parent w' of w . Then $B(T_w) \cap V' \subseteq B(w)$, i.e., $|B(T_w) \cap V'| \leq k + 1$ and $|V' \setminus B(T_w)| \geq 1$. Consequently, $B(w) \cap B(w')$ separates v from at least one vertex in $V' \setminus B(T_w)$. \square

If for example we search for a separator for $V' = \{\text{🇷🇺}, \text{🇩🇪}, \text{🇪🇺}, \text{🇫🇷}, \text{🇮🇹}\}$ in the graph on the right side of Fig. 1.3.1 and if we consider the 3-normal tree decomposition shown in Fig. 1.3.3 with the upper node in the middle taken as the root, then the algorithm of the proof of Lemma 1.3.5 obtains as separator for V' the set $\{\text{🇪🇺}, \text{🇮🇹}, \text{🇪🇺}\}$.

Corollary 1.3.6. *If a graph $G = (V, E)$ has treewidth $k \in \mathbb{N}$, there exists for each subgraph G' of G with at least $k + 2$ vertices a separator of size k in G and thus also in G' .*

Since we can find a tree decomposition (T, B) for each graph G —we simply have to add all vertices of G to a bag of one node of T —we can solve problems on each graph G by a divide-and-conquer approach. However, the smallest instances we have to solve without further recursion and the time for combining smaller instances to a bigger instance depends on the treewidth k of (T, B) . With growing k we have to spend more and more extra time compared to an algorithm on trees. In this sense the treewidth of a graph G is a parameter for measuring how *tree-like* a given problem can be solved on G by a divide-and-conquer approach.

A more structural similarity between trees and arbitrary graphs with their normal tree decompositions can be obtained after defining a generalization of trees called (partial) k -trees.

Definition 1.3.7 ((partial) k -tree). Let $k \in \mathbb{N}$. A k -tree is a graph that can be obtained from a k -vertex clique by zero or more applications of the following rule: Choose a k -vertex clique C in the current graph, add a new vertex v and connect v precisely to the vertices in C . A *partial k -tree* is a subgraph of a k -tree.

Observation 1.3.8. *Each tree is a 1-tree and each 1-tree is a tree.*

We next show that we can obtain from a k -normal tree decomposition a partial k -tree and vice versa. An example of a 2-tree G and a tree decomposition

(T, B) for G of width 2 is shown in Fig. 1.3.4 and 1.3.5, respectively. Note also the similarities in the construction of G and (T, B) described in the caption of the figures. Since each new node creates a new triangle in G , by the similarity mentioned above we can create a bijective mapping from the triangles in G to the nodes of T such that two triangles in G share an edge if and only if their identified nodes are connected in T . We start with two auxiliary lemmata.

Lemma 1.3.9. *If a graph G contains a clique C , any tree decomposition (T, B) of G has a node w of T with $C \subseteq B(w)$.*

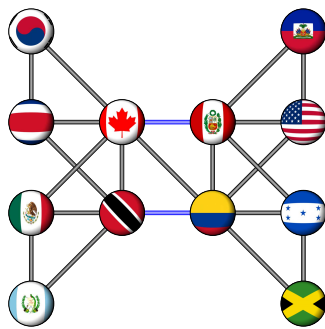


Figure 1.3.4: A 2-tree, where a possible construction is to take Canada 🇨🇦, Peru 🇵🇪, and Colombia 🇨🇴 as the initial clique and to extend it stepwise by USA 🇺🇸, Trinidad and Tobago 🇹🇹, Haiti 🇭🇹, Costa Rica 🇨🇷, South Korea 🇰🇷, Mexico 🇲🇽, Guatemala 🇬🇹, Honduras 🇧🇮, and Jamaica 🇯🇲.

Ignoring the two non-black edges we obtain the matches of the CONCACAF Gold Cup in 2000.

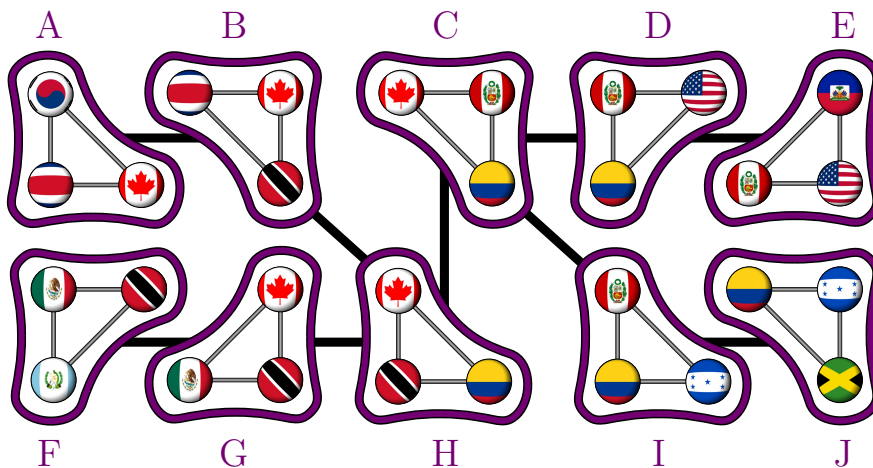


Figure 1.3.5: A k -normal tree decomposition (T, B) of the graph G shown in Fig. 1.3.4. Starting with the node C we can create (T, B) parallel to the construction of G by adding stepwise the nodes D, H, E, B, A, G, F, I , and finally J .

Proof. The lemma can be proved by induction over the cardinality of C . If $|C| \leq 2$, the lemma follows from property (TD1). In the inductive case G contains a clique of size at least 3. Let v_1, v_2 and v_3 be 3 different vertices in C . We know from the induction hypothesis that there are nodes w_1, w_2 and w_3 in T , where $B(w_1) \supseteq C - \{v_1\}, B(w_2) \supseteq C - \{v_2\}, B(w_3) \supseteq C - \{v_3\}$.

Note that in a tree the three unique paths between w_1, w_2 and, w_3 have a common node w^* since otherwise, we have a cycle containing w_1, w_2 and w_3 in the tree. From property (TD2) we can conclude that $B(w_1) \cap B(w_2) \subseteq B(w^*), B(w_2) \cap B(w_3) \subseteq B(w^*),$ and $B(w_1) \cap B(w_3) \subseteq B(w^*),$ i.e., $C \subseteq B(w^*)$. \square

Lemma 1.3.10. *Let $k \in \mathbb{N}$. Graphs with a k -normal tree decomposition are exactly the partial k -trees with at least $k + 1$ vertices.*

Proof. We show first that a graph G with a k -normal tree decomposition (T, B) is a partial k -tree. For the time being, we assume that each pair u, v of vertices contained in a bag of (T, B) is adjacent in G . Start with choosing a node r of T as the root. Take the clique C induced by k arbitrary vertices of $B(r)$ as the initial k -tree and extend it by the $(k + 1)$ -th vertex in $B(r)$ together with its edges to the vertices of C . Then traverse T top-down starting from r . For each node w visited by the traversal, extend the k -tree as described next. Let w' be the parent of w in T and let v be the unique vertex in $B(w) \setminus B(w')$. Since under our assumption $B(w) \setminus \{v\}$ induces a clique in G , we can extend the k -tree by v . After the traversal, we obtain a k -tree that consists of exactly the vertices of G and contains all edges of G . Finally removing the superfluous edges we obtain G as a partial k -tree.

For the converse, we only have to show that each k -tree $G = (V, E)$ has a k -normal tree decomposition (T, B) since, for every partial k -tree $G' = (V', E')$ subgraph of G , the tree decomposition obtained from (T, B) by removing $V \setminus V'$ from all bags is a tree decomposition for G' , which can be made normal (Lemma 1.3.4).

We now use an induction over the number of vertices. Let G be a k -tree and v be the last vertex added in the construction process of a k -tree. In the basic step of the induction, $G = (V, E)$ has $k + 1$ vertices. Thus, G is a $(k + 1)$ -clique. The tree consisting of a single node w and the mapping $B(w) = V$ is a k -normal tree decomposition for G . In the inductive step, let v be the last vertex added last during the construction of G . We then can assume that the graph $G[V - \{v\}]$ has a k -normal tree decomposition (T, B) . Moreover, the neighbors V' of v induce a clique in G , and by the last lemma (T, B) has a node w with $V' \subseteq B(w)$. Thus, we can extend T by a new node w' and an edge $\{w, w'\}$, define $B(w') = V' \cup \{v\}$, and we are done. \square

Let $k \in \mathbb{N}$. Note that every partial k -tree G has treewidth at most k since this is the case if G has at least $k + 1$ vertices by Lemma 1.3.10 and since every graph with at most k vertices has treewidth smaller than k . Conversely, every graph with treewidth $\ell \leq k$ has an ℓ -normal tree decomposition by Lemma 1.3.4 and is therefore by Lemma 1.3.10 a partial ℓ -tree, i.e., a partial k -tree.

Theorem 1.3.11. *Graphs with treewidth at most k are the partial k -trees.*

Recall that the width of a tree decomposition is defined as the maximal size of a bag minus one. The motivation for subtracting one lies in the desired equivalence stated in the next lemma.

Lemma 1.3.12. *Trees are exactly the connected graphs with treewidth 1.*

Proof. For a given tree G , we can construct a tree decomposition (T, B) as follows: Create, for every vertex v of G , a node w for T with $B(w) = \{v\}$ and, for every edge e of G , a node w for T with $B(w) = e$. Finally, connect different nodes w_1 and w_2 of T if $|B(w_1)| = 1 = |B(w_1) \cap B(w_2)|$. Thus, (T, B) is a tree decomposition for G .

For the converse we conclude from Theorem 1.3.11 that if a connected graph G is no tree, i.e. by Obs. 1.3.8, if G is not a 1-partial tree, G does not have treewidth 1. \square

Using Lemma 1.3.4 we can additionally conclude the following:

Corollary 1.3.13. *Every graph $G = (V, E)$ with treewidth $k \in \mathbb{N}$ has a tree decomposition (T, B) of width k such that T has at most $|V| - k$ nodes.*

Proof. The corollary follows from two facts. First, G has a k -normal tree decomposition by Lemma 1.3.4. Second, each k -normal tree decomposition of an n -vertex graph has $n - k$ nodes since by induction over the number x of nodes of a k -normal tree decomposition it becomes obvious that a k -normal tree decomposition with x nodes is a tree decomposition for a graph with $k + x$ vertices. \square

We finish the comparison between trees and tree decompositions with a statement about the number of edges. Note that a tree with n vertices has $n - 1$ edges. Something similar holds for an arbitrary graph: the number of edges is bounded by its number of vertices and its treewidth.

Lemma 1.3.14. *A graph $G = (V, E)$ with treewidth k has $k|V|$ edges.*

Proof. Let (T, B) be a k -normal tree decomposition for $G = (V, E)$ with an arbitrary node r of T chosen as root of T . In addition, let us call in this proof $v \in V$ and $e \in E$ *unseen* with respect to the bag $B(w)$ of a node w of T if $\{v\}$ and e is not a subset of any bag of an ancestors w' of w in T , respectively. For each node w of T , we can observe that if the number of unseen vertices in $B(w)$ is $\ell \in \mathbb{N}$, then there are at most ℓk unseen edges in $B(w)$. Since each vertex is unseen in only one bag, $|E| \leq k|V|$. \square

1.4 Large treewidth

Graphs with small treewidth have a certain similarity to trees. One may ask now if the treewidth of a graph can be arbitrary large. The answer is no since an n -vertex graph has treewidth at most $n - 1$. However, from Lemma 1.3.9 we can conclude that every graph class has arbitrary large treewidth if it contains the n -clique for all $n \in \mathbb{N}$. As a consequence from the next lemma, a graph class \mathcal{G} also has arbitrary large treewidth if, for all $n \in \mathbb{N}$, \mathcal{G} contains a graph that is a supergraph of an n -clique.

Lemma 1.4.1. *If G' is a subgraph of a graph G , the treewidth of G' is at most as large as the treewidth of G .*

Proof. Let (T, B) be a tree decomposition for G of minimal width k . By removing all vertices of G not belonging to G' from the bags of (T, B) , we obtain a tree decomposition for G' of width k or even smaller. \square

In Chapter 3, we search for a tree decomposition for a special graph class called planar graphs.

Definition 1.4.2 (curve, (end)point). A *curve* in \mathbb{R}^2 with *endpoints* $a, b \in \mathbb{R}^2$ is an injective continuous function $f : [0, 1] \rightarrow \mathbb{R}^2$ with $\{f(0), f(1)\} = \{a, b\}$. Moreover, for $0 \leq x \leq 1$, $f(x)$ is called a *point* of f .

Definition 1.4.3 (geometrical embedding, planar graph). A *geometrical embedding* of a graph $G = (V, E)$ is a function ψ that maps V to pairwise disjoint points in \mathbb{R}^2 and every edge $\{u, v\} \in E$ to a curve in \mathbb{R}^2 with endpoints $\psi(u)$ and $\psi(v)$. If additionally, for each pair of edges, their curves have a common point at most in their endpoints, ψ is a *planar embedding*. A graph G is called *planar* if a planar embedding of G exists.

A planar embedding ψ of a graph $G = (V, E)$ divides the points of \mathbb{R}^2 not belonging to the curves of the edges into regions of ψ as follows: Two points lie in the same region if and only if they are endpoints of a curve that has no common point with a curve $\psi(e)$ of an edge $e \in E$. These regions are called the *faces* of ψ . Among all faces of ψ , there is one face not contained in any circle. We call it the *outer face* of ψ . A vertex or an edge is *incident* to a face F and vice versa if each point of its image under ψ is on the boundary of F , i.e., arbitrary close to a point of F . Moreover, a vertex or an edge is *on the boundary* of a face F if it is incident to F and if additionally, for each $\epsilon > 0$ and each point p of its image under ψ , there is a point q not contained in F and the Euclidian distance between p and q is smaller than ϵ .

Not only general graphs, but also planar graphs can have arbitrary large treewidth. However, this is not as easy to see as it is the case for general graphs since planar graphs can not have the 5-clique (or any larger one) as a subgraph (Wagner's theorem). Therefore, we have to define another kind of graphs being planar and having large treewidth.

Definition 1.4.4 ($(\ell_1 \times \ell_2)$ -grid). The $(\ell_1 \times \ell_2)$ -*grid* is a graph that, for $1 \leq i, i' \leq \ell_1$ and $1 \leq j, j' \leq \ell_2$ with $i \neq j$ or $i' \neq j'$, consists of vertices (i, j) and (i', j') , where (i, j) and (i', j') are adjacent if and only if $|i - i'| + |j - j'| = 1$.

If no explicit planar embedding of a grid is given, let us always embed a grid such that a vertex (i, j) has the x -coordinate i and the y -coordinate j and such that each curve of an edge is as short as possible. Obviously, grids are planar; an example of a planar embedding of a grid is shown in Fig. 1.4.1.

Next we show that, for each $\ell \in \mathbb{N}$, the $((\ell+1) \times \ell)$ -grid has treewidth exactly ℓ . With a more complicated proof one can also show that the $(\ell \times \ell)$ -grid has treewidth exactly ℓ . For showing that planar graphs can have arbitrary large treewidth, the weaker result is enough. A tree decomposition for the $((\ell+1) \times \ell)$ -grid of width ℓ —i.e., bags of size $\ell+1$ —is sketched by Fig. 1.4.2.

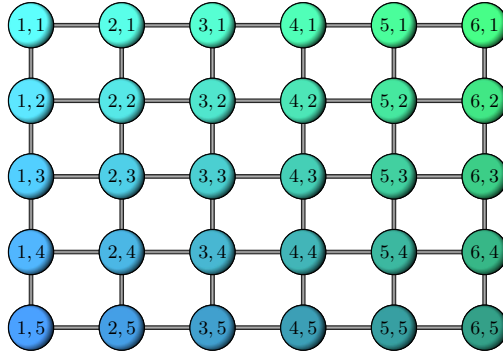


Figure 1.4.1: A (6×5) -grid.

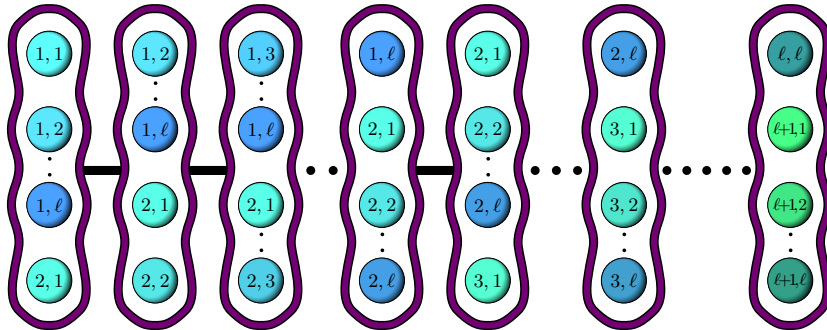


Figure 1.4.2: A sketch of a tree decomposition for an $((\ell+1) \times \ell)$ -grid.

For showing a lower bound of the treewidth of a graph $G = (V, E)$ we use a *one-robber- ℓ -cops-game* [82] in which ℓ cops play against one robber who has to wear a tracking transmitter, i.e., the cops always know his position. (If the robber now feels to get a raw deal, the author regrets the circumstances but rules can not be changed.) Initially the cops and then the robber chooses a starting position. In each round, one cop first lifts its helicopter and makes an announcement—taped by the robber—to which vertex he will fly; then the robber may move along as many edges as he likes and finally the cop lands at his announced position.

The cops win the game if they catch the robber in at most $|V|$ rounds, i.e., during the first $|V|$ rounds the robber and a cop are temporary at the same position. In particular, if X and X' are the vertex sets, where at least one cop stays before and after a round, respectively, for not loosing the game the robber is neither allowed in this round to move through a vertex in $X \cap X'$ nor to stop his move at a vertex X' . The robber wins the game if he is not caught after $|V|$ rounds.

Lemma 1.4.5. *If the one-robber- ℓ -cops-game is played on a graph $G = (V, E)$ with treewidth $k < \ell$, the cops have a winning strategy.*

Proof. For an easier understanding we use Lemma 1.3.4, i.e., we use the fact that G has a k -normal tree decomposition (T, B) and describe a winning strategy

for the cops on (T, B) instead of G . First the cops choose a node w_C of T and distribute themselves to the vertices in $B(w_C)$ —i.e., at least one cop per vertex in $B(w_C)$. Since all cops always stay at the vertices contained in the bag of one node w_C of T , let us say that the cops are at the node w_C . Then, the robber chooses a vertex, say v_R . Take w_R such that $v_R \in B(w_R)$. Since it is not of interest at which vertex of $B(w_R)$ exactly the robber stays, let us say that the robber is at node w_R .

If at the beginning of a round the cops are at node w_C and the robber is at node w_R , let w'_C be the node adjacent to w_C and part of the path from w_C to w_R . Possibly, $w'_C = w_R$. A cop who does not stay at a vertex $B(w_C) \cap B(w'_C)$ lifts his helicopter and announces to move to the only vertex in $B(w'_C) \setminus B(w_C)$. Then the robber may move and finally the cop lands as announced, i.e., at the end of the round, the cops are at node w'_C .

From property (TD1) and (TD2) we can conclude that the robber is never allowed to move from a node w_1 to a node w_2 if the cops are on a node w_C being on the w_1 - w_2 -path. Recall that T has at most $|V| - k$ nodes (Corollary 1.3.13). Consequently, the cops catch the robber after at most $|V| - k$ rounds. \square

Lemma 1.4.6. *There exists a winning strategy for the robber on the one-robber- ℓ -cops-game on the $((\ell+1) \times \ell)$ -grid.*

Proof. The winning strategy for the robber is as follows: The robber goes at the beginning to an arbitrary vertex in a cop-free column. The robber wins the game since he can arrange to stay after each round in a cop-free column if this is the case before the round. In detail, a round starts with the lifting of a cop's helicopter and an announcement of his final position. Let i and i' be the cop-free column at the beginning and at the end of the current round, respectively. Note that during the move of the robber there is additionally to the initially cop-free column i a cop-free row j since one cop is in the air. The robber moves first vertically to (i, j) and then horizontally to (i', j) . Finally the cop moves to his announced position. \square

Corollary 1.4.7. *The treewidth of planar graph classes can be arbitrary high.*

Proof. Let \mathcal{G} be a planar graph class containing, for each $\ell \in \mathbb{N}$, a graph that contains the $((\ell+1) \times \ell)$ grid as a subgraph. Lemma 1.4.1 then implies that the treewidth of \mathcal{G} is not bounded by any constant. \square

Containing a large grid as a subgraph means that there are many vertices with degree 4. By the next lemma we can conclude that even graph classes consisting only of graphs with no vertex of degree larger than 3 can have large treewidth.

Definition 1.4.8 (minor, minor-operation, edge-contraction). A graph G^* is a *minor* of a graph $G = (V, E)$ if G^* can be obtained from G by zero or several *minor-operations* consisting of a vertex removal or an edge removal or a so-called edge-contraction. An *edge-contraction* is the process of merging two adjacent vertices v' and v'' to one new vertex v^* , i.e., removing v' , v'' , and $\{v', v''\}$ as well as adding a new vertex v^* and new edges $\{v^*, u\}$ to all neighbors u of v' or v'' .

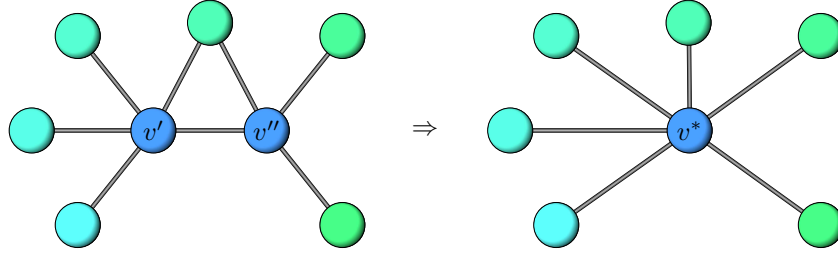


Figure 1.4.3: Edge-contraction of vertices v' and v'' to a new vertex v^* .

For a sketch of an edge-contraction see Fig. 1.4.3. It is not hard to see that a graph obtained from a planar graph by an edge-contraction remains planar, and therefore we get:

Observation 1.4.9. *A minor of planar graph is planar.*

Lemma 1.4.10. *If G is a graph with treewidth k , each minor of G has treewidth at most k .*

Proof. Let G^* be a minor of G . We only need to show that one minor-operation does not increase the treewidth; the lemma then follows by induction. Removing a vertex or an edge clearly does not increase the treewidth, so it remains to consider an edge-contraction. Let H and H^* be the graphs before and after an edge-contraction of v' and v'' to one new vertex v^* . If we replace in a tree decomposition for H all occurrences of v' or v'' by v^* , we obtain a tree decomposition for H^* of the same width or even a smaller width. \square

Corollary 1.4.11. *The treewidth of planar graph classes can be arbitrary high even if we restrict the maximal degree of a vertex to three.*

Proof. Take a planar graph class \mathcal{G} with arbitrary large treewidth— \mathcal{G} exists by Corollary 1.4.7. It suffices to show that each graph G in \mathcal{G} can be replaced by a planar graph G^* with at least the same treewidth as G such that the maximal degree of a vertex in G^* is three.

Let us consider a fixed planar embedding ψ of G . We initially set $G^* = G$ and then modify G^* as long as there is a vertex v in G^* of degree $\deg(v) \geq 4$ as follows: First, draw a circle C with midpoint $\psi(v)$ and a radius such that each edge incident to v intersect C exactly once and no other edge intersects C . Second, iterate over all neighbors u of v and embed a new vertex v_u on the common point of C and the edge $\{v, u\}$. Third, split C into new embedded edges connecting consecutive vertices on C . Finally, replace v and all its incident edges by new edges $\{u, v_u\}$ embedded on a part of the embedded edge $\{u, v\}$ for all $u \in N(v)$.

Note that during the whole process, G is a minor of G^* , i.e., Lemma 1.4.10 implies that the treewidth does not decrease by the construction. Since the degree of new vertices is always bounded by three and since by this construction we also obtain a planar embedding of G^* , the modification above ends with a graph being planar such that each vertex has degree at most three. \square

Chapter 2

Tree Decompositions

2.1 An $O(n \log n)$ -Time Algorithm

Many NP-hard graph problems can be solved on graphs with small treewidth similar to the algorithm described in Section 1.2 solving the maximum independent set problem. Recall that this algorithm traverses a tree decomposition bottom-up. The problem is now that we usually obtain only a graph and we have to find a tree decomposition somehow before we can start the real calculation. Moreover, since the running time of the real computation depends usually exponentially on the treewidth of the used tree decomposition, we should try to find a tree decomposition whose width is very small.

In 1996, Bodlaender [14] solved the problem to find a tree decomposition on n -vertex graphs with treewidth k in $\Theta(f(k) \cdot n)$ time where $f(k) = 2^{\Theta(k^3)}$ as Röhrig showed two years later [80]. The exponential growth in k is no surprise because of the fact that given a graph G and $k \in \mathbb{N}$ it is NP-hard to determine whether G has treewidth k [4].

Definition 2.1.1 (optimization problem, feasible solutions). An *optimization problem* is a tuple $(X, F, Z, \{\max, \min\})$, where

- X is a set of *instances*,
- F is a function that maps each instance $x \in X$ to a set $F(x)$ of *feasible solutions* of x , and
- Z is a function that maps each pair (x, y) , where $x \in X$ and $y \in F(x)$, to a real number $Z(x, y)$ —called the *value* of y .

Definition 2.1.2 (approximation algorithm). An *approximation algorithm* for an optimization problem $P = (X, F, Z, \otimes)$ is an algorithm that computes a feasible solution for each instance.

The only practical algorithms to find a tree decomposition on general graphs with arbitrary treewidth up to now are approximation algorithms.

Definition 2.1.3 (approximation ratio). Let $P = (X, F, Z, \otimes)$ be an optimization problem. If an approximation algorithm A computes for an instance $x \in X$ a feasible solution \tilde{y} and if $OPT(x) = \otimes_{y \in F(x)} Z(x, y)$, then the *approximation ratio* of A for y is

- 1 if $Z(x, \tilde{y}) = OPT(x)$,
- ∞ if either $Z(x, \tilde{y}) = 0$ or $OPT(x) = 0$, and
- $\max \{Z(x, \tilde{y})/Opt(x), Opt(x)/Z(x, \tilde{y})\}$ otherwise.

An approximation algorithm A for P has an *approximation ratio* $\alpha \in \mathbb{R}$ if, on every input $x \in X$, A returns a feasible solution of an approximation ratio bounded by α .

One of the best known algorithms with a constant approximation ratio is Reed's algorithm [76]; his ideas are also used in the remaining section. Given an n -vertex graph with treewidth k his algorithm has, for some large $c \in \mathbb{N}$, a running time of $O(c^k n \min\{n, k! \log n\})$ and finds a tree decomposition of width $4k$, i.e., the algorithm has an approximation ratio of 4. Bodlaender noted in [13] that minor improvements can lower the width of the tree decomposition computed to $3k + 2$. For obtaining a simpler proof, on an n -vertex graph with treewidth k , by Theorem 2.2.9, we show only an $O(3^{3k} k^3 n^2)$ -time algorithm that computes a tree decomposition with a width $4k + 1$ and an $O(2^{16k} n \log n)$ -time algorithm that computes a tree decomposition with a width $8k + 6$.

If we want to find a tree decomposition for a given graph $G = (V, E)$ with treewidth k , as a first approach we might use Corollary 1.3.6, i.e., we would like to use the knowledge that a separator $S \subseteq V$ of size k in G exists. In detail, first search for a separator $S \subseteq V$ of size k in G and divide $V \setminus S$ into two disjoint subsets X and Y such that there is no edge between a vertex in X and a vertex in Y . Then, create a root r of a tree decomposition (T, B) for G and define $B(r) = S$. Next, do the same recursively for the two graphs $G[X \cup S]$ and $G[Y \cup S]$. Finally, connect r with the roots of the tree decompositions obtained recursively for $G[X \cup S]$ and for $G[Y \cup S]$.

However, this approach does not work because a vertex $v \in S$ occurs possibly not in the bag of the root in the tree decomposition obtained for $G[X \cup S]$ (or $G[Y \cup S]$) but in some other bag. Thus, the second property (TD2) for tree decompositions fails for v . See also Fig 2.2.1.

As a consequence, if we want to use an approach similar to the one above, we need the following special tree decomposition: For a graph $G = (V, E)$ and a set of vertices $X \subseteq V$, an *X-tied* tree decomposition is a rooted tree decomposition (T, B) such that every vertex in X occurs in the bag of the root of T . If a tree decomposition (T, B) for a graph G is now obtained by first creating a root r and defining $B(r) = S$ for a set S separating two sets $V_1, V_2 \subset V$ with $V_1 \cup V_2 = V$ and second by connecting r to the roots of the trees of two S -tied tree decompositions (T_1, B) and (T_2, B) for $G[V_1 \cup S]$ and $G[V_2 \cup S]$, respectively, then we can conclude the following. Property (TD1) is true by construction. Since property (TD2) _{v} is true in (T_1, B) and in (T_2, B) for each vertex $v \in S$ and since S is part of the bags of r and part of both roots of T_1 and T_2 , property (TD2) _{v} is also true in (T, B) for each vertex $v \in S$. Property (TD2) _{v} is also true in (T, B) for each vertex $v \notin S$ since these vertices occur either in (T_1, B)

or in (T_2, B) . Thus, (TD2) is true for (T, B) . Before describing the construction in more detail, let us define some special kind of separators.

Definition 2.1.4 (*X-separator, balanced, weak, weighted*). Given a graph $G = (V, E)$, a vertex set $X \subseteq V$, and a weight function $w : X \rightarrow \mathbb{N}$ a *weighted X-separator of size k* is a separator S in G with $|S| \leq k$ such that we can partition the connected components of $G[V \setminus S]$ into two sets \mathcal{C}_1 and \mathcal{C}_2 such that the *weight condition* holds: The total weight of all vertices contained in both X and in a component of \mathcal{C}_i ($i = 1, 2$) is at most $\max(k, W)$, where $W = \lfloor 2/3 \sum_{v \in X} w(v) \rfloor$.

Moreover, S is called a *weak weighted X-separator* if we replace in the weight condition the value of W by $\lfloor 19/24 \sum_{v \in X} w(v) \rfloor$.

An (*unweighted*) *X-separator* is a weighted *X-separator*, where no weight function is given and where we assume implicitly $w(v) = 1$ for all $v \in X$.

If, for a separator S , every component C of $G[V \setminus S]$ consists of at most $\lfloor 11/12 |V| \rfloor$ vertices, S is called *balanced*.

Let G be a given graph with treewidth $k \in \mathbb{N}$. As we see in the following a tree decomposition (T, B) for G can then be constructed by calling $\text{TD}_k(G, \emptyset)$, where TD_k denotes the following function.

```

( $T, B$ ) function  $\text{TD}_k(\text{graph } G = (V, E), \text{vertex set } X)$ 
[01]   tree  $T = \text{new tree}()$ ; // empty tree
[02]   mapping  $B = \text{new mapping}()$ ;
[03]   node  $r = \text{new node}()$ ; define  $r$  as the root of  $T$ ;
[04]   if  $|V| \leq 4k + 2$  then set  $B(r) = V$  and return  $(T, B)$ ;
[05]   let  $S$  be an  $X$ -separator of size  $k + 1$  in  $G$ ;
[06]   set  $B(r) = (S \cup X)$ ;
[07]   for each connected component  $C = (V_C, E_C)$  in  $G[V \setminus S]$  do
[08]      $(T_C, B_C) = \text{TD}_k(G[V_C \cup S], (X \cap V_C) \cup S)$ ;
[09]     connect the root of  $T_C$  with  $r$ ;
[10]     define  $B(w) = B_C(w)$  for all nodes  $w$  of  $T_C$ ;
[11]   return  $(T, B)$ ;

```

Since initially the function TD_k is called with \emptyset as second parameter, by induction over the recursion depth we can conclude from the properties of an X -separator of size $k + 1$ that TD_k is called recursively such that the cardinality of the second parameter is always at most $\lfloor 2/3(3k + 1) \rfloor + (k + 1) = 3k + 1$. Therefore, the cardinality of the bags obtained is bounded by $(3k + 1) + (k + 1) = 4k + 2$, and we obtain a tree decomposition of width $4k + 1$.

A tree decomposition can be found more efficiently for some graphs by replacing line [05] by

```

[05*]   let  $S$  be a balanced  $X$ -separator of size  $2k + 2$  in  $G$ .

```

We call this modified version the *balanced version* of TD_k . For the balanced version, we can observe that the second parameter is during all recursive calls at most $\lfloor 2/3(6k + 4) \rfloor + (2k + 2) = 6k + 4$, i.e., we obtain a tree decomposition of width $(6k + 4) + (2k + 2) - 1 = 8k + 6$.

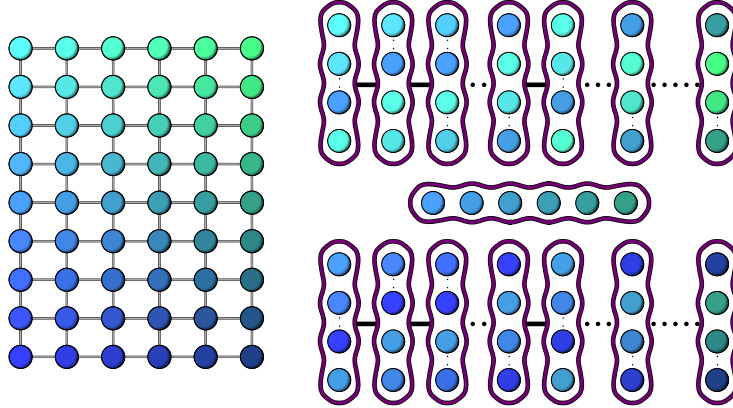


Figure 2.2.1: The left side shows a (6×9) -grid, where we take, e.g., the horizontally middle vertices as a separator S —shown on the right side in the middle. Thus, we obtain two (6×5) -grids. The remaining parts on the right side show a schematic tree decomposition—constructed as described on page 20—for each of the two (6×5) -grids. Observe that each cycle-free connection of these two tree decompositions violates $(TD2)_v$ for at least one vertex $v \in S$.

It remains to show that, for the standard version, an X -separator of size $k + 1$ and, for the balanced version, a balanced X -separator of size $2k + 2$ exists in each recursive call of the algorithm and how to find it.

2.2 Finding an X -separator

Lemma 2.2.1. *Let $G = (V, E)$ be a graph with treewidth $k \in \mathbb{N}$ and with at least $3k$ vertices. For each set $X \subseteq V$, there exists an X -separator of size at most $k + 1$ in G .*

Proof. If $|X| \leq k$, every separator in G of size at most $k + 1$ works and we can use Corollary 1.3.6. Moreover, if $|V| \leq 3$ or $k = 1$, one can easily observe that the lemma holds. We now assume that the first two cases do not apply, i.e., in particular $|X| \geq 3$. Let (T, B) be an r -rooted, k -normal tree decomposition of G such that r is a node of T with degree at most 1. Traverse T downwards always going to a node w with $|X \cap B(T_w)|$ being maximal until the last node w with $|X \cap B(T_w)| \geq \lceil |X|/2 \rceil$ is reached, i.e., $|X \cap B(T_{w'})| \leq \lceil |X|/2 \rceil - 1 \leq \lfloor |X|/2 \rfloor$ for each child w' of w . Note that w is not r because (T, B) is a k -normal tree decomposition and in such a tree decomposition, there exists only one vertex in $B(r)$ not occurring in the bag of the only adjacent vertex of r . Additionally, by construction $|X - B(T_w)| \leq \lfloor |X|/2 \rfloor$. If w is no leaf in T , $B(w)$ is an X -separator of size $k + 1$ in G since one can easily partition the connected components of $G[V \setminus B(w)]$ into two sets \mathcal{C}_1 and \mathcal{C}_2 such that the weight condition of an X -separator holds. Otherwise, $B(w) \cap B(w^*)$ with w^* being the parent of w is an X -separator of size k in G . \square

We next consider the problem of finding an X -separator of size $k + 1$, where we start with a description of a standard algorithm for exploring a graph. Given a directed or an undirected graph G , a *depth-first search*—also abbreviated as

DFS—is an algorithm for traversing G until a goal vertex is reached or all vertices of G are considered. The algorithm starts its search at some vertex r of G by setting a pointer p on r and then moves repeatedly p along edges—in a directed graph always from tail to head—to so-called unseen vertices. A vertex is called *unseen* if up to now p has never pointed to it. The algorithm then either finds a certain goal vertex or hits a vertex that has no unseen neighbors. In the first case, the algorithm stops. In the second case, the algorithm backtracks and returns p to the most recent pointed vertex that is adjacent to an unseen vertex v and continues the search from v . If finally the algorithm can not move to any further unseen vertices that are adjacent to a seen vertex and if there are still unseen vertices, the algorithm starts again from an unseen vertex.

In other words, first mark all vertices as unseen. Then as long as there is an unseen vertex r we *start a DFS* from vertex r by calling the procedure $DFS(G, r, goal)$, where $goal$ is either a vertex of G or $goal = null$. In the following, we see a description of a DFS in pseudo code, which is extended by some additional information stored in variables d, f , and F . These variables are not necessary for traversing a graph or finding an X -separator, but are useful in later applications of a DFS. Note that the pointer p can be defined implicitly by pointing p always to the topmost vertex on S .

```

// global variables:
stack S =new stack(); // realized by an empty list
int t = 0; // time counter for obtaining useful extra data during a DFS
node_array<int> d(G), f(G);
graph F=new graph(); // initially an empty forest
procedure DFS(graph G, vertex v, vertex goal)
// DFS starts from v and browses G (until goal is found)
[01]   S.push(v); // add v on the top of S
[02]   d[v] = t++; // set discovery time of v
[03]   if (v == goal) print "path: ", S.toString(); stop;
[04]   for all vertices u adjacent to v do
[05]     if u is unseen then
[06]       add vertex u and edge {u, v} to F;
[07]       mark u as seen;
[08]       call DFS(G, u, goal); // u gets a child of v
[09]   S.pop(); // removes v from the end of S
[10]   f[v] = t++; // set finishing time of v

```

It is not hard to see that a DFS in a graph $G = (V, E)$ can be done in $O(|V| + |E|)$ time. Moreover, for a fixed DFS that was started from vertices r_1, \dots, r_x ($x \in \mathbb{N}$), we call the obtained forest F with roots r_1, \dots, r_x a *DFS-forest* for G and, if $x = 1$, also a *DFS-tree* for G . Additionally, the edges of F are called *tree edges*.

Instead of describing a procedure for finding an X -separator of size $k + 1$, we first construct an algorithm that finds—if possible— $k + 2$ edge-disjoint paths all connecting a vertex s and a vertex t and use this algorithm to find—if possible— $k + 2$ internal vertex-disjoint paths all connecting a vertex s' and a vertex t' . If the latter algorithm fails, we will see that as a byproduct we obtain an X -separator.

Definition 2.2.2 (residual graph). The *residual graph* of a directed graph $G = (V, E)$ and a path P_1 with edges $E' \subseteq E$ consists of the vertex set V and the edge set $E \setminus E' \cup \{(v, u) \mid (u, v) \in E'\}$. Moreover, the *residual graph* of G and $\ell \in \{2, 3, \dots\}$ paths P_1, \dots, P_ℓ is defined inductively as the residual graph of G^* and P_ℓ , where G^* is the residual graph of G and $P_1, \dots, P_{\ell-1}$.

Note explicitly that the definition of a residual network of a graph G and path P_1, \dots, P_ℓ does not mean that P_1, \dots, P_ℓ are path in G .

Lemma 2.2.3. Let $G = (V, E)$ be a directed graph with no cycle of length 2, let $s, t \in V$ with $s \neq t$, and let $k \in \mathbb{N}$. In $O(k(|V| + |E|))$ time, we can either find k edge-disjoint paths P_1, \dots, P_k from s to t in G or at most $k - 1$ edges whose removal disconnects s and t .

Proof. The main idea is to search for an s - t -path k times using a DFS in G . Unfortunately, this approach does not work in general since already one path may block any other s - t -path in G . For an example, see the graph G with 4 s - t -paths in Fig. 2.2.2. In Fig. 2.2.3, a first s - t -path P is found in G . While we search for a further path, we must get rid of the blocking caused by P . In Fig. 2.2.4 we see the residual graph G^* of G and P , where 3 s - t -paths still exist. A key observation is that having found a path P' in G^* we can obtain two paths in G by taking all edges (u, v) in G where (u, v) and (v, u) are in total used an odd number of times by P and P' .

Generalizing this approach we first search for a path P_1 in $G_0 = G$, and then, for each $i \in \{2, \dots, k\}$, we search for a path P_i by a DFS in the residual graph G_{i-1} of G and $\{P_1, \dots, P_{i-1}\}$. If we have finally found k paths (each in a different graph), we can easily describe a construction of k edge-disjoint paths in G after collecting some more properties.

Let E^* be the multiset of edges being part of the k paths. Since we constructed only s - t -paths, each vertex $v \notin \{s, t\}$ is incident to an even number of edges in E^* . From the construction of the residual graphs we can conclude that each edge (u, v) of G is used at most once more than (v, u) , and (v, u) is used at most as often as (u, v) is used. Let us say that an edge (u, v) of G is a

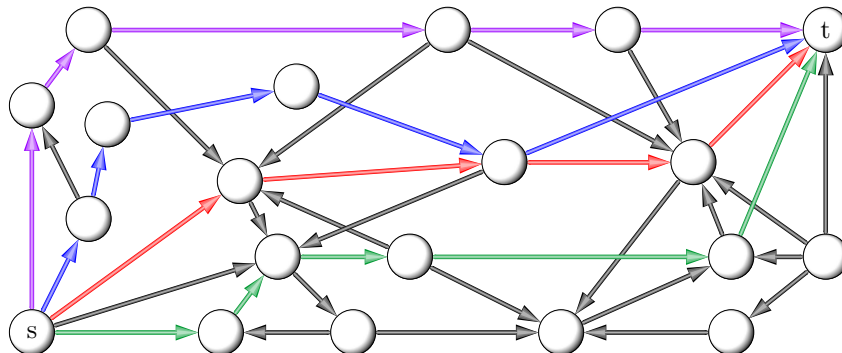


Figure 2.2.2: A graph G and 4 s - t -paths.

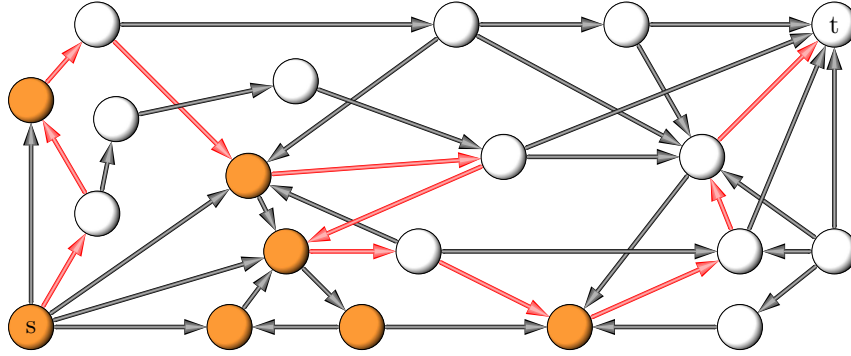


Figure 2.2.3: In the graph G of Fig. 2.2.2 one s - t -path P —shown by the non-black edges—blocks all other s - t -paths. Observe that the white vertices are reachable from s only by the use of one or more edges of P .

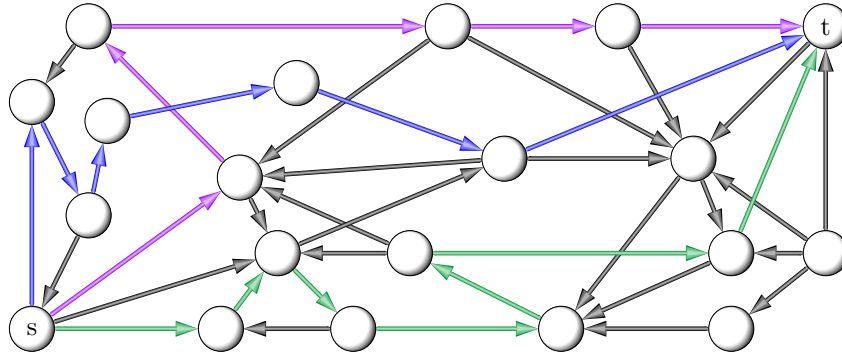


Figure 2.2.4: The residual graph G^* of the graph G and the path P shown in Fig. 2.2.3. Note that after the 'removal' of one path P in G exactly 3 of 4 s - t -paths in G remain in a modified version in G^* .

canceled edge if the k paths P_1, \dots, P_k use (u, v) as often as (v, u) . If E' is the set of non-canceled edges, we can observe that every vertex $v \notin \{s, t\}$ has an even degree in the graph (V, E') . Also note that the number of edges with head s minus the number of edges with tail s is exactly k in E^* and thus in E' .

For constructing k edge-disjoint paths in G we can start in (V, E') a DFS from s and reach t without backtracking. The visited vertices and edges define our first path P . After removing the edges of P from E' we start another DFS as long as s is incident to an edge, in other words, as long as we have not constructed k edge-disjoint s - t -paths in G . Since the whole algorithm consists basically only of k DFS for constructing P_1, \dots, P_k and further k DFS for constructing the disjoint path in G , it runs in the promised time.

It remains to show that there are indeed no k s - t -paths if the algorithm fails, i.e., if, for some $0 \leq \ell < k$, G_ℓ has no path from s to t . Let C be the connected component of G_ℓ containing s but not t . For each $1 \leq i \leq \ell$, G_i has one more edge in $C \times (V \setminus C)$ than G_{i-1} has since P_i has one more edge in $C \times (V \setminus C)$ than in $(V \setminus C) \times C$. Consequently, G has exactly ℓ being edges part of $C \times (V \setminus C)$ and G does not contain $\ell + 1$ or even more edge-disjoint path from s to t . Thus, removing these ℓ edges we disconnect s and t . \square

Given a graph $G = (V, E)$ and two non-adjacent vertices $s, t \in V$ with $s \neq t$, we can find k paths from s to t that pairwise have no internal vertices in common by first transforming G into the directed graph G' with $V' = \{v', v'' \mid v \in V\}$ and $E' = \{(u'', v'), (v'', u') \mid \{u, v\} \in E\} \cup \{(v', v'') \mid v \in V\}$ and then by finding in G' k paths from s'' to t' that are pairwise edge disjoint. We say that G' is the *vertex-disjoint-to-edge-disjoint version* of G . An example of the transformation is shown in Fig. 2.2.5. It is not hard to see that G has k pairwise internal vertex-disjoint paths from s to t if and only if G' has k pairwise edge-disjoint paths from s'' to t' .

If we have found k edge-disjoint paths from s'' to t' in G' , each encountered path P' defines a path P in G by applying the *backward vertex transformation* from G to G' : Replace each vertex v' as well as v'' in P' by v and finally remove duplicate occurrences. Additionally, if no solution is found in G' , then there are $\ell < k$ edges $E^* = \{e_1, \dots, e_\ell\}$ whose removal in G' disconnects s and t . Since there is no edge from s to t in G , we can take an endpoint $v \notin \{s, t\}$ of each edge in E^* and add it to an initially empty set. If S is the vertex set obtained, S is a separator of size at most ℓ for s'' and t' in G' . Again applying the backward vertex transformation, i.e., replacing each vertex v' as well as v'' in S by v , we obtain a separator of size at most ℓ for s and t in G .

Note that both, the transformation from G to G' and the computation of a solution in G from a solution in G' takes only $O(k(|V| + |E|))$ time. Using Lemma 2.2.3 we therefore obtain the next lemma.

Lemma 2.2.4. *Given a graph $G = (V, E)$, two non-adjacent vertices $s, t \in V$ with $s \neq t$, and $k \in \mathbb{N}$, we find in $O(k(|V| + |E|))$ time either k internal-vertex-disjoint s - t -paths or at most $k - 1$ vertices whose removal disconnects s and t .*

A slight generalization of the last lemma is stated in the next corollary.

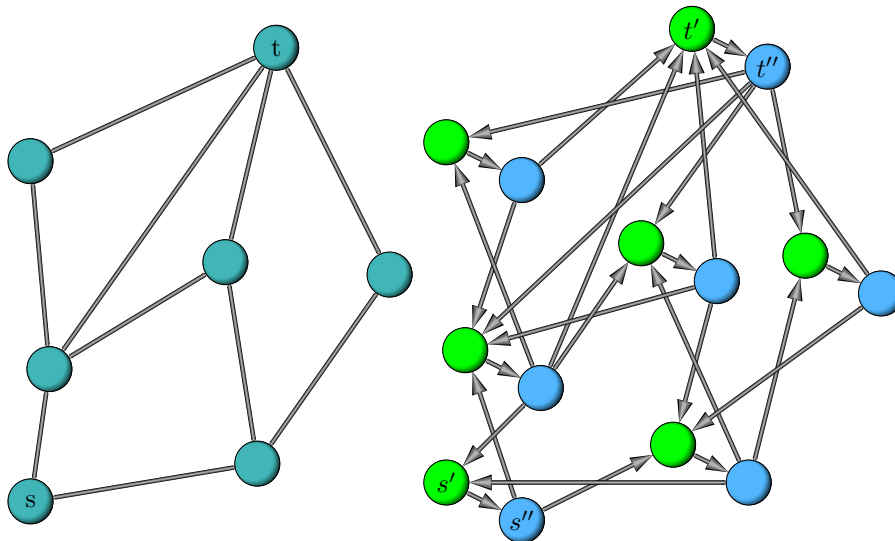


Figure 2.2.5: Vertex-disjoint s - t -paths in the left graph can be found by finding edge-disjoint s'' - t' -paths in the right graph.

Corollary 2.2.5. Given a graph $G = (V, E)$, $k \in \mathbb{N}$, and two sets $S, T \subset V$ such that $S \cap T = \emptyset$ and there is no edge $\{u, v\} \in E$ with $u \in S$ and $v \in T$, we find in $O(k(|V| + |E|))$ time either k internal-vertex-disjoint paths each with endpoints in both, S and T or at most $k-1$ vertices in $V \setminus (S \cup T)$ whose removal disconnects S and T .

Proof. Connect all neighbors of a vertex in S to a new vertex s and all neighbors of a vertex in T to a new vertex t . See Fig. 2.2.6 for an example. Then remove all vertices $S \cup T$ with its incident edges and search for k internal-vertex-disjoint paths from s to t in the graph obtained. \square

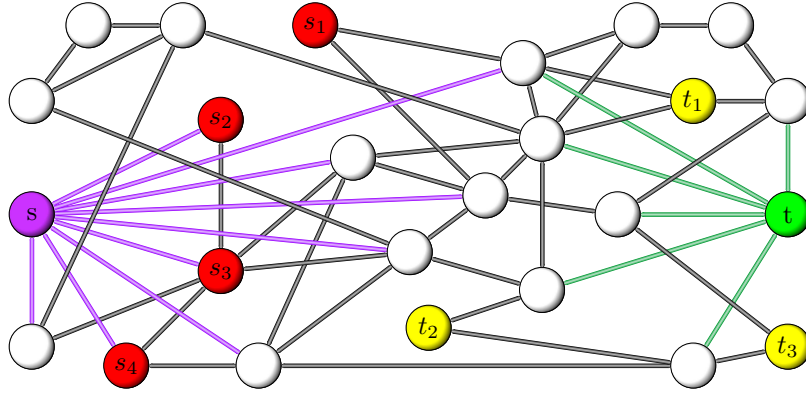


Figure 2.2.6: A graph G with two new vertices s and t connecting the neighbors of $S = \{s_1, \dots, s_4\}$ and $T = \{t_1, \dots, t_3\}$, respectively.

Now we can describe our algorithm for finding an X -separator.

Lemma 2.2.6. Let $G = (V, E)$ be a graph with treewidth $k \in \mathbb{N}$, let $X \subseteq V$, and let $w : X \rightarrow \mathbb{N}$. If a kind of X -separator (weak or non-weak, weighted or unweighted) of size k exists in G , then it can be found in $O(3^{|X|} \cdot k^2 \cdot |V|)$ time.

Proof. By the assumption there exists an X -separator S that separates a set $X_1 \subset X$ from $X_2 \subset X$. Since we do not know X_1 and X_2 , we simply iterate over all possibilities to partition X into the three sets X_1, X_2 and X_S with $|X_S| \leq k$ such that—depending on the kind of X -separator—the following holds.

- *unweighted*: $|X_1|, |X_2| \leq \max(k, \lfloor 2/3|X| \rfloor)$,
- *weak unweighted*: $|X_1|, |X_2| \leq \max(k, \lfloor 19/24|X| \rfloor)$,
- *weighted*: $\sum_{v \in X_1} w(v), \sum_{v \in X_2} w(v) \leq \max(k, \lfloor 2/3 \sum_{v \in X} w(v) \rfloor)$, and
- *weak weighted*: $\sum_{v \in X_1} w(v), \sum_{v \in X_2} w(v) \leq \max(k, \lfloor 19/24 \sum_{v \in X} w(v) \rfloor)$.

In each iteration, try to find a separator S of size k such that

1. there is no path in $G[V - S]$ from a vertex in X_1 to a vertex in X_2 ,
2. $S \cap (X_1 \cup X_2) = \emptyset$, and
3. $|X_S| \leq k$.

In other words, we try to find a set S' of size at most $k - |X_S|$ separating X_1 and X_2 in $G[V - X_S]$. For fixed sets X_1 and X_2 , by Corollary 2.2.5 the search for S' can be done in $O(k(|V| + |E|)) \stackrel{\text{L. 1.3.14}}{=} O(k^2|V|)$ time. \square

The combination of Lemma 2.2.1 and 2.2.6 implies the following.

Corollary 2.2.7. *Let k and $n \geq 3k$ be in \mathbb{N} . For each n -vertex graph $G = (V, E)$ with treewidth k and each set $X \subseteq V$, an X -separator of size $k + 1$ in G can be found in $O(3^{|X|} \cdot k^2 \cdot n)$ time.*

For the balanced version of TD_k , it remains to show the existence of a balanced X -separator and how we can find it.

Lemma 2.2.8. *Let k and $n \geq 3k$ be in \mathbb{N} . If $G = (V, E)$ is an n -vertex graph with treewidth k , for each set $X \subseteq V$, we can find in G a balanced X -separator of size $2k + 2$ in $O((3^{|X|} + 2^{15.7k})k^2 \cdot n)$ time.*

Proof. First, let us try to find a separator S_1 of size $k + 1$ such that each component in $G[V - S_1]$ contains at most $\lfloor 11n/12 \rfloor$ vertices. A first idea might be to take a V -separator of size $k + 1$ as S_1 since it fits the matter and since it exists by Lemma 2.2.1. Using Lemma 2.2.6 we can find S_1 in $O(3^n n^3)$ time—unfortunately this is too slow.

For improving the running time, the idea is to determine a set R of $O(k)$ vertices called representatives that are spread—in some sense—smoothly in G such that each set being a weak weighted R -separator in G is a separator S_1 with the properties defined above. In detail, we determine in the following a function f , a set R of at most $8(k + 1)$ vertices, and a partition \mathcal{P} of $V \setminus R$ into sets P_1, \dots, P_ℓ all of size at most $\lfloor n/(8(k + 1)) \rfloor$ such that each set P of \mathcal{P}

- induces a connected component in G and
- is mapped by f to one vertex in R that has a neighbor in P .

For each $P \in \mathcal{P}$, each $v \in P$, and each $r \in R$, define $f(P)$ as the representant of v , and define r as a representant for itself. In addition, for each vertex $r \in R$, set the weight $w(r) = |\{v \in V \mid r \text{ representant for } v\}|$. Note that the total weight over all vertices in R is exactly n .

R and \mathcal{P} can be found as follows: W.l.o.g. we assume that G is connected. Otherwise, as long as G contains more than two connected components, connect them by an edge. Note that this construction neither increases the treewidth nor the asymptotical size of the graph. First, construct a DFS-tree T for G . Then traverse T bottom-up and store for each vertex v in T the number of its descendants in T . If this number exceeds $\lfloor n/(8(k + 1)) \rfloor$, cut T into two parts by removing the edge between v and its parent, which now becomes a leaf. Each root of a tree obtained is a representative r —by the construction we have at most $8(k + 1)$ representatives—and, for each child v of r , define the set of the vertices in T_v as one set of \mathcal{P} , which is thus of size at most $\lfloor n/(8(k + 1)) \rfloor$.

For the time being let us assume that a V -separator S of size $k + 1$ exists with $S \cap R = \emptyset$. Let \mathcal{C}_1 and \mathcal{C}_2 be a partition of the connected components of $G[V \setminus S]$ such that the weight condition of the V -separator S holds. Let $V(\mathcal{C}_i)$ ($i = 1, 2$) be the union of the vertices over all connected components of \mathcal{C}_i . A key observation is now that in the graph $G[V - S]$ almost all vertices of V are in the same connected component as their representants. More exactly, S can

disconnect vertices of at most $k+1$ sets $P \in \mathcal{P}$, i.e., in total $\lfloor \frac{n}{8(k+1)} \rfloor (k+1) \leq \lfloor \frac{n}{8} \rfloor$ vertices, from their representatives. For $i=1, 2$, thus the observations below hold:

Obs. 1: $|V(\mathcal{C}_i)| - \sum_{r \in R \cap V(\mathcal{C}_i)} w(r) \leq \lfloor \frac{n}{8} \rfloor$

Obs. 2: $\sum_{r \in R \cap V(\mathcal{C}_i)} w(r) \leq |V(\mathcal{C}_i)| + \lfloor \frac{n}{8} \rfloor \leq \lfloor \frac{2n}{3} + \frac{n}{8} \rfloor = \frac{19}{24}n = \frac{19}{24} \sum_{r \in R} w(r)$,
i.e., S is a weak weighted R -separator.

Moreover, by Lemma 2.2.6 a weak weighted R -separator S_1 can be found in $O(3^{8k} k^2 \cdot n)$ time, and Obs. 1 together with Obs. 2 implies that S_1 satisfies the extra condition for balanced X -separators, i.e., each component in $G[V \setminus S_1]$ contains at most $\lfloor n/8 \rfloor + \lfloor 2n/3 + n/8 \rfloor \leq \lfloor 11n/12 \rfloor$ vertices. Let $X \subseteq V$. If we also compute an X -separator S_2 of size $k+1$, then $S_1 \cup S_2$ is a balanced X -separator of size $2k+2$.

Unfortunately, the idea from above only works if a V -separator S of size $k+1$ exists with $S \cap R = \emptyset$. As we already know, a V -separator S of size $k+1$ exists in G . Consequently, if the algorithm above fails, we can conclude that there exists a vertex $r \in R$ that is part of a V -separator. Thus, we can iterate over all $r \in R$ and assume in each iteration that r is part of our V -separator S ; therefore we can remove r from G and search recursively for a V -separator of size k instead of $k+1$ in the graph $G[V \setminus \{r\}]$.

Finally note that by Corollary 2.2.7 an X -separator of size $k+1$ can be found in $O(3^{|X|} \cdot k^2 \cdot n)$ time, and the running time of the algorithm for finding a weak weighted R -separator is bounded by $T(n, k+1)$, where

$$T(n, k+1) \leq \begin{cases} c \cdot 3^{8k} k^2 n + 8(k+1) \cdot T(n, k) & \text{if } k > 0 \\ c & \text{otherwise} \end{cases}$$

i.e., $T(n, k+1) = O(3^{8k} k^2 n) = O(2^{15.7k} k^2 n)$. \square

Concerning the running time of $\text{TD}_k(G = (V, E), \emptyset)$, we can observe the following. Since each instance in one recursion depth can be assigned to a vertex $v \in V$ that occurs only in this instance and since each instance contains only $O(k)$ vertices that occur in other instances of the same recursion depth, the total size of all instances of one recursion depth is $O(k \cdot n)$.

In the standard version, i.e., using an arbitrary X -separator we can observe that, for each recursive call, the size of the instance shrinks by at least one vertex. Thus, we have a recursion depth of at most n . In the balanced version the size n' of an instance shrinks in each recursive call by a constant factor $\leq (\lfloor 11n'/12 \rfloor + (2k+2))/n' \leq 29/30$ as long as $n' \geq 100k$, i.e., we can bound the recursion depth by $O(k + \log n)$. We therefore can conclude the following.

Corollary 2.2.9. *Given a graph $G = (V, E)$ with treewidth k we find a (rooted) tree decomposition of width $4k+1$ and $8k+6$ in $O(3^{3k} k^3 \cdot n^2)$ and $O(2^{15.7k} k^3 \cdot n \cdot (k + \log n)) = O(2^{16k} \cdot n \log n)$ time, respectively.*

By Lemma 1.3.4 we also can find a (rooted) $(4k+1)$ -normal and $(8k+6)$ -normal tree decomposition, respectively, in the times mentioned above. Moreover, if we want to obtain a binary tree decomposition (T, B) , do the following as long as a node of T has too many children: Iteratively replace a node w with $\ell > 2$ children by a path P of length $\ell - 2$, define $B(w') = B(w)$ for each node w' of P , connect an arbitrary node of P to the parent of w , and distribute the children of w to the nodes of P such that each node of P has exactly 2 children.

2.3 Clique Trees

Before describing an algorithm for constructing a clique tree for a chordal graph we need some more properties of chordal graphs.

Lemma 2.3.1. *If $G=(V,E)$ is a chordal graph, for every $U \subseteq V$, the graph $G[U]$ is also chordal.*

Proof. Assume $G' = G[U]$ is not chordal, i.e., G' contains a chordless cycle C of length at least 4. Let V^* be the set of vertices of C . Since each edge $e \subseteq V^*$ is in G if and only if it is in G' , C is also a chordless cycle in G . Contradiction. \square

Definition 2.3.2 (perfect elimination order, successors). The *successors* of a vertex v with respect to an order L are the vertices that are adjacent to v and that appear behind v in L . An ordering L of the vertices v_1, v_2, v_3, \dots of a graph G is a *perfect elimination order* if each vertex v induces together with its successors a clique in G .

Lemma 2.3.3. *Every graph with a perfect elimination order is chordal.*

Proof. Let $G = (V, E)$ be an n -vertex graph with a perfect elimination order L . Assume for contradiction that G is not chordal, i.e., G contains a set V^* of $\ell \geq 4$ vertices that induce a chordless cycle C in G . Let v be the first vertex in L being part of V^* and let v' and v'' be the two neighbors of v in C . Since C is a chordless cycle of length at least 4, v' and v'' are not adjacent in G ; however, this contradicts L being a perfect elimination order. \square

Definition 2.3.4 (rift, height). Let $G = (V, E)$ be an n -vertex graph and L be an order v_1, \dots, v_n of V . A *rift* in G with respect to L is a chordless path P of length at least 2 such that the endpoints of P appear behind all inner vertices of P with respect to L . A v_i - v_j -*rift* is a rift P from v_i to v_j and the *height* of P is $\max(i, j)$.

Lemma 2.3.5. *On a chordal graph G , let L be the order of G computed by the algorithm below. Then, G does not contain a rift with respect to L .*

```

list function findOrder(graph  $G = (V, E)$ )
[01]   list  $L = \text{new list}()$ ; // empty list
[02]   for  $i = 1$  to  $|V|$  do
[03]     choose  $v \notin L$  such that no other vertex  $v' \notin L$ 
                                     has more neighbors than  $v$  in  $L$ ;
[04]     add  $v$  in front of  $L$ ;
[05]   return  $L$ ;

```

Proof. Take $G = (V, E)$. Let $v_1, v_2, \dots, v_{|V|}$ be the vertices of G ordered with respect to L . Assume for contradiction that the lemma is wrong and $P = (v_{\sigma_1}, v_{\sigma_2}, \dots, v_{\sigma_\ell})$ is a rift such that $\sigma_1 < \sigma_\ell$ and such that its height is maximal with respect to all rifts in G . See also Fig. 2.3.1. Since $\sigma_{\ell-1} < \sigma_1$ (P is a rift), i.e., the findOrder algorithm has chosen v_{σ_1} before $v_{\sigma_{\ell-1}}$ and since $v_{\sigma_{\ell-1}}$, but not v_{σ_1} , belongs to the neighbors $N(v_{\sigma_\ell})$ of v_{σ_ℓ} in G , there has to

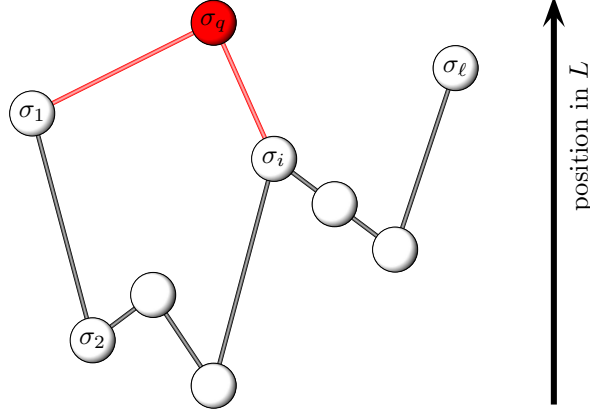


Figure 2.3.1: The rift P of the proof of Lemma 2.3.5 is shown by the white vertices. The values in the vertices are their indices with respect to ordering L .

be a vertex $v_q \in N(v_{\sigma_1}) \setminus N(v_{\sigma_{\ell-1}})$ with $q > \sigma_{\ell}$. Let v_{σ_i} be the last vertex of P before $v_{\sigma_{\ell-1}}$ that is adjacent to v_q in G , possibly $\sigma_i = \sigma_1$. Since P is a rift, $v_{\sigma_i}, \dots, v_{\sigma_{\ell}}$ induce a chordless path in G and by the definition of i all vertices $v_{\sigma_{i+1}}, \dots, v_{\sigma_{\ell-1}}$ are not adjacent to v_q . As a consequence from G being chordal v_q and $v_{\sigma_{\ell}}$ are also not adjacent—otherwise $v_q, v_{\sigma_i}, \dots, v_{\sigma_{\ell-1}}, v_{\sigma_{\ell}}$ is a chordless cycle of length at least 4. Therefore, $v_{\sigma_{\ell}}, v_{\sigma_{\ell-1}}, \dots, v_{\sigma_i}, v_q$ is a rift with $\sigma_{\ell} < q$ and height $q > \sigma_{\ell}$; a contradiction to our choice of P . \square

The algorithm `findOrder` and also the next theorem is based on ideas of Tarjan and Yannakakis [91].

Theorem 2.3.6. *Chordal graphs are exactly the graphs with a perfect elimination order.*

Proof. By Lemma 2.3.3 it remains to show that every chordal graph has a perfect elimination order. Assume for contradiction that the algorithm `findOrder` computes on a chordal graph an order L , which is not a perfect elimination order. Then there exists a vertex v having two successors with respect to L that are not adjacent, i.e., G contains a rift with respect to L ; a contradiction. \square

Lemma 2.3.7. *An efficient implementation of the algorithm `findOrder` on a graph $G = (V, E)$ runs in $O(|V| + |E|)$ time.*

Proof. The only problem is to find a vertex v in each iteration. For this purpose we use an array $A[0, \dots, |V|-1]$ of lists, where each list $A[i]$ ($0 \leq i \leq |V|-1$) contains all vertices that have exactly i neighbors in L . Moreover, we have an array $B[1 \dots |V|]$ of pointers that allows us to find each vertex in A in constant time. Last, we have an integer z which stores $\max\{i \mid A[i] \text{ is a non empty list}\}$.

Initially, all vertices are part of $A[0]$ and $z = 0$. Finding a vertex v with the desired properties can be done in constant time; take the first vertex from $A[z]$. After adding v into L , we have to iterate over all its neighbors and correct their storage in A and also their pointers in B , both can be done using B in $O(N(v))$ time. Possibly we also have to increase z by one, which can be checked in constant time, or we have to decrease z , which cost in total not more than

the total costs of all increments of z so that we can neglect this time under the asymptotically point of view.

Consequently, one iteration takes $O(N(v) + 1)$ time and the total running time is $O(|V| + |E|)$. \square

A clique tree (T, B) of a chordal graph $G = (V, E)$ can be obtained by first computing a perfect elimination order L of G . We consider the vertices of G in the reverse order of L . While constructing a clique tree we will stepwise define a function Ψ that maps each vertex v to a node whose bag contains v , all its successors, and possibly some more vertices.

Let v be the currently considered vertex. If v has no successor, create a new node w of T with a bag containing only v . Moreover, if T has another node w' —i.e., v is not the first vertex considered—then connect w to w' . If v has a successor, let $\ell = \min\{j \mid v_j \text{ successor of } v\}$. If $B(\Psi(v_\ell))$ contains exactly the successors of v , add v to that bag and define $\Psi(v) = \Psi(v_\ell)$. Otherwise create a new node w of T with a bag containing v and all its successors and define $\Psi(v) = w$. In the latter case, for guaranteeing property (TD2), observe that

- by induction property (TD2) holds before considering v and that
- the set S of successors of v induces a clique in G , i.e., v_ℓ is adjacent to each vertex in $S - \{v_\ell\}$ and $S \subseteq B(\Psi(v_\ell))$.

Consequently, we simply can connect w to $\Psi(v_\ell)$.

By our construction the vertices of each bag induce a maximal clique different to all maximal cliques induced by other bags. In reverse direction, for each maximal clique, there exists a bag containing exactly its vertices V_C since each maximal clique C contains a vertex v such that C is the subgraph of G induced by v and its successors, i.e., $V_C \subseteq B(\Psi(v))$, and since C being a maximal clique implies that no vertex $v' \in V \setminus V_C$ can be part of $B(\Psi(v))$ with $B(\Psi(v))$ still inducing a clique in G . Thus, property (TDC) holds.

Note that the clique tree has at most $|V|$ nodes and, for each node w , the size of its bag is bounded by the degree of a vertex v with $\Psi(v) = w$. Consequently, the construction time and the size of the clique tree is bounded by $O(|V| + |E|)$. Finally by Lemma 1.3.9 we can observe that the construction above finds a tree decomposition of minimal width.

Theorem 2.3.8. *Given a chordal graph $G = (V, E)$ with treewidth k , a clique tree (T, B) for G can be found in $O(|V| + |E|)$ time such that*

- (T, B) has treewidth k and size $O(|V| + |E|)$ and
- T has at most $|V|$ nodes.

As we have seen in the introduction, a graph G needs not to have small treewidth for obtaining efficient algorithms as long as a clique tree for G exists. But which graphs do have a clique tree?

Lemma 2.3.9. *Chordal graphs are exactly the graphs for which a clique tree exists.*

Proof. By the last lemma it remains to show that chordal graphs are the only graphs having a clique tree. Let G be a graph for which a clique tree

(T, B) exists. Choosing an arbitrary node r of T as the root, we can traverse T top-down. Add the vertices of $B(r)$ to a list L . When we visit $w \neq r$ during this traversal, add the vertices of $B(w)$ not in L at the beginning of L . After the traversal L is a perfect elimination order of G , i.e., G is chordal (Theorem 2.3.6). \square

As promised in Section 1.2 now a description is given how the tree T of a clique tree (T, B) can be made binary if we allow to replace property (TDC) by property (TDC'). Recall that, for a tree T' with a node w , T'_w is the complete subtree of T' with root w .

As long as T has a node w with $c \geq 2$ children, we modify T as follows: Let v_1, \dots, v_ℓ be the vertices occurring in the bag of w or in the bag of a child of w . First, create an r -rooted binary tree T' of depth ℓ with 2^ℓ leaves $w_0, \dots, w_{2^\ell-1}$ from left to right. Define $B(r) = B(w)$ and, for each leaf w_i , $B(w_i) = \{v_j \mid (i \& 2^{\ell-j}) = 2^{\ell-j}\}$ (& denotes here the bitwise AND). For the remaining nodes w' of T' , define $B(w')$ as small as possible such that $(TD2)_v$ holds for all $v \in B(w)$, i.e., add a vertex v_i into $B(w')$ if $v_i \in B(\hat{w})$ for some leaf \hat{w} being a descendant of w' . See Fig. 2.3.2 for a small example. Second, for $1 \leq j \leq c$, identify each child \hat{w} of w in T with the leaf w_i of T' such that $B(w_i) = B(\hat{w})$. Note that, because of (TDC) two different children of w are identified with different leaves in T' . Third, set $T'' = T'$, and as long as there is a leaf w' of T'' not identified with a child of w (in T) modify T'' by a *shrinking step*: Remove w' as well as its parent \bar{w} and connect the other child $w'' \neq w'$ of \bar{w} to the parent of \bar{w} . At the end, replace the edges between w and its children in T by T'' .

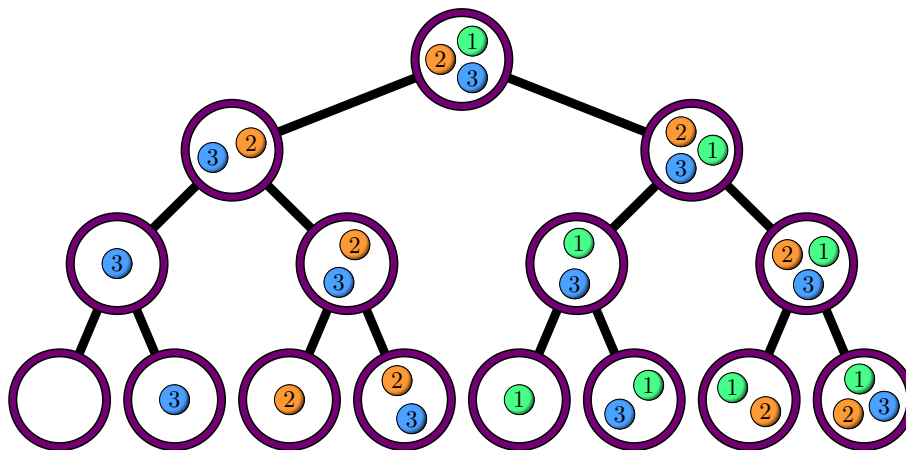


Figure 2.3.2: The example shows the tree T' and its bags described in the construction of a binary weak clique tree for the case $\ell = 3$.

Let us now analyze the size of (T'', B) compared to the total size over all bags of the leaves of T'' , i.e., the total size over all bags of children of w . For each vertex v occurring in a bag of (T'', B) , we have to count 1, in other words, we have cost 1 for v . Instead of counting these costs directly we distribute the costs of each vertex in the bag of an inner node of T'' to the vertices in the bags of the leaves of T'' and then count the assigned cost for each vertex in the bag

of each leaf of T'' . In detail, for each vertex v_j ($1 \leq j \leq \ell$) occurring in the bag of a non-leaf \bar{w} of T' and of T'' , let us add a cost of 2^{-i+1} to each v_j occurring in the bag of a leaf w' in $T'_{\bar{w}}$ and in $T''_{\bar{w}}$, respectively, where i denotes the distance between \bar{w} and w' . Observe that, if w'_1, w'_2, \dots, w'_q ($q \in \mathbb{N}$) are the nodes of T' in depth j , exactly half of the subtrees $T'_{w'_1}, T'_{w'_2}, \dots, T'_{w'_q}$ contain in all its bags v_j . Additionally, observe that each inner node of T' has exactly two children. The two observations above imply that, for $v_j \in B(\bar{w})$, we have assigned in T' total cost of at least 1. Moreover, the same is true in T'' since, for each shrinking step, the total cost do not change. Moreover, for each vertex u being part of the bag of a leaf w' , we have assigned in T'' cost of at most $2^{-1+1} + 2^{-2+1} + \dots \leq 2$ to u . Thus, the total size of the new nodes in (T'', B) and its bags is bounded by the size of the bags of the children of w .

An efficient implementation of the construction of T'' is described next. Recall that v_1, \dots, v_ℓ are the vertices occurring in a bag of w or of a child of w . Initially, create an array S of sets that maps each vertex $v_j \in B(w)$ to the set of all children of w whose bags contain v_j . Moreover, create an array P of pointers that we update dynamically and that maps each child of w to a node in a tree \tilde{T} that we construct in the following. Take a new node w^* as the root of \tilde{T} , define $\tilde{B}(w^*) = B(w)$, and initially set $P[\hat{w}] = w^*$ for each child \hat{w} of w . For each node w' of \tilde{T} , we also store in $Z[w']$ the number of pointers pointing to it. Since the values of Z can be updated easily when moving a pointer from one node to another. The updates of Z are not described explicitly.

The algorithm then works in rounds $j = 1, \dots, \ell$. The idea of the algorithm is that whenever there are two children \hat{w}_1 and \hat{w}_2 of w in T with $v_j \in \tilde{B}(\hat{w}_1) \setminus \tilde{B}(\hat{w}_2)$ such that both $P[\hat{w}_1]$ and $P[\hat{w}_2]$ point to the same node w' of \tilde{T} , then we want to add a left child w'_l and a right child w'_r to w' . In addition, for each child \hat{w} of w in T whose value of P points to w' , update $P[\hat{w}]$ such that $P[\hat{w}]$ now points to w'_l if $v_j \notin \tilde{B}(\hat{w})$ and such that $P[\hat{w}]$ now points to w'_r if $v_j \in \tilde{B}(\hat{w})$. The challenge is to do the updates of the pointers of all children of w for a round j in time $O(1 + |S[v_j]|)$; details how to obtain this running time are described now. In round j , iterate through the set of $S[v_j]$. For each \hat{w} in $S[v_j]$, check if $w' = P[\hat{w}]$ already has a right child w'' . If this is not the case, mark w' , create a right child w'' for w' , and define $\tilde{B}(w'') = \tilde{B}(w')$. In both cases, update $P[\hat{w}] = w''$. At the end of the j -th round, for each marked node w' , check if $Z[w'] \neq 0$. If so, replace w' in \tilde{T} by a new node w^+ — w^+ has the same parent, child, and bag as w' —and connect afterwards w' as new left child of w^+ and set $\tilde{B}(w') = \tilde{B}(w^+) \setminus \{v_j\}$. Note that by these w' - w^+ -exchanges over all marked vertices w' we implicitly update all pointers of nodes \hat{w} with $v_j \notin \tilde{B}(\hat{w})$. Otherwise, i.e., if $Z[w'] = 0$, we have no reason for introducing a left child of w' and therefore we do not want to have a right child. Thus, replace w' by its right child w'' by making the parent of w' to be the parent of w'' and then by deleting w' . At the end of round j , unmark all marked nodes.

After the iteration over all j , identify the root of \tilde{T} with w and each child \hat{w} of w in T with the leaf $P[\hat{w}]$ of \tilde{T} . It is not hard to see that before the first round (take $j=0$) and after the j -th round, the two invariants hold:

- for each node w' part of both \tilde{T} and T'' , we have $\tilde{B}(w') = B(w')$, and

- \tilde{T} can be obtained from T'' by deleting 'a lower part' of T'' , i.e., by deleting, for each $i = \ell, \ell - 1, \dots, j + 1$, all nodes \bar{w} and $\overline{\bar{w}}$ of T'' with a common parent and with $(\tilde{B}(\bar{w}) \setminus \tilde{B}(\overline{\bar{w}})) \cap \{v_i\} \neq \emptyset$.

Consequently, at the end of the algorithm (\tilde{T}, \tilde{B}) is the same as (T'', B) .

Concerning the running time we can observe that, apart from the time for constructing the bags, we have to spend only constant time for each vertex in a bag of a child of w . Since the construction of each bag can be done in linear time with respect to its size, the time of the whole algorithm above is linear in the size of the weak clique tree obtained, i.e., the running time is $O(|V| + |E|)$.

If we apply the modifications to all nodes with more than two children, we obtain a binary weak clique tree (T^*, B) from a clique tree (T, B) . In addition, from our cost analysis we can conclude that the increase of the size from (T, B) to (T^*, B) is bounded by the size of (T, B) , i.e., bounded by $O(|V| + |E|)$. Thus, (T^*, B) is also of size $O(|V| + |E|)$. Note that T^* has $O(|V|)$ leaves and each non-leaf with only one child is one original node of T , i.e., there are only $O(|V|)$ such nodes. Thus, T^* has $O(|V|)$ nodes.

Corollary 2.3.10. *Given a chordal graph $G = (V, E)$ with treewidth k , a weak clique tree (T, B) for G can be found in $O(|V| + |E|)$ time such that*

- (T, B) has treewidth k and size $O(|V| + |E|)$ and
- T is a binary tree with $O(|V|)$ nodes.

Chapter 3

Tree Decompositions on Planar Graphs

3.1 Introduction

The usual approach to solve an NP-hard problem on a graph G with n vertices and 'small' treewidth k consists of two steps: First, compute a tree decomposition for G of width ℓ , and second, solve the problem using this tree decomposition. Unfortunately, there is a trade-off between the running times of the first and second step depending on our choice of ℓ . Choosing ℓ closer to k increases the running time of the first step, but decreases that of the second step. It was shown that the second step on n -vertex graphs with a given tree decomposition of constant width can be solved in $O(n)$ time for all graph problems that are expressible in the so-called monadic second-order logic (MSOL) [26]. For an exact definition and some further information on MSOL see Section 5.1. The so-called NP-hard graph problems 3-coloring, triangle matching, Hamiltonian circuit, dominating set, vertex cover as well as independent set are examples for problems expressible in MSOL.

It appears that it is mostly the first step that is the bottleneck for the construction of efficient algorithms solving NP-hard problems on graphs of small treewidth. For n -vertex graphs G of treewidth k , the algorithm achieving the so far smallest width among the algorithms being polynomial in n and k is the algorithm of Feige, Hajiaghayi, and Lee [30]. It constructs a tree decomposition of width $O(k\sqrt{\log k})$. In particular, on general graphs no algorithms with constant approximation ratio are known that are polynomial in the number of vertices and in the treewidth. Better algorithms are known for the special case of planar graphs. On these graphs, all NP-hard problems given as example above remain NP-hard. Seymour and Thomas [83] showed that the so-called branch width $bw(G)$ of G and a so-called branch decomposition of width $bw(G)$ for a planar n -vertex graph G can be computed in $O(n^2)$ and $O(n^4)$ time, respectively, independent of the exact size of $bw(G)$. A minimum branch decomposition of a graph G can be used directly—like a tree decomposition—to develop efficient algorithms. Additionally, for each graph G , its branch width $bw(G)$ is closely related to its treewidth $tw(G)$, in detail, $bw(G) \leq tw(G) + 1 \leq \max(3/2 bw(G), 2)$ [78]. Thus, one can compute a 3/2-approximation of the treewidth on a pla-

nar graph in $O(n^2)$ time. Moreover, given a branch decomposition of width $bw(G) \geq 2$, we can find a tree decomposition of width $3/2 bw(G)$ in linear time, which is a $3/2$ -approximation of the treewidth of G . Gu and Tamaki [44] improved the running time to $O(n^3)$ for constructing a branch decomposition and thus for finding a tree decomposition of width $O(tw(G))$. It is unknown whether the problem of computing the treewidth for planar graphs can be solved in polynomial time or if it is NP-hard.

For planar graphs with n vertices and treewidth $k \in \mathbb{N}$, we present the first $O(n \cdot \text{poly}(k))$ -time algorithm that computes a tree decomposition of width $O(k)$ if the given graph has treewidth at most k . More precisely, if $k \in \mathbb{N}$ is given, for any α with $0 < \alpha < 1$, we can compute a tree decomposition of width at most $9k + 3\lfloor \alpha k \rfloor + 9$ in $O(n \cdot k^3 \min(1/\alpha, k) \log k)$ time for planar graphs of treewidth k . If no k is given, we can combine the new algorithm with a binary search and obtain so an $O(n \cdot k^3 \log^2 k)$ -time algorithm for constructing a tree decomposition of width $O(k)$. A simplified version of the algorithm described in the rest of this chapter with a weaker approximation ratio is published in [56].

3.2 Peelings, Mountains, and Connectivity

This and the following chapter focus on planar graphs. To describe algorithms on planar graphs, some more terminology and properties of planar graphs are necessary. The *directed version* of an undirected graph $G = (V, E)$ is the graph obtained from G by replacing each edge $\{u, v\}$ by the directed edges (u, v) and (v, u) . The two directed edges are then called *directed version* of $\{u, v\}$.

Definition 3.2.1 (combinatorial embedding). A (*combinatorial*) *embedding* φ of graph G and the directed version (V, D) of G is a permutation of D such that there exists a planar embedding ψ of G and such that for each edge $(v_1, v_2) \in D$, a vertex $v_3 \in V$ exists with

- $\varphi((v_1, v_2)) = (v_2, v_3)$ and
- edge $\psi(\{v_2, v_3\})$ follows directly after edge $\psi(\{v_1, v_2\})$ while clockwise turning around $\psi(v_2)$.

Then, φ is also called a combinatorial embedding of (G, ψ) .

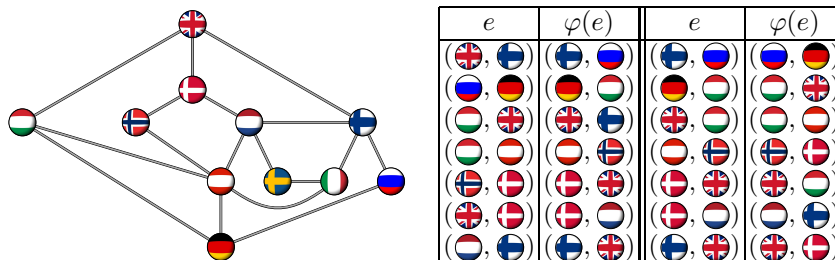


Figure 3.2.1: A planar embedding and some values of the corresponding combinatorial embedding φ .

See Fig. 3.2.1 for an example of the last definition. Let us call a list (x_0, \dots, x_{i-1}) ($i \in \mathbb{N}$) a *cycle* of a function f if all elements of the list are pairwise disjoint and $f(x_j) = x_{j+1 \bmod i}$ holds for all $0 \leq j < i$.

For simplicity, in this thesis we consider a combinatorial embedding only for connected graphs. The reason for this restriction is that we then can easily define a *face* of (G, φ) or of φ as a cycle of φ . A *rooted* embedding φ is a combinatorial embedding, where one face is specified as *outer face* of φ and where all other faces are called *inner faces* of φ . We define an *embedded graph* as a tuple (G, φ) , where G is a connected planar graph and φ is the rooted combinatorial embedding. An edge e is *incident* to a face F and vice versa if the cycle describing F contains a directed version of e . Moreover, e is *on the boundary* of F if this is the case for exactly one directed version of e . A vertex is *incident* to a face or is *on the boundary* of a face if it is the endpoint of an edge incident to the face and on the boundary of the face, respectively.

Note that, if ψ is a planar embedding of a non-empty graph G and if φ is the combinatorial embedding of (G, ψ) , there is a bijective mapping m between the faces of ψ and φ such that the vertices and edges being incident to a face F of ψ are incident to the face $m(F)$ of φ and such that the same holds for the vertices and edges on the boundary. Even if we do not mention the planar embedding of an embedded graph explicitly, for an easier intuition it is mostly a good idea to assume that an embedded graph (G, φ) has some fixed planar embedding ψ such that φ is the combinatorial embedding of (G, ψ) .

If $\{v, u_0\}, \dots, \{v, u_{i-1}\}$ ($i \in \mathbb{N}$) are the incident edges of a vertex v and if $\varphi((u_j, v)) = (v, u_{j+1 \bmod i})$ holds for all $0 \leq j < i$, the *clockwise order around* v is the cyclic order that assigns the vertex $u_{j-1 \bmod i}$ as predecessor and the vertex $u_{j+1 \bmod i}$ as successor to each vertex u_j ($0 \leq j < i$). We also say that the series $u_j, \dots, u_{i-1}, u_0, u_1, \dots, u_{j-1}$ ($0 \leq j < i$) *appears clockwise around* v (with respect to φ). We say the same if some vertices of the series are missing. Let $\{u_0, u_1\}, \dots, \{u_{i-2}, u_{i-1}\}$ and $\{u_{i-1}, u_0\}$ ($i \in \mathbb{N}$) be the edges incident to a face F such that $\varphi((u_{j-1 \bmod i}, u_j)) = (u_j, u_{j+1 \bmod i})$ holds for all $0 \leq j < i$. If F is the outer face, we say that the series $u_j, \dots, u_{i-1}, u_0, u_1, \dots, u_{j-1}$ ($0 \leq j < i$) *appears clockwise around* F (with respect to φ), and if F is an inner face, we say this to the reverse of this series. Once again, we say the same if some vertices of the series are missing. The *counter-clockwise order* is the reverse of the clockwise order. A series *appears counter-clockwise* if the reverse of the series appears clockwise. Moreover, a series of vertices *appears around* a vertex v if it appears clockwise or counter-clockwise around v .

An embedded graph is called *almost triangulated* if each inner face is incident to exactly three vertices and three edges. If additionally the outer face is incident to exactly three vertices and three edges, the graph is called *triangulated*. A *multigraph* is a tuple (V, E) , where V is a set of vertices and E is a multiset of edges. The *dual graph* H of an embedded graph (G, φ) is the multigraph that consists of a vertex v_F for each face of (G, φ) and of an edge $e' = \{v_{F'}, v_{F''}\}$ for each edge e of G , where F' and F'' are the two faces being incident to e . Intuitively, we can place each vertex v_F of H into the face F of (G, φ) . For that reason, we also say that an edge e in G and an edge e' in H cross if the existence of e is the reason for having e' in H .

Some less standard or less natural definitions are also needed in the following. A *path on the boundary* of a face F means a path in the graph defined by the boundary of F . The *fence* of a face F in an embedded graph (G, φ) is the

embedded subgraph obtained from the boundary of F by deleting all edges being incident to the outer face and all vertices that are not an endpoint of one of the remaining edges.

An *area* is a set of faces. The *boundary* of an area A is the embedded subgraph of G that consists of all vertices and edges that are on the boundary of exactly one face contained in A . The *fence* of an area A is obtained from its boundary by deleting all edges incident to the outer face and all vertices that are not an endpoint of one of the remaining edges. The edges of a boundary and of a fence (with respect to a face or an area) are called *boundary edges* and *fence edges*, respectively. *Boundary vertices* and *fence vertices* are defined analogously. The vertices and edges *incident* to an area A are the vertices and edges that are incident to a face contained in A . Let (G, φ) be an embedded graph and H be a subgraph of G with a combinatorial embedding ζ . Let us say that a face or an area F_1 of (H, ζ) *covers* another face or area F_2 of (G, φ) if there is a set of faces of (G, φ) including F_2 such that the area consisting of the union of these faces has the same boundary as F_1 .

For an area A in an embedded graph (G, φ) , we call a subset A' of A a *subarea* of A . In addition, for a set \mathcal{A}^* of subareas of A , we define $A - \mathcal{A}^*$ as $A \setminus \bigcup_{A' \in \mathcal{A}^*} A'$ for a simpler notation and denote by $G[A - \mathcal{A}^*]$ the graph that consist of all vertices and edges that are incident to a face in $A - \mathcal{A}^*$ or that are part of the fence of an area $A' \in \mathcal{A}^*$.¹ An area A in an embedded graph (G, φ) is *connected*, if, for all faces F and F' contained in A , there is a list of faces $F_1 = F, F_2, \dots, F_t = F'$ all contained in A such that each pair of consecutive faces F_i and F_{i+1} in this list has at least one common fence edge. Let us say

- for a path P containing three consecutive vertices v_1, v_2 , and v_3 that a neighbor $u' \notin \{v_1, v_3\}$ of v_2 *leaves on the left (right) side* of P (with respect to an embedding φ) if v_1, u', v_3 appear clockwise (counter-clockwise) around v_2 .
- that two neighbors u' and u'' of v_2 are *on the same side* and *on different sides* of P (with respect to an embedding φ) if u' and u'' leave P on the same side and on different sides, respectively.

For an efficient computation, we represent each faces of an embedded graph (G, φ) with $G = (V, E)$ by a single number, and we additionally have an array of pointers assigning to each face an incident edge. If we want to know the remaining edges incident to F , we can simply follow the edges appearing in clockwise order around F . The next two lemmata describe well-known properties on planar graphs.

Lemma 3.2.2 (Euler's formula). *If G is a planar graph with n vertices and m edges, each combinatorial embedding φ of G has $f = m - n + c + 1$ faces, where c denotes the number of connected components of G .*

Proof. We prove the lemma by an induction over m . If $m = 0$, we have n connected components and one face, i.e., the lemma holds. Otherwise, let us remove from G an arbitrary edge e . Let us denote by m' , f' , and c' the number of edges, of faces, and of connected components of the graph obtained, respectively. Clearly, $m' = m - 1$. Either e is incident to one face or to two faces

¹An edge may be a fence edge of two ares in \mathcal{A}^* and, thus, not incident to a face in $A - \mathcal{A}^*$.

with respect to φ . In the first case, the connected component of G to which e belongs is divided into two connected components, i.e., $c' = c + 1$, and the number of faces remains unchanged, i.e., $f' = f$. In the latter case, the two faces incident to e are 'merged' by the removal to one face F , i.e., $f' = f - 1$, and since the endpoints of e are still connected by the edges incident to F , the number of connected components does not change, i.e., $c' = c$. The lemma follows in both cases since $f' = m' - n + c' + 1$ holds by induction hypothesis. \square

Lemma 3.2.3. *Every combinatorial embedding of a graph with n vertices has at most $\max\{0, n - 1, 3n - 6\}$ edges and at most $2n$ faces.*

Proof. Let φ be a combinatorial embedding of an n -vertex graph G having m edges and c connected components. Let f be the number of faces of φ . W.l.o.g. G is connected. Otherwise, as long as there is a connected component containing a vertex u and another connected component containing a vertex v , add an edge $\{u, v\}$ into G . Note that this operation does not change the number of faces and only increases the number of edges.

Since a graph with $q \in \{1, 2\}$ edges has at least $1 + q$ vertices and one face, it remains to prove the lemma in the case $m \geq 3$. Then, each face is incident to at least 3 edges, and the fact that an edge can be incident to at most two faces implies that $3f \leq 2m$. Subtracting from this inequality two times the equality of the last lemma, we can conclude that $f \leq 2n - 2c - 2$. Thus, $m \leq 3n - 3c - 3 \leq 3n - 6$. \square

It is a well-known fact that a graph G is planar if and only if each of the two graphs shown on Fig. 3.2.2 is not a minor of G .

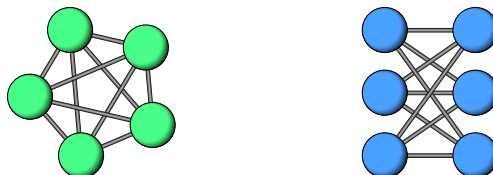


Figure 3.2.2: The K_5 and the $K_{3,3}$ is shown on the left and right side, respectively.

Definition 3.2.4 (inherited embedding). Given a connected embedded graph (G, φ) and a subgraph H of G , the *inherited embedding* $\varphi|_H$ from φ is a rooted embedding of H for which the edges of H incident to each vertex v of H appear around v in the same order in φ and in $\varphi|_H$. Moreover, the outer face of $\varphi|_H$ covers the outer face of φ .

Next, we define a generalization of a path called alley and some further terminology for alleys. Many problems search for some kind of path in a given graph—e.g., see Section 2.2—but it turns out that we need alleys instead of paths for the following approach to compute a tree decomposition.

Adding an edge e into a face F of an embedded graph (G, φ) with $G = (V, E)$ means replacing (G, φ) by a graph (G', φ') such that $G' = (V, E \cup \{e\})$, $\varphi'|_G = \varphi$, and F is the only face of φ not being a face of φ' . If F_1 and F_2 are the faces

of (G', φ') covered by F , we also say that F_1 and F_2 are *obtained from F by adding e* . If (H, ψ) and (G, φ) are embedded graphs such that $H = (V', E')$ is a subgraph of $G = (V, E)$ and such that $\psi = \varphi|_H$, we say that a face F of (H, ψ) is *incident* to a vertex of $V \setminus V'$ and an edge of $E \setminus E'$ if the area of (G, φ) with the same boundary than F is incident to the vertex and the edge, respectively. Let φ' be an embedding of a graph G' obtained from adding a new vertex (new edge) into G . We then say that φ' *embeds* this vertex (this edge) into a face F' of (G, φ) if F' is incident to this vertex (this edge).

For an embedded graph (G, φ) with $G = (V, E)$, we call two elements $s_1, s_2 \in V \cup E$ to be *face-adjacent* (with respect to φ) if they are incident to a common face F of (G, φ) . The face F in this case is called a *connection face* of s_1 and s_2 . A face without this property is called a *non-connection face* of s_1 and s_2 . In an embedded graph (G, φ) , an *A-alley* for an area A from a vertex $s \in V$ to a vertex $t \in V$ is a list $P = (s, \dots, t)$ of vertices being pairwise disjoint, except for possibly $s = t$, such that

- for each pair of consecutive vertices u and v , there is a connection face contained in A and
- one can add edges between all pairs of consecutive non-adjacent vertices into one of their connection faces contained in A without destroying planarity.

If $s = t$, the A -alley is called a *cyclic A-alley*. For an A -alley $P = (v_1, \dots, v_k)$ and a pair $i, j \in \mathbb{N}$ with $1 \leq i \leq j \leq k$, the list $(v_i, v_{i+1}, \dots, v_j)$ is called an *A-suballey* of P . If P is a cyclic alley, i.e., $v_k = v_1$, we also call the list $(v_j, \dots, v_{q-1}, v_1, \dots, v_i)$ to be a suballey of P . If A is the area obtained from the union of all inner faces in (G, φ) , we use the abbreviations *alley* and *suballey* for A -alleys or A -suballeys of (G, φ) . Note that we never use the outer face as a connecting face for two consecutive vertices of an alley. The *inner vertices* of an A -alley from s to t are the vertices of the alley different from its *endpoints* s and t . An alley with a first vertex s and a last vertex t is called an *s-t-connecting* alley. The *length* $|P|$ of an alley $P = (u_1, \dots, u_q)$ is $q - 1$. Two alleys $P_1 = (u_1, \dots, u_p)$ and $P_2 = (v_1, \dots, v_q)$ in an embedded graph (G, φ) *cross* if, for the multigraph G' obtained from G by adding two sets $E_1 = \{(u_1, u_2), \dots, (u_{p-1}, u_p)\}$ and $E_2 = \{(v_1, v_2), \dots, (v_{q-1}, v_q)\}$ of new edges, there is either

- (a) no combinatorial embedding φ' for the new edges with $\varphi'|_G = \varphi$ or
- (b) for all such combinatorial embeddings φ' of G' with $\varphi'|_G = \varphi$, there exists a vertex with four incident edges in $E_1 \cup E_2$ such that the edges of E_1 and E_2 alternate clockwise around v .

In case (a), we say that P_1 and P_2 have a *non-planar crossing* and, in case (b), they have a *planar crossing*. Note that in the case of a non-planar crossing, there are tuples (u_i, u_{i+1}) and (v_i, v_{i+1}) of consecutive vertices of P_1 and P_2 , respectively, such that u_i, v_i, u_{i+1} , and v_{i+1} appear around a common connection face F in (G, φ) .

We call a set $S \subseteq V$ of vertices in an embedded graph (G, φ) to be *face-connected* if there is an alley in (G, φ) from each vertex of S to each other vertex of S having as inner vertices only vertices of S . The *depth* of an alley P or a path P in an embedded graph (G, φ) —denoted by $\text{depth}(P)$ —is the minimum peeling number of its vertices, where the peeling numbers are defined as follows.

Definition 3.2.5 (peeling, ℓ -outerplanar, peeling index, peeling number). The *peeling* \mathcal{P} of an embedded graph (G, φ) is the (unique) list of vertex sets (V_1, \dots, V_ℓ) with $\bigcup_{i=1}^\ell V_i = V_G$ and $V_\ell \neq \emptyset$ such that each set V_i ($1 \leq i \leq \ell$) contains all vertices that are incident to the outer face of the embedding $\varphi|_{G[V']}$ with $V' = V \setminus (V_1 \cup \dots \cup V_{i-1})$. We then say that (G, φ) and φ are ℓ -outerplanar and that both have *peeling index* ℓ . For a vertex $v \in V_i$, we call i the *peeling number* of v (with respect to φ)—also denoted by $\mathcal{N}_\varphi(v)$. The peeling number of an edge of G is the average peeling number of its endpoints and denoted by $\mathcal{N}_\varphi(e)$, i.e., $\mathcal{N}_\varphi(\{u, v\}) = (\mathcal{N}_\varphi(u) + \mathcal{N}_\varphi(v))/2$.

An example of an embedded graph with its peeling numbers can be found in Fig. 3.3.1. We call an edge of an embedded graph (G, φ) *horizontal* (with respect to φ) if it connects two vertices of the same peeling number with respect to φ . Otherwise, we call it *vertical* (with respect to φ).

Definition 3.2.6 (crest, mountain). Let (G, φ) be an embedded graph. A maximal connected set U of vertices having all the same peeling number $c \in \mathbb{N}$ with respect to φ is called a *crest* in (G, φ) (of *height* c) if no vertex in U is face-adjacent to a vertex with a peeling number larger than c . An embedded graph is called a *mountain* if it has exactly one crest.

We see in Section 3.5 that it is easy to construct a tree decomposition for a mountain. Therefore, the idea of our main algorithm is to decompose our graph into mountains and to combine tree decompositions for the mountains to a tree decomposition for the whole graph. The next section shows how a graph can be decomposed into mountains. Unfortunately, this algorithm only works on graphs with a certain so-called connectivity. The idea is now to decompose a given graph such that each part of the decomposition has the desired connectivity.

Let us define a graph G to be *biconnected* (*triconnected*) if, between each pair of non-adjacent vertices, there are two (three) internally vertex-disjoint paths ending in these two vertices. From Lemma 2.2.4 we can conclude that the following definition is equivalent: a graph G is biconnected (triconnected) if there does not exist ≤ 1 vertex (≤ 2 vertices) of G whose removal from G disconnects two non-adjacent vertices of G . If one vertex disconnects G , it is called *cutpoint*, and a pair of vertices disconnecting G is called *separation pair*. Say (G, φ) is biconnected and triconnected if this is true for G . A *biconnected component* or a *block* of G is a maximal biconnected subgraph of G . Next, we describe a decomposition of a graph into biconnected components. For this purpose, let us define a *block-cutpoint tree* [35, 47] of a connected graph G as a graph T such that

- the nodes of T are the cutpoints and the biconnected components of G and
- each biconnected component B is incident in T to all cutpoints of G that are part of B .

Note that each edge of a block-cutpoint tree connects one cutpoint and one biconnected component. We next show that T is a tree. Assume for a contradiction that T contains a cycle. Such a cycle has as vertices two biconnected components. These components cannot be disconnected in G by the removal of one cutpoint of G . Contradiction. As a consequence, T is a tree.

Lemma 3.2.7. *A block-cutpoint tree can be found in linear time [87].*

Proof. For finding the cutpoints of a given connected graph $G = (V, E)$, let us consider for a fixed DFS on G its DFS-tree T . It is not hard to see that the following three conditions characterize the vertices being no cutpoints.

- The root r of T is no cutpoint if and only if it has exactly one child.
- A leaf of T is no cutpoint.
- An inner vertex $v \neq r$ of T is no cutpoint of G if and only if, for each child u of v in T , there is an edge from a proper descendant u' of u to a proper ancestor v' of v .

During the DFS, where we already have computed discovery and finishing times for the vertices, we additionally determine, for each vertex v , a value $a[v]$ as the minimum discovery time over all discovery times of vertices that are connected by a non-tree edge to a descendant of v . Just before we set the finishing time of v , we can compute $a[v]$ as the minimum over the smallest $a[u]$ with u being a child of v and over the smallest discovery time of a vertex $u \in V$ adjacent to v not being the parent of v .

For finding the cutpoints of G , it only remains to show that we can test the last condition above, i.e., the condition on an inner vertex $v \neq r$ of T . Let $S[v]$ be the set of children of v . Again, just before setting the finishing time of v , determine the set $S^*[v]$ of vertices $u \in S[v]$ for which $a[u]$ is not smaller than the discovery time $d[v]$ of v . Note that, for each $u \in S^*[v]$, no descendant of u is connected to a proper ancestor of u . Moreover, if v is a cutpoint for two vertices u_1 and u_2 , one of the two vertices is a descendant of v and contained in $S^*[v]$. Thus, v is a cutpoint if and only if $S^*[v] \neq \emptyset$.

Assume that the cutpoints of G are the colored red whereas all other vertices are uncolored. Let a *white component* be a connected component in the graph obtained from G by removing the red vertices. Then, a block-cutpoint tree can be obtained by

1. merging each white component to one so-called *component node*,
2. merging, for each cutpoint v , all component nodes containing a vertex $u \in S[v] \setminus S^*[v]$ with the component node containing the parent of v in T , and
3. adding the red vertices to each adjacent component node.

Note that the modified DFS has the same asymptotic running time as a usual DFS; thus, a block-cutpoint tree can be found in $O(|V| + |E|)$ time. \square

To define so-called triconnected components, further definitions are necessary. Let us say for a subgraph H of a graph $G = (V, E)$ that two vertices u and v are connected in G by an *internally H -avoiding path* if there is a u - v -path in G whose inner vertices do not belong to H . Moreover, an *H -attached component* in G is a subgraph $G' = G[V']$ of G for a maximal set of vertices $V' \subseteq V$ such that each pair of vertices in V' is connected in G by an internally H -avoiding path. A subgraph H of G is a *peninsula* of G if $H = G$ or each H -attached component of G has at most two vertices in common with H . The

reader may take a look at Fig. 4.4.1 for an example of a peninsula. In addition, for a peninsula H , define \mathcal{R}_H as the function that maps each subgraph C of G to the set of vertices common to C and H . Let C be an H -attached component. Note that, by the definition of a peninsula, $|\mathcal{R}_H(C)| \leq 2$. If $|\mathcal{R}_H(C)| = 2$, we call $\mathcal{R}_H(C)$ also a *virtual edge*.

Before we can define a triconnected component, we finally need the definition of the induced graph $H(G)$ of a peninsula H of G .

Definition 3.2.8 (induced graph, virtual edge). Given a connected planar graph G and a peninsula $H = (V', E')$ of G , the *induced graph* $H(G)$ has the vertex set V' and the edge set $\{\{u, v\} \mid (\{u, v\} \in E') \vee (u, v \in V' \text{ and } u \text{ and } v \text{ are connected in } G \text{ by an internally } H\text{-avoiding path})\}$. An edge $\{u, v\}$ in $H(G)$ is called a *virtual edge* if there is an internally H -avoiding path between u and v in G (even if $\{u, v\} \in E'$).

A *triconnected component* of G is a triconnected induced graph $H(G)$ of a peninsula H of G such that H is not a subgraph of another peninsula H' with $H'(G)$ being triconnected. In addition, a *cycle component* in G is an induced graph $H(G)$ of a peninsula H such that $H(G)$ is a cycle. A *multiple-edge component* is a multigraph consisting only of 2 vertices (a separation pair of G) connected by at most one edge of G and several virtual edges.

For obtaining a decomposition of G into triconnected components, we make use of the so-called SPQR trees. An *SPQR tree* [11] of a biconnected graph G is a tuple (T, M) , where $T = (W, F)$ is a tree and M is a mapping from W to a set of so-called split-components of G . A *split-component* is either a triconnected component, a cycle component, or a multiple-edge component. It is not hard to see that a split-component is the induced graph $H(G)$ of a peninsula H of G , which is extended by some edges in the case of a multiple-edge component. Additionally, for an SPQR tree the following three properties hold:

- Each edge $\{u, v\}$ of G is part of exactly one split-component $H \in M(W)$ —possibly, further virtual edges $\{u, v\}$ are part of other split-components.
- For each pair of vertices u and v of G , there is at most one multiple-edge component having u and v as vertices.
- Two nodes $w_1, w_2 \in W$ with $H_i = M(w_i)$ ($i = 1, 2$) are connected in T by an edge in F if and only if $|\mathcal{R}_{H_1}(H_2)| = 2$ and either one of $\{H_1, H_2\}$ is a multiple-edge component or there is no multiple-edge component with vertex set $\mathcal{R}_{H_1}(H_2)$.

By the definition above, two multiple-edge components of an SPQR tree (T, M) are not adjacent in T . One can show that an SPQR tree of a biconnected planar graph can be found in linear time [11, 45]. An example of an SPQR tree is shown in Fig. 3.2.3. Note that, if (T, M) is an SPQR tree of a planar graph, each split component of (T, M) is also planar by Obs.1.4.9. Finally, for an easier notation, let us extend the mapping M from the nodes of T to each subtree T' of T in the canonical way: If w_1 is the root of T' , if $\{w_2, \dots, w_\ell\}$ are the remaining nodes of T' , if $(W'_i, F'_i) = M(w_i)$ ($1 \leq i \leq \ell$), and if S is the set of all virtual edges in F'_1, \dots, F'_ℓ , we define $M(T') = (\bigcup_{i=1}^{\ell} W'_i, \bigcup_{i=1}^{\ell} F'_i \setminus S \cup S')$, where $S' = \emptyset$ if w_1 has no parent in T , and otherwise, S' consists exactly of the common virtual edge of w_1 and its parent in T .

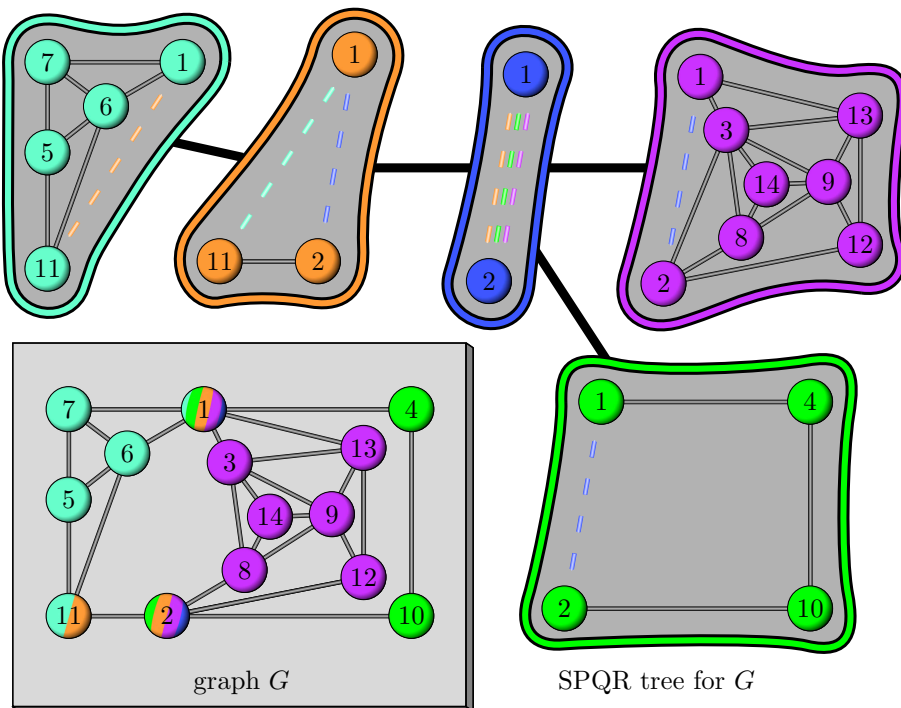


Figure 3.2.3: A biconnected graph G and an SPQR tree of it.

3.3 Decomposition into mountains

In this section, we want to decompose a graph into mountains. However, it appears that it is much easier to find such a decomposition for a certain embedded supergraph of (G, φ) . Hence the idea is to construct such a decomposition for the supergraph and to use it also for (G, φ) . For this approach to work, the supergraph of (G, φ) should have the same crests as the original graph. Since the definition of a crest depends on the peeling numbers, it is therefore useful to choose a supergraph with the same peeling numbers. The following lemma shows that such supergraphs exist. In this lemma, we call an embedded graph (G, φ) *down-connected* if each vertex of G is either incident to the outer face or adjacent to a vertex with a smaller peeling number.

Lemma 3.3.1. *Let (G, φ) and (G', φ') be embedded graphs with $\varphi'|_G = \varphi$ such that*

- (a) G' is obtained from G by adding a set of edges, where each edge connects vertices of different peeling numbers with respect to φ or
- (b) G is down-connected and G' is obtained from G by adding an arbitrary set of edges into inner faces of G .

Then, in both cases the peeling numbers with respect to φ and to φ' are the same.

Proof. Adding edges into (G, φ) can only increase the peeling numbers. However, each vertex u with $N_\varphi(u) \geq 2$ is in (G, φ) face-adjacent (case (a)) or

adjacent (case (b)) to a vertex v with $\mathcal{N}_\varphi(v) < \mathcal{N}_\varphi(u)$. Since this property is maintained after adding edges into inner faces connecting vertices of the boundary of the inner face with different peeling numbers or in case (b) with arbitrary peeling numbers, the peeling numbers cannot increase. \square

The following definition describes the supergraphs that we want to use later:

Definition 3.3.2 (down closure, down edge, down vertex). A *down closure* of an embedded graph (G, φ) is a triple $(G', \varphi', \downarrow)$ such that

- (G', φ') is obtained from (G, φ) by adding, as long as possible, not already existing edges with endpoints of different peeling numbers into inner faces of the current graph, and adding afterwards as long as possible, not already existing edges with endpoints of the same peeling number i into inner faces whose boundary contains vertices of peeling number i and possibly some further vertices of peeling number $i + 1$.
- \downarrow is a function that assigns to each vertex $v \in V$ of peeling number $\ell \geq 2$ one of its neighbors in (G', φ') with smaller peeling number.

$\downarrow(v)$ is called the *down vertex* of v and is denoted by $v\downarrow$. The *down edge* of v is the edge whose endpoints consist of v and its down vertex.

Some of the nice properties of a down closure are described in the next four lemmata. For an example of a down closure, see Fig. 3.3.1, but for the moment, ignore the difference between dotted, dashed, and usual edges.

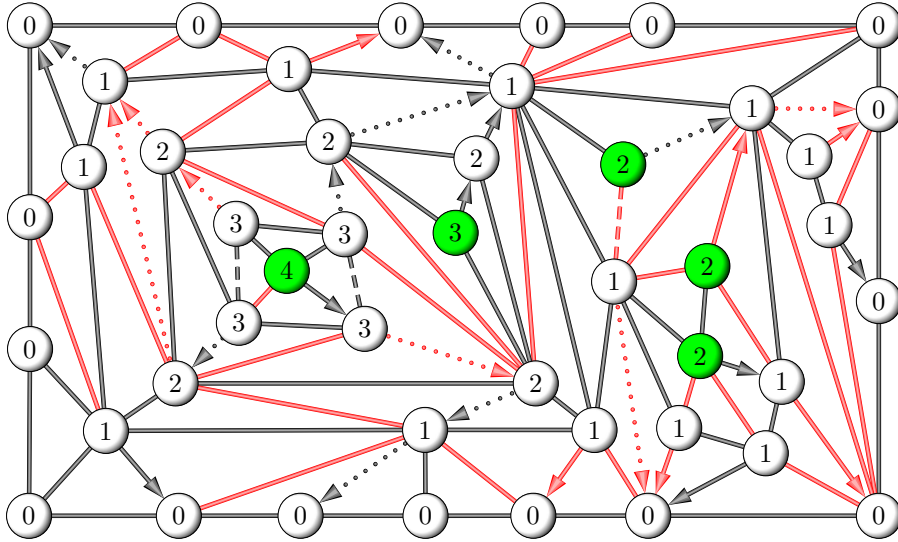


Figure 3.3.1: The figure shows an undirected embedded graph (G, φ) , whose edges are black, and the down closure $(G', \varphi', \downarrow)$ of (G, φ) , whose edges are black and red. Moreover, the arrows depict the function \downarrow . The numbers inside the vertices are the peeling numbers of (G, φ) as well as of (G', φ') , and the green vertices are part of a crest in (G, φ) . Dashed and dotted edges are edges of a crest separator.

Lemma 3.3.3. *Let $(G', \varphi', \downarrow)$ be a down closure of an embedded connected graph (G, φ) . Then $\mathcal{N}_\varphi(v) = \mathcal{N}_{\varphi'}(v)$ for all vertices v of G . Moreover, each crest of (G, φ) is a crest of (G', φ') and vice versa.*

Proof. That the peeling number of the vertices in (G, φ) and (G', φ') are the same follows directly from Lemma 3.3.1.

For showing the second part of the lemma, let us first show some auxiliary observation. For each vertex u of G belonging to a crest of (G, φ) , the following sets U and U' are equal: U and U' are the maximal connected sets of vertices in (G, φ) and in (G', φ') , respectively, such that both sets contain u and consist only of vertices with the same peeling number $i \in \mathbb{N}$. Since two vertices adjacent in (G, φ) are also adjacent in (G', φ') $U \subseteq U'$. If $U' \subseteq U$ is not true, there is an edge $\{u_1, u_2\}$ in G' , but not in G , with $u_1 \in U$ and $u_2 \in U' \setminus U$. Possibly, in the construction process of the down closure, the edge $\{u_1, u_2\}$ was added into a face whose boundary had exactly four vertices with the vertices different from u_1 and u_2 all having peeling number $i + 1$. In this case, u_1 is face-adjacent to a vertex of peeling number $i + 1$ in (G, φ) and cannot be part of a crest in (G, φ) . In all other cases, the construction rules of a down closure guarantee that there is a path P on the boundary of the face, in which $\{u_1, u_2\}$ was added, consisting exclusively of vertices of peeling number i . If possibly some of the edges of P were added before $\{u_1, u_2\}$ and are not contained in G , they can be replaced in the same way as $\{u_1, u_2\}$. Consequently, we have $U = U'$.

Let U be a crest of (G, φ) . We next show that all vertices of U belong to the same crest in (G', φ') . Let i be the peeling number of the vertices in U . If U is no crest in (G', φ') , either

- a vertex of u is face-adjacent to a vertex of peeling number $i + 1$ or
- there is a path from a vertex of U to a vertex $v \notin U$ consisting completely of vertices with peeling number i and v is face-adjacent to a vertex of peeling number $i + 1$.

By the observation above, the latter case cannot occur. Moreover, since two vertices are face-adjacent in (G', φ') only if this is true in (G, φ) and since no vertex of u is face-adjacent to a vertex with peeling number $i + 1$ in (G, φ) , also the first case cannot occur.

We finally show that, for each crest U' of (G', φ') , all vertices of U' belong to one crest in (G, φ) . Assume that, in (G, φ) , a vertex $u' \in U'$ is face-adjacent to a vertex v of peeling number $i + 1$. Thus, there is a u' - v -path P on the boundary of a connection face that connects u' and v . Since the vertices of P all have either i or $i + 1$ as peeling number, the first vertex \hat{v} on P with peeling number $i + 1$ is adjacent to a vertex \hat{u} of U' in (G, φ) . Since \hat{v} is also adjacent to \hat{u} in (G', φ') , we can conclude that no vertex in U' can belong to a crest in (G', φ') . It remains to show that all vertices of U' are part of the same crest in (G, φ) . This follows again from the observation above. \square

Because of the lemma above, when considering an embedded graph (G, φ) and a down closure $(G', \varphi', \downarrow)$ of (G, φ) , we often refer to peeling numbers or crests without mentioning the corresponding embedded graph.

For proving the next lemma, we need a more sophisticated definition how a set can disconnect two sets: S *weakly disconnects* two vertex sets U_1 and U_2 if

no connected component of $G[V \setminus S]$ contains vertices of both sets U_1 and U_2 . If additionally $S \cap (U_1 \cup U_2) = \emptyset$ holds, S *strongly disconnects* U_1 and U_2 .

Lemma 3.3.4. *Let $(G', \varphi', \downarrow)$ be a down closure of a triconnected embedded graph (G, φ) . Then (G', φ') is an almost triangulated graph.*

Proof. Assume that there is an inner face F of (G', φ') whose boundary consists of more than three vertices. Define i to be the minimum of $\mathcal{N}_{\varphi'}(v)$ over all boundary vertices v of F . Choose u, u_1, u_2 , and u' as four different boundary vertices of F such that $\mathcal{N}_{\varphi'}(u) = i$ and such that u is adjacent to u_1 and u_2 . Since no edge $\{u, u'\}$ was added into F , the definition of a down closure implies that there must be already an edge connecting u and u' . This means that $\{u, u'\}$ strongly disconnects $\{u_1\}$ and $\{u_2\}$. However, this is not possible since G is a triconnected graph. \square

Lemma 3.3.5. *Let $(G', \varphi', \downarrow)$ be a down closure of a triconnected embedded graph (G, φ) , and let F be an inner face of (G, φ) for which the set of its boundary vertices is divided into non-empty sets V_i and V_{i+1} ($i \in \mathbb{N}$) containing the vertices with peeling numbers i and $i+1$, respectively. For each vertex $u \in V_{i+1}$, there is then a vertex $v \in V_i$ with $\{u, v\}$ being an edge of G' .*

Proof. The property that a vertex $u \in V_{i+1}$ is face-adjacent to a vertex of V_i is maintained even after adding edges into F with at least one endpoint of peeling number i . Since (G', φ') is finally an almost triangulated graph, the face-adjacency turned into a usual adjacency. \square

Lemma 3.3.6. *Let $(G', \varphi', \downarrow)$ be a down closure of an embedded graph (G, φ) . If F' is a face of φ' covered by a face F of (G, φ) and if at least one boundary vertex of F has peeling number i , whereas all other boundary vertices of F have peeling number at least i , then at least one boundary vertex of F' has also peeling number i .*

Proof. If, during the construction of a down closure as described in Definition 3.3.2, we add an edge $\{u, v\}$ into a face F^* covered by F and if at least one boundary vertex of F^* has peeling number i , whereas all other vertices have a peeling number at least i , then at least one of the vertices u and v has also peeling number i . Hence the two faces F' and F'' obtained from F^* by adding the edge $\{u, v\}$ will both have a boundary vertex with peeling number i . Hence the lemma follows by a simple induction. \square

Note also that a down closure can be constructed efficiently.

Lemma 3.3.7. *A down closure for a triconnected embedded graph (G, φ) with $G = (V, E)$ can be constructed in $O(|V|)$ time.*

Proof. In a first step, for each face F of (G, φ) with boundary vertices of different peeling numbers i and $i+1$, choose a boundary vertex v of F with peeling number i , and add edges between v and each boundary vertex u of F with peeling number $i+1$ if this edge does not already exist. Note that an already existing edge $\{u, v\}$ must be a boundary edge since, otherwise, $\{u, v\}$ would be a separator of size 2. Afterwards the boundary of each inner face with vertices of different peeling numbers i and $i+1$ contains at most two vertices of peeling number $i+1$. Choose, for each such face F , a boundary vertex v of F

with peeling number $i + 1$, and add edges between v and each boundary vertex u of F with peeling number i if this edge is not already a boundary edge of F . After this second step, each boundary of a face F with at least four boundary vertices not all having the same peeling number consists of exactly four vertices. If v_1, v_2, v_3 , and v_4 is the series of vertices appearing around F , then v_1 and v_3 have the same peeling number different from the same peeling number of v_2 and v_4 . If the two vertices with the smaller peeling number are already connected, they would again define a separator of size 2. Therefore, we can connect them by an additional edge in a third step of our algorithm. After the third step, for each face F the boundary vertices of F consist either of exactly three vertices or of a set of vertices having all the same peeling number. Therefore, the remaining graph can be *almost triangulated*, if for each inner face F with more than 3 boundary vertices, we take one boundary vertex u of F and add edges $\{u, v\}$ into F for all boundary vertices $v \neq u$ of F that are not already adjacent to u . We finally choose, for each vertex of peeling number $\ell \geq 2$, arbitrarily one of its neighbors with a smaller peeling number as its down vertex. \square

For an embedded graph (G, φ) with $G = (V, E)$ and two vertices s and t of V , let us define an *s-t-ridge* as a path starting in s and ending in t that among all paths with these properties has maximal depth. Ridges will later be useful for finding a special kind of a separator with nice properties. This is the reason for considering them in the next lemma. For two vertices u and v in a triconnected embedded graph (G, φ) and a connection face F of u and v , let us denote for the next lemma the two paths from u to v on the boundary of F by $P^1(F, u, v)$ and $P^2(F, u, v)$.

Lemma 3.3.8. *Let $(G', \varphi', \downarrow)$ be the down closure of a triconnected embedded graph (G, φ) with $G = (V, E)$ and $s, t \in V$. For each s-t-ridge R in (G', φ') , there exists an s-t-path in (G, φ) having the same depth as R .*

Proof. Take $G' = (V, E')$. We only have to show how to replace the edges $\{u, v\} \in E' \setminus E$ used by R by a u - v -path in G of depth at least $\text{depth}(R)$. Since two consecutive vertices u and v of R must be face-adjacent in (G, φ) , the idea is to replace the edge $\{u, v\}$ by one of the two paths $P^1(F, u, v)$ and $P^2(F, u, v)$ for some connection face F of u and v . If afterwards some vertices are visited several times, the path could be shortened appropriately. The idea described above works, if for every edge $\{u, v\} \in E' \setminus E$, there is a connection face F with one of the two paths $P^1(F, u, v)$ and $P^2(F, u, v)$ containing only vertices of peeling number at least $i = \min(\mathcal{N}_\varphi(u), \mathcal{N}_\varphi(v))$. If $\mathcal{N}_\varphi(u) \neq \mathcal{N}_\varphi(v)$, all vertices incident to F have peeling number either i or $i + 1$, i.e., both paths $P^1(F, u, v)$ and $P^2(F, u, v)$ have the desired properties. Hence, let us assume that $\mathcal{N}_\varphi(u) = \mathcal{N}_\varphi(v)$. If we consider a construction of (G', φ') from (G, φ) as described in the definition of a down closure, it follows that immediately before the addition of edge $\{u, v\}$ into a face F' (covered by F), the boundary of F consists completely of vertices whose peeling numbers are at least i . Consequently, the same is true for the paths $P^1(F', u, v)$ and $P^2(F', u, v)$, and we can replace $\{u, v\}$ by a path P' from u to v in the graph immediately before the addition of $\{u, v\}$. If P' still contains edges in $E' \setminus E$, they can be replaced in the same way so that at the end we obtain a path in (G, φ) that is of the same depth as R . \square

For splitting an embedded graph (G, φ) into mountains, we use so-called crest separators, which we distinguish from the usual separators. By the way,

the dashed edges together with the dotted edges in Fig. 3.3.1 are the edges of three crest separators.

Definition 3.3.9 (crest separator, lowpoint, size, height). A *crest separator* in an embedded graph (G, φ) is a tuple $X = (L_1, L_2)$ consisting of the two lists $L_1 = (u_1, u_2, \dots, u_q)$ and $L_2 = (v_1, v_2, \dots, v_q)$ such that Properties 1-5 hold:

1. $u_1 = v_1$ or u_1 and v_1 are face-adjacent.
2. $\mathcal{N}_\varphi(u_i) = \mathcal{N}_\varphi(v_i) = q - i + 1$ for all $i \in \{1, \dots, q\}$.
3. u_i and u_{i+1} are face-adjacent for all $i \in \{1, \dots, q-1\}$.
4. v_i and v_{i+1} are face-adjacent for all $i \in \{1, \dots, q-1\}$.
5. If there is a $j \in \{2, \dots, q\}$ with $u_j = v_j$, then $u_i = v_i$ for all $i \in \{j, \dots, q\}$.

If the index j in Property 5 exists, for the smallest p with $u_p = v_p$, u_p is called the *lowpoint* of X . The *size* $|X|$ of X is defined as the number of different vertices in the union over all vertices in L_1 and in L_2 , and the *height* X is $\mathcal{N}_\varphi(u_1)$. The latter parameter is denoted by $\text{height}(X)$.

Since crest separators are in the main focus of our paper, it is useful to introduce some additional definitions: The *top vertices* of a crest separator $X = (L_1, L_2)$ consist of the first vertex in L_1 and the first vertex in L_2 —this means possibly only of one vertex. If we are given two alleys or paths $P_1 = (u_1, \dots, u_q)$ and $P_2 = (v_1, \dots, v_q)$ such that $X = ((u_1, \dots, u_q), (v_1, \dots, v_q))$ is a crest separator, we say that X is the *crest separator induced by P_1 and P_2* . If $X = (L_1, L_2)$ is a crest separator, we also say v is in X and write $v \in X$ if v is contained in one of the lists L_1 and L_2 .

For a crest separator $X = ((u_1, \dots, u_q), (v_1, \dots, v_q))$ of an embedded graph (G, φ) , the tuples $\{u_1, u_2\}, \dots, \{u_{q-1}, u_q\}$, $\{v_1, v_2\}, \dots, \{v_{q-1}, v_q\}$ and, in the case $u_1 \neq v_1$, also the tuple $\{u_1, v_1\}$ are called the *border edges* of X . The *border edges* of a set \mathcal{S} of crest separators are the union of the border edges of all crest separators in \mathcal{S} . The *border* of X is the graph consisting of all $v \in X$ and of all border edges. The *essential border* of X is the subgraph of the border of X induced by the vertex set $\{u_1, \dots, u_q, v_1, \dots, v_q\}$ if X has no lowpoint, and by $\{u_1, \dots, u_p, v_1, \dots, v_p\}$ if X has a lowpoint u_p . Moreover, we define the *crest alley* from u_i to u_j with $1 \leq i < j \leq q$ to be the alley $(u_i, u_{i+1}, \dots, u_j)$ and the *crest alley* from a vertex v_i to a vertex v_j with $\{v_i, v_j\} \neq \{u_i, u_j\}$ to be the alley $(v_i, v_{i+1}, \dots, v_j)$. For $i, j \in \{1, \dots, q\}$, the *crest alley* from a vertex $u_i \neq v_i$ to a vertex $v_j \neq u_j$ (or from the lowpoint u_i to the lowpoint v_j , i.e., $i = j$) is the alley $(u_i, u_{i-1}, \dots, u_1, v_1, \dots, v_{j-1}, v_j)$ if $u_1 \neq v_1$ or $(u_i, u_{i-1}, \dots, u_1, v_2, \dots, v_{j-1}, v_j)$ if $u_1 = v_1$. The reverse of the alleys above is defined to be the crest alleys from u_j to u_i , from v_j to v_i , and from v_j to u_i , respectively.

Moreover, for a set \mathcal{S} of crest separators of an embedded graph (G, φ) , we define $G \oplus \mathcal{S}$ to be the graph obtained from G by adding each border edge $\{u, v\}$ of \mathcal{S} into G if this edge does not already exist. Let φ' be a rooted combinatorial embedding of $G \oplus \mathcal{S}$ such that $\varphi'|_G = \varphi$. Then, for a subset $\mathcal{S}' \subseteq \mathcal{S}$, we define two inner faces F and F' of $(G \oplus \mathcal{S}, \varphi')$ to be (\mathcal{S}', φ') -connected if there is a

list (F_1, \dots, F_ℓ) ($\ell \in \mathbb{N}$) of inner faces of $(G \oplus \mathcal{S}, \varphi')$ with $F_1 = F$ and $F_\ell = F'$ such that, for each $i \in \{1, \dots, \ell - 1\}$, the faces F_i and F_{i+1} share a common boundary edge not being a border edge of \mathcal{S}' . A set \mathcal{F} of inner faces of $(G \oplus \mathcal{S}, \varphi')$ is (\mathcal{S}', φ') -connected if each pair of faces in \mathcal{F} is (\mathcal{S}', φ') -connected. A set of crest separators can be used to split a graph into a special kind of components:

Definition 3.3.10 ((\mathcal{S}', φ') -area, (\mathcal{S}', φ') -component). Let \mathcal{S}' be a subset of a set \mathcal{S} of crest separators of an embedded graph (G, φ) , and let φ' be an embedding of $G \oplus \mathcal{S}$ with $\varphi'|_G = \varphi$. For a maximal nonempty (\mathcal{S}', φ') -connected set \mathcal{F} of inner faces of $(G \oplus \mathcal{S}, \varphi')$, the area consisting of the union of the faces in \mathcal{F} is called an (\mathcal{S}', φ') -area and, for this area A , the subgraph of $G \oplus \mathcal{S}$ consisting of the vertices and the edges incident to A is called an (\mathcal{S}', φ') -component.

Let \mathcal{S} be a set of crest separators in an embedded graph (G, φ) with $G = (V, E)$, let φ' be an embedding of $G \oplus \mathcal{S}$ with $\varphi'|_G = \varphi$, and let \mathcal{S}' be a subset of \mathcal{S} . Take A as an (\mathcal{S}', φ') -area A and C as the (\mathcal{S}', φ') -component consisting of the vertices and edges incident to A . We define the *boundary* and *fence* of C to be the boundary and fence of A . In addition, for a crest separator $X \in \mathcal{S}$, we call an $(\{X\}, \varphi')$ -area C -inner if it covers A , and call it C -outer if the other $(\{X\}, \varphi')$ -area completely covers A . We also say that a crest separator $X \in \mathcal{S}$ with $X = (L_1, L_2)$ goes weakly between two subsets $U_1, U_2 \subseteq V$ if no $(\{X\}, \varphi')$ -component contains vertices of both, $U_1 \setminus (L_1 \cup L_2)$ and $U_2 \setminus (L_1 \cup L_2)$. X goes strongly between U_1 and U_2 if $L_1 \cup L_2$ additionally does not contain any vertex of $U_1 \cup U_2$. The tuple (\mathcal{S}, φ') is called *crossing-free* if, for each pair of crest separators $X_1, X_2 \in \mathcal{S}$, the set consisting of all vertices v in X_1 , but not in X_2 is completely contained in one $(\{X_2\}, \varphi')$ -component. We often say that \mathcal{S} is crossing-free if (\mathcal{S}, φ') is crossing-free and if φ' is clear from the context.

An alley *crosses* a crest separator $X = ((u_1, \dots, u_q), (v_1, \dots, v_q))$ if it crosses the crest alley from u_q to v_q in the case, where X has no lowpoint, or the crest alley from u_p to $v_p = u_p$ in the case, where u_p is the lowpoint of X . Note that for a triconnected embedded graph (G, φ) and a crest separator X of (G, φ) , $G \oplus \{X\}$ has only two rooted embeddings φ' and its inverse. This means that an alley P in a triconnected embedded graph (G, φ) crosses a crest separator X of (G, φ) if and only if it is not possible to choose one $(\{X\}, \varphi')$ -area A such that P is an A -alley.

For decomposing a graph into mountains, we later use a so-called mountain structure. Informally, a mountain structure should be a tuple (\mathcal{S}, φ') with \mathcal{S} being a crossing-free set of crest separators of an embedded graph (G, φ) and with φ' being an embedding of $G \oplus \mathcal{S}$ such that each (\mathcal{S}, φ') -component has only one crest with respect to the peeling numbers in $(G \oplus \mathcal{S}, \varphi')$. The problem is that, if we consider an (\mathcal{S}, φ') -component C as an isolated graph without the vertices of $G \oplus \mathcal{S}$ not contained in C , the peeling number of a vertex in the (\mathcal{S}, φ') -component may completely different from the peeling number of the same vertex in the original graph $(G \oplus \mathcal{S}, \varphi')$. We therefore may obtain crests for the components that are totally different from the crests of G . To solve this problem, we later define so-called extended components. Such a component will be obtained from an (\mathcal{S}, φ') -component by adding some extra vertices and edges such that, apart from the newly introduced vertices, the peeling numbers of all other vertices in the extended (\mathcal{S}, φ') -component with respect to the embedding

of the extended component are the same as the peeling numbers with respect to the embeddings φ' and φ of $G \oplus \mathcal{S}$ and G . However, an extended component will have a much simpler structure than the whole graph $(G \oplus \mathcal{S}, \varphi')$, namely, it is a mountain.

One problem is that it is not so easy to find a set of crest separators that is crossing-free and that decomposes a graph into mountains. We therefore use a very special algorithm to construct a certain set \mathcal{S} of crest separators. Unfortunately, for guaranteeing the correctness of this algorithm and several nice properties of the output of the algorithm, it is necessary to start it on the down closure $(G', \varphi', \downarrow)$ of an embedded graph (G, φ) instead of the original graph itself. This means that the algorithm outputs a set of crest separators for (G', φ') instead of for (G, φ) . The next lemma shows that this is not a real problem.

Lemma 3.3.11. *If $(G', \varphi', \downarrow)$ is a down closure of an embedded graph (G, φ) , each crest separator of (G', φ') is also a crest separator of (G, φ) .*

Proof. The introduction of edges into an embedded graph cannot make two vertices face-adjacent that were not face-adjacent before the addition. This together with the fact that the peeling numbers do not change (Lemma 3.3.1) shows the lemma. \square

Let us define a *down path* from a vertex v with a peeling number $i \in \mathbb{N}$ in the down closure $(G', \varphi', \downarrow)$ of an embedded graph (G, φ) as the path of length i whose j th vertex ($j \in \{1, \dots, i+1\}$) is the vertex $\downarrow^{j-1}(v)$, i.e., obtained from $j-1$ applications of \downarrow . We use the following algorithm to construct a crossing-free set \mathcal{S} of crest separators for the graph (G', φ') of the down closure $(G', \varphi', \downarrow)$ of an embedded graph (G, φ) .

set **function** MS(down closure $(G', \varphi', \downarrow)$)

Step 1: Take $G' = (V, E')$ and initially set $\mathcal{S} = \emptyset$.

Step 2: For each vertex $u \in V$ of peeling number $\ell \geq 2$, we partition its neighbors in G' into sets of maximal size such that the vertices of each set appear around u immediately one after each other and have either all peeling number $\ell-1$ or all peeling number $\geq \ell$. For each of these sets containing a vertex of peeling number $\ell-1$, but not the down vertex $u\downarrow$, we fix one vertex v of the set as *representant* around u for the vertices in the set. More exactly, to have a unique computation, let us always choose the vertex v as representant that appears around u immediately after a vertex with peeling number ℓ . The down path from u and the path obtained by concatenating $\{u, v\}$ with the down path from v induces a crest separator X that is added to \mathcal{S} . We call the edge $\{u, v\}$ the *top edge* of X .

Step 3: For each horizontal edge $\{u, v\} \in E'$ connecting two vertices of peeling number $\ell \geq 2$, the two down paths from u and from v induce a crest separator that is added to \mathcal{S} . In this case, we call $\{u, v\}$ the *top edge* of the crest separator.

Step 4: Return \mathcal{S} .

Note that the top edges of the crest separators in Fig. 3.3.1 are exactly the dashed edges. It is easy to see that each element of the set \mathcal{S} constructed by MS is indeed a crest separator, and that each top edge is on the boundary of two

(\mathcal{S}, φ) -components. Since the vertices of each crest separator in \mathcal{S} are exactly the vertices of two down paths, \mathcal{S} must be crossing-free. Given the down closure, the top edge of a crest separator constructed by the algorithm MS uniquely defines the crest separator since the remaining part can be constructed with the function \downarrow . We can therefore conclude:

Lemma 3.3.12. *The running time of the algorithm MS for a down closure $(G', \varphi', \downarrow)$ of a triconnected embedded ℓ -outerplanar graph (G, φ) with $G = (V, E)$ is $O(\ell|V|)$.*

Proof. Each of the $O(|V|)$ edges of G' can define at most one top edge of a crest separator. Since the height of such a crest separator is $O(\ell)$, the crest separator itself can be outputted following the down path in $O(\ell)$ time. \square

The (\mathcal{S}', φ') -components can be computed efficiently, too.

Lemma 3.3.13. *Given a set of crest separators \mathcal{S} of height at most q constructed by the algorithm MS for the down closure $(G', \varphi', \downarrow)$ of a triconnected embedded graph (G, φ) with $G = (V, E)$, a subset $\mathcal{S}' \subseteq \mathcal{S}$, and an embedding φ' of $G \oplus \mathcal{S}$ with $\varphi'|_G = \varphi$, one can construct in $O(q|V|)$ time all (\mathcal{S}', φ') -components C and store the set of all top edges contained in C . In addition, we can store the two components for each top edge to which it belongs.*

Proof. First determine the connected components in the graph obtained from the dual graph H of $(G \oplus \mathcal{S}, \varphi')$ after removing the vertex for the outer face with its incident edges as well as all edges of the dual graph that crosses a border edge of a crest separator $X \in \mathcal{S}'$. Note that each connected component of the remaining subgraph of the dual graph represents exactly one (\mathcal{S}', φ') -area A by its vertices or, more precisely, all inner faces of (G, φ) contained in A . For each such area A , compute first all vertices and edges incident to A as the vertices and edges of one (\mathcal{S}', φ') -component C , determine second all top edges of C , and store third, for each top edge e in C , C as one component to which e belongs. It is easy to see that almost all steps above can be implemented in $O(|V|)$ time. However, for the construction of H , we need to know all border edges. To compute the set of border edges, we must consider each crest separator in \mathcal{S}' .

Since each crest separator can consist of $O(q)$ vertices that need not be disjoint to the vertices of other crest separators, the computation of H can be done in $O(q|\mathcal{S}'|) = O(q|V|)$ time. For the last equality, we use the fact that all top edges must be pairwise disjoint, i.e., $|\mathcal{S}'| = O(|E|) = O(|V|)$. \square

Lemma 3.3.14 and 3.3.15 describe important properties of the set of crest separators computed by MS on the down closure (G', φ', d) of a triconnected embedded graph (G, φ) with $G = (V, E)$.

Lemma 3.3.14. *The top edge of a crest separator $X \in \mathcal{S}$ is disjoint from all border edges of all other crest separators in \mathcal{S} .*

Proof. The lemma follows from the fact that the top edges of the crest separators are pairwise disjoint and the fact that no top edge is a down edge and therefore part of a down path. \square

Lemma 3.3.15. *For $s, t \in V$, let R be an s - t -ridge in G' that has the smallest number of vertices with a peeling number equal to its depth among all s - t -ridges.*

Take v as a vertex of R with smallest peeling number. If the depth of R is smaller than the peeling numbers of both, s and t , then there is a crest separator $X \in \mathcal{S}$ that goes strongly between $\{s\}$ and $\{t\}$ with v being a top vertex of X and with its top edge not being part of R .

Proof. Let us say that an inner vertex u of R with a peeling number i is *dominated on the right (left) of R* if all edges not on R leaving u to the right (left) of R lead to a vertex with peeling number $i + 1$ and if u on the right (left) side of R is not incident to the outer face. Note that this latter condition is superfluous in the case $i \geq 2$. Let i be the peeling number of v .

The first case that we consider is the case, where $i \geq 2$ and v is dominated neither on the left nor on the right of R . In particular, v is connected by an edge to a vertex $u \in V \setminus \{s, t\}$ with peeling number $i - 1$ or i leaving R on a different side than $v \downarrow$. If $\mathcal{N}_\varphi(u) = i - 1$, there is a crest separator $X \in \mathcal{S}$ using $\{v, u'\}$ as top edge, where u' is the representant of u around v . Note that X goes strongly between $\{s\}$ and $\{t\}$ since $v \downarrow$ and u' are on different sides with respect to R and since R is a path not visiting any vertex of peeling number $i - 1$, i.e., R cannot be crossed by a down path from $v \downarrow$ or from u' . If $\mathcal{N}_\varphi(u) = i$, there is a crest separator $X \in \mathcal{S}$ using $\{v, u\}$ as top edge. In this case, u and $v \downarrow$ leave on different sides of R . Again, X goes strongly between $\{s\}$ and $\{t\}$.

In the second case, we assume that $i = 1$ and v is dominated neither on the left nor on the right of R . Then v is incident to the outer face and, possibly, to an edge $\{u, v\}$ that is not incident to the outer face. Since v is not dominated on any side, u must have peeling number 1. This means that $\{v\}$ or, if such a vertex u exists, $\{u, v\}$ is a separator of size at most 2. A contradiction to the triconnectivity of G' .

Finally, we consider the remaining case in which v is dominated on the left or right side of R . Since G' is triangulated, we then can replace v by a list L of its neighbors all having a larger peeling number than v and obtain so an s - t -path R' either having a larger depth than R or having the same depth as R , but a smaller number of vertices with a peeling number equal to the depth of R . In both cases, this contradicts our choice of R . Note explicitly that this is true even if L contains s or t since we then obtain an s - t -ridge contradicting our choice of R by shortening R' . \square

The idea is now to use the lemma above to split a graph into several parts such that no part contains the vertices of more than one crest. In detail, for the down closure $(G', \varphi', \downarrow)$ of a triconnected embedded graph (G, φ) and a subset \mathcal{S}^* of the set \mathcal{S} of crest separators constructed by the algorithm MS on $(G', \varphi', \downarrow)$, we call the sextuple $(G, \varphi, G', \varphi', \downarrow, \mathcal{S}^*)$ to be a *mountain structure* if, for each pair of different crests H_1 and H_2 in (G, φ) , there is a crest separator $X \in \mathcal{S}^*$ such that

- X goes strongly between H_1 and H_2
- the height of X is equal to the depth of an s - t -ridge in (G', φ') for some $s \in H_1$ and $t \in H_2$ —and hence for all $s \in H_1$ and $t \in H_2$.

The next lemma shows that the set of crest separators constructed by the algorithm MS defines a mountain structure.

Lemma 3.3.16. *For the down closure $(G', \varphi', \downarrow)$ of a triconnected embedded graph (G, φ) and the set \mathcal{S} constructed by the algorithm MS on $(G', \varphi', \downarrow)$, the tuple $(G, \varphi, G', \varphi', \downarrow, \mathcal{S})$ is a mountain structure.*

Proof. Let H_1 and H_2 be two different crests of (G, φ) , and let $s \in H_1$ as well as $t \in H_2$. Corollary 3.3.3 implies that H_1 and H_2 define also different crests in (G', φ') . Hence the depth of an s - t -ridge in (G', φ') is smaller than $\min(\mathcal{N}_{\varphi'}(s), \mathcal{N}_{\varphi'}(t))$, i.e., by Lemma 3.3.15 there must be a crest separator $X \in \mathcal{S}$ that goes strongly between s and t in (G', φ') and hence also in (G, φ) . Note that the height of X is smaller than the peeling number of s and t . Therefore, X also goes strongly between H_1 and H_2 . \square

The next two lemmata help us later to restrict the set \mathcal{S} computed by MS to a smaller set $\mathcal{S}' \subset \mathcal{S}$ such that $(G, \varphi, G', \varphi', \downarrow, \mathcal{S}')$ is still a mountain structure, but has some further nice properties.

Lemma 3.3.17. *Let $(G, \varphi, G', \varphi', \downarrow, \mathcal{S})$ be a mountain structure, and let C be an (\mathcal{S}, φ') -component that contains no vertex of a crest in (G', φ') . If X is the crest separator of largest height among all crest separators in \mathcal{S} whose top edge belongs to C , the tuple $(G, \varphi, G', \varphi', \downarrow, \mathcal{S} \setminus \{X\})$ is a mountain structure, too.*

Proof. If $(G, \varphi, G', \varphi', \downarrow, \mathcal{S} \setminus \{X\})$ is not a mountain structure, there must be two vertices s and t part of different crests in (G', φ') such that X goes strongly between $\{s\}$ and $\{t\}$ and such that there is an s - t -ridge R whose depth is equal to the height of X . Let S and T be the crests containing s and t , respectively. W.l.o.g. let s be the vertex contained in the C -inner $(\{X\}, \varphi')$ -area A . Since C does not contain any vertex of S , there must be a further crest separator X' without any vertex of S that goes weakly between S and the vertices of C . This crest separator goes strongly between S and T since C also contains no vertex of T . By our choice of X , $\text{height}(X') \leq \text{height}(X) = \text{depth}(R)$. Since X' crossing R implies that $\text{height}(X') \geq \text{depth}(R)$, we have $\text{height}(X') = \text{depth}(R)$. Thus, we can remove X from \mathcal{S} and still have a mountain structure. \square

Lemma 3.3.18. *If $(G, \varphi, G', \varphi', \downarrow, \mathcal{S})$ is a mountain structure and if X is a crest separator in \mathcal{S} such that its top vertices are part of a crest of (G', φ') , then $(G, \varphi, G', \varphi', \downarrow, \mathcal{S} \setminus \{X\})$ is a mountain structure, too.*

Proof. Assume that $(G, \varphi, G', \varphi', \downarrow, \mathcal{S} \setminus \{X\})$ is not a mountain structure. This implies that there must be two vertices s and t part of different crests in (G', φ') such that X goes strongly between $\{s\}$ and $\{t\}$ and such that there is an s - t -ridge R whose depth is equal to the height of X . Let u be a vertex common of X and R . Since $\text{depth}(R) = \text{height}(X)$, u must be a top vertex of X . By our choice of X , u must be part of a crest. However, this crest must be different from the crests containing s and t —otherwise X cannot strongly disconnect these crests. Therefore, the subpath of R from s to u has a vertex with a peeling number strictly smaller than u , i.e., the depth of R is smaller than the height of X . Contradiction. \square

For an (\mathcal{S}, φ') -component C of a mountain structure $(G, \varphi, G', \varphi', \downarrow, \mathcal{S})$, there can exist a crest separator $X \in \mathcal{S}$ with a lowpoint such that, for the $(\{X\}, \varphi')$ -area A that has no edge with the outer face of (G', φ') in common, C is covered in A . In this case, there is only one such crest separator and we say that X encloses C . We are now ready to define extended (\mathcal{S}, φ') -components.

Definition 3.3.19 (extended (\mathcal{S}, φ') -component). Let C be an (\mathcal{S}, φ') -component of a mountain structure $(G, \varphi, G', \varphi', \downarrow, \mathcal{S})$, and let $\mathcal{S}' \subseteq \mathcal{S}$ be the set of crest separators with a top edge in C . Moreover, if C is enclosed by a crest separator $X \in \mathcal{S}'$, let $\mathcal{S}^* = \{X\}$ and otherwise define $\mathcal{S}^* = \emptyset$. The *extended (\mathcal{S}, φ') -component* (C^+, φ_{C^+}) of C is the embedded graph obtained from $(C, \varphi'|_C)$ as follows:

- (1) For each crest separator $X \in \mathcal{S}'$, extend C by all $v \in X$ and all border edges of X that are not already contained in C .
- (2) For each $X = ((u_1, \dots, u_q), (v_1, \dots, v_q)) \in \mathcal{S}'$, add new vertices x_i and y_i as well as edges $\{u_i, x_i\}, \{x_i, y_i\}, \{y_i, v_i\}$ into the C -outer $(\{X\}, \varphi')$ -area for all $i \in \{2, \dots, j\}$, where $j = q$ if X has no lowpoint, and where j is chosen such that v_{j+1} is the lowpoint of X otherwise.
- (3) If, for a crest separator $X \in \mathcal{S}'$ with two top vertices, the above steps introduced a new edge (x_2, y_2) —which is mostly the case—add a further new vertex z_2 , and replace the edge (x_2, y_2) by two edges (x_2, z_2) and (z_2, y_2) .
- (4) If $\mathcal{S}^* = \{X\}$, take $X = ((u_1, \dots, u_q), (v_1, \dots, v_q))$, and add new vertices x_i and y_i as well as edges $\{u_i, x_i\}, \{x_i, y_i\}$, and $\{y_i, v_i\}$ into the C -outer $(\{X\}, \varphi')$ -area for all $i \in \{j, \dots, q\}$ with u_j being the lowpoint of X .

If \mathcal{S} and φ' is not of interest or clear from the context, we call an extended (\mathcal{S}, φ') -component sometimes simply *extended component*. The vertices and edges of (C^+, φ_{C^+}) introduced in the steps (2), (3), and (4) are called the *virtual vertices* and *virtual edges*, respectively, of (C^+, φ_{C^+}) . It is easy to see that the combinatorial embedding obtained from $\varphi'|_C$ by adding the virtual edges introduced for one crest separator X into the C -outer $(\{X\}, \varphi')$ -area is unique. However, to show that the virtual edges for different crest separators are crossing-free, we prove in the next lemma that the inner faces, into which we embed the virtual edges, are different for different crest separators.

Lemma 3.3.20. *Let C be one of the (\mathcal{S}, φ') -components of a mountain structure $(G, \varphi, G', \varphi', \downarrow, \mathcal{S})$. For each pair of different crest separators $X_1, X_2 \in \mathcal{S}$ with top edges in C , the C -outer (X_1, φ') -area A_1 and the C -outer (X_2, φ') -area A_2 are disjoint, except for possibly some common fence vertices or fence edges.*

Proof. Since \mathcal{S} is crossing-free, the vertices that belong to X_1 , but not to X_2 are completely incident to one $(\{X_2\}, \varphi')$ -area A'_2 and the vertices that belong to X_2 , but not to X_1 are completely incident to one $(\{X_1\}, \varphi')$ -area A'_1 . Since crest separators beside their top edges only consist of down edges, there are exactly three $(\{X_1, X_2\}, \varphi')$ -components: One component C' consisting of all vertices and edges being incident to both A'_1 and A'_2 , another consisting of all vertices and edges being incident to the $(\{X_1\}, \varphi')$ -area $A''_1 \neq A'_1$ and the third consisting of all vertices and edges being incident to the $(\{X_2\}, \varphi')$ -area $A''_2 \neq A'_2$. The different components are disjoint, except for possibly some common fence edges and fence vertices. The only component containing the top edges of both, X_1 and X_2 , is C' . Therefore, C must be a subgraph of C' , and, for $i \in \{1, 2\}$, A''_i must be the C -outer (X_i, φ') -area. The lemma now follows from the fact

that A_1'' and A_2'' are disjoint, except for possibly some fence vertices and fence edges. \square

We also want to remark that the reason for connecting two vertices u_i and v_i of a crest separator $X = ((u_1, \dots, u_q), (v_1, \dots, v_q))$ in an extended (\mathcal{S}, φ) -component not by a single virtual edge $\{u_i, v_i\}$, but by a path of virtual edges, is that we want to add additional edges into an extended component in the next section. However, the results of this section would also hold if we introduce only one virtual edge $\{u_i, v_i\}$ instead of a path from u_1 to v_i visiting further vertices. As we have seen in Lemma 3.3.13, the (\mathcal{S}, φ) -components can be computed efficiently. The same holds for the extended (\mathcal{S}, φ) -components.

Lemma 3.3.21. *Given a mountain structure $(G, \varphi, G', \varphi', \downarrow, \mathcal{S})$ with $G = (V, E)$ such that each crest separator of \mathcal{S} has height at most q , one can construct all extended (\mathcal{S}, φ) -components in $O(q|V|)$ time.*

Proof. First, compute all (\mathcal{S}, φ) -components and, for each such (\mathcal{S}, φ) -component $C = (V', E')$, the set of all top edges contained in C to determine the set \mathcal{S}' as defined in Definition 3.3.19. This can be done in $O(q|V|)$ time (Lemma 3.3.13). Second, apply the steps (1)-(4) of Definition 3.3.19. Since G is planar, we can have only $O(|E|) = O(|V|)$ top edges, and therefore our computation takes $O(q|V|)$ time. \square

The main reason for introducing extended (\mathcal{S}, φ') -components and using the name mountain structure is that the extended (\mathcal{S}, φ') -components of a mountain structure are mountains. This is shown in Lemma 3.3.24 by using Lemma 3.3.23. We start with an auxiliary lemma.

Lemma 3.3.22. *Let $(G, \varphi, G', \varphi', \downarrow, \mathcal{S})$ be a mountain structure, and let C be an (\mathcal{S}, φ') -component. Each fence edge of C is a border edge of a crest separator with a top edge in C .*

Proof. Let us consider a fence edge $e = \{u, v\}$ of C . It is clear that e must be a border edge of a crest separator $X \in \mathcal{S}$, but let us assume for a contradiction that it is not part of a crest separator in \mathcal{S} with a top edge in C . Hence e must be a down edge. Let us further assume w.l.o.g. that among all possible choices we have chosen e such that the peeling numbers of the endpoints of e are as large as possible, say $i = \mathcal{N}_{\varphi'}(u)$ and $\mathcal{N}_{\varphi'}(v) = i - 1$. Let us consider the other fence edge e' of C incident to u . If e' is the top edge of a crest separator $X \in \mathcal{S}$, e is also part of X and we obtain a contradiction. Thus, e' must be a down edge of a crest separator $X \in \mathcal{S}$ without a top edge in C . Since v is the down vertex of u , the endpoint of e' opposite to u must have peeling number $i + 1$. This contradicts our choice of e . \square

Lemma 3.3.23. *For a mountain structure $(G, \varphi, G', \varphi', \downarrow, \mathcal{S})$, the peeling numbers of the non-virtual vertices of the extended (\mathcal{S}, φ') -component (C^+, φ_{C^+}) of an (\mathcal{S}, φ') -component C are the same with respect to φ_{C^+} as with respect to φ' . Moreover, the virtual edges of (C^+, φ_{C^+}) are horizontal.*

Proof. It is easy to see that the peeling numbers of all non-virtual vertices of C^+ are the same with respect to φ_{C^+} as with respect to φ' if the same holds only for all fence vertices of C . Hence by Lemma 3.3.22, it suffices to consider only the peeling numbers of all endpoints of border edges of a crest separator

with a top edge in C . Since the border edges are contained in C^+ and since the peeling numbers of the endpoints of a border edge differ by exactly one with respect to φ' —except for possibly for a top edge, which by definition must connect two vertices of the same peeling number—it is also easy to see that the peeling numbers of these vertices cannot increase when switching from φ' to φ_{C^+} . Hence let us assume that, for a crest separator $X \in \mathcal{S}$ with a top edge in C , there is a vertex in X whose peeling number decreases when switching from φ' to φ_{C^+} . In other words, there exists a vertex $u \in X$ such that u is face-adjacent with respect to φ_{C^+} to a vertex v that either is a vertex of G' with $\mathcal{N}_{\varphi'}(v) \leq \mathcal{N}_{\varphi'}(u) - 2$ or is a virtual vertex v with $\mathcal{N}_{\varphi_{C^+}}(v) \leq \mathcal{N}_{\varphi'}(u) - 2$. Let F be a connection face of u and v in (C^+, φ_{C^+}) . Then there must be a crest separator $Y \in \mathcal{S}$ with a top edge in C such that F is contained in the C -outer $(\{Y\}, \varphi')$ -area. However, we added into this C -outer $(\{Y\}, \varphi')$ -area paths consisting of virtual edges connecting the vertices of Y with the same peeling number with respect to φ' . By an induction over the peeling number of the endpoints of these paths, one can easily observe that the vertices of such a path have all the same peeling number with respect to φ_{C^+} . We therefore can conclude that the virtual edges are horizontal and that there cannot be a connecting face of u and v (contradiction). Thus, the peeling numbers cannot decrease when switching from φ' to φ_{C^+} . \square

Lemma 3.3.24. *If $(G, \varphi, G', \varphi', \downarrow, \mathcal{S})$ is a mountain structure, each extended (\mathcal{S}, φ') -component of G is a mountain.*

Proof. Let C be an (\mathcal{S}, φ') -component, and let (C^+, φ_{C^+}) be the extended (\mathcal{S}, φ') -component of C . Take $G' = (V, E')$ and $C^+ = (V^+, E^+)$. Assume that there are vertices s and t belonging to two different crests in (C^+, φ_{C^+}) , and let $i = \mathcal{N}_{\varphi_{C^+}}(s)$ and $j = \mathcal{N}_{\varphi_{C^+}}(t)$. W.l.o.g. we can assume that s and t are non-virtual. Otherwise, there is a path that has s (or t) and a non-virtual vertex as endpoints and that consists only of horizontal edges (Lemma 3.3.23). Therefore, we can replace s (or t) by the other endpoint of the path.

We next show that each s - t -ridge in (G', φ') has a depth lower than $\min\{i, j\}$. For a contradiction, let us assume that there is a ridge R' in (G', φ') of depth at least $\min\{i, j\}$. Our intermediate goal is to show that we can replace R' by a ridge R of depth at least $\min\{i, j\}$ in (G', φ') that only uses edges of C , i.e., edges in $E' \cap E^+$. Let $e = \{u, v\}$ be the first edge of R' in $E' \setminus E^+$ with u being visited before v . By Lemma 3.3.22 each fence edge of C is a border edge of a crest separator in \mathcal{S} with its top edge in C . Hence, there is a crest separator $X' = (L_1, L_2) \in \mathcal{S}$ with a top edge in C such that $u \in X'$ with e being not incident to the C -inner $(\{X'\}, \varphi')$ -area. Let u' be the last vertex of R' in X' . Note that $u' \neq u$ and that u and u' have both a peeling number of at least $\min\{i, j\}$. If X' has a lowpoint, u and u' cannot appear in L_1 or L_2 after the lowpoint by their definition. This implies that there is a crest alley P from u to u' . We next replace the subpath of R' from u to u' by P and call the resulting ridge R'' . Let $\mathcal{S}_{R'}$ and $\mathcal{S}_{R''}$ be the set of crest separators $X \in \mathcal{S}$ with a top edge in C , for which R' and R'' , respectively, visits an edge that is not incident to the C -inner $(\{X\}, \varphi')$ -area. Note that the subpath P may contain new edges not being contained in C . However, for each edge of P that is not incident to the C -inner $(\{X\}, \varphi')$ -area of a crest separator $X \in \mathcal{S}$ with a top edge in C , it follows from \mathcal{S} being crossing-free that also the edge e is not

incident to the C -inner $(\{X\}, \varphi')$ -area. Consequently, $\mathcal{S}_{R''} \subseteq \mathcal{S}_{R'}$. Since P does not contain any new edges that are not part of the C -inner area of X' , we even have $\mathcal{S}_{R''} \subseteq \mathcal{S}_{R'} \setminus \{X'\}$. This means that after a finite number of replacements as described above, we obtain a ridge R that has the same depth as R' and that only uses edges of C . Since R is a path of depth $\min\{i, j\}$ in (C^+, φ_{C^+}) , s and t cannot belong to different crests in (C^+, φ_{C^+}) . By this contradiction, we can conclude that each s - t -ridge R' has a depth lower than $\min\{i, j\}$.

Let us choose an s - s' -path P_s and a t - t' -path P_t both in (G', φ') such that s' and t' belong to a crest in (G', φ') and such that the peeling numbers do not decrease along these paths, possibly $s' = s$ or $t' = t$. An s' - t' -ridge R' in (G', φ') must have a depth $d < \min\{i, j\}$ since otherwise there would be an s - t -path in (G', φ') of depth at least $\min\{i, j\}$ using P_s , R' and P_t . Hence, s' and t' belong to different crests of (G', φ') and by our definition of a mountain structure there is a crest separator X of height d going strongly between $\{s'\}$ and $\{t'\}$. Since all vertices of P_s and P_t have a peeling number greater than d , X can have no common vertices with P_s and P_t . Consequently, X also goes strongly between $\{s\}$ and $\{t\}$ and they cannot both belong to C^+ . \square

We next prove some further simple and useful properties of mountain structures. Let us define the *mountain connection tree* T of a mountain structure $(G, \varphi, G', \varphi', \downarrow, \mathcal{S})$ as follows: Each (\mathcal{S}, φ') -component is identified with a node of T and two nodes w_1 and w_2 of T are connected if and only if they have a common top edge. Since the crest separators constructed by the algorithm MS are crossing-free and since they are usual separators the following lemma holds.

Lemma 3.3.25. *The mountain connection tree is a tree.*

Proof. Take T to be the mountain connection tree of a mountain structure $(G, \varphi, G', \varphi', \downarrow, \mathcal{S})$. For each (\mathcal{S}, φ') -component C , let us call the (\mathcal{S}, φ') -area A_C . We first show that T is connected. Otherwise, consider a connected component T' of T , and let \mathcal{A} be the area obtained from the union of all (\mathcal{S}, φ') -areas A_C over all (\mathcal{S}, φ') -components C that are vertices of T' . Since G is triconnected and since T' is not the only connected component of T , the boundary edges of \mathcal{A} cannot completely consist of edges that are incident to the outer face in (G', φ') . Note that all fence edges of \mathcal{A} are fence edges of an area A_C with C being contained in T' . Since T' is a maximal connected subgraph of T such a fence edge cannot be a top edge of a crest separator. Therefore, all fence edges of \mathcal{A} must be down edges. But this is not possible since this would imply that there is at least one vertex v incident to two down edges—instead of only one defined—on the boundary both connecting v with a vertex of lower peeling number.

Assume that T is connected but not a tree, i.e., we have (\mathcal{S}, φ') -components $C_1, \dots, C_z, C_{z+1} = C_1$ with $z \geq 2$ that in this order induce a cycle in T . Let $X \in \mathcal{S}$ be the crest separator whose top edge e belongs to C_1 and to C_2 . As all crest separators do, X splits (G', φ') into two $(\{X\}, \varphi')$ -areas. Since, for an (\mathcal{S}, φ') -component C , the inner faces of $\varphi'|_C$ cannot belong to both $(\{X\}, \varphi')$ -areas, there must be another top edge e' belonging to C_i and to C_{i+1} for some $i \geq 2$ such that C_i and C_{i+1} are part of different $(\{X\}, \varphi')$ -areas. Consequently, e' must be a border edge of X . This fact contradicts Lemma 3.3.14. \square

For the rest of this section, we consider the mountain connection tree as a rooted tree by choosing an arbitrary node of T as root.

Lemma 3.3.26. *Given a mountain structure $(G, \varphi, G', \varphi', \downarrow, \mathcal{S})$ with $G = (V, E)$ such that each crest separator of \mathcal{S} has height at most q , the mountain connection tree can be computed in $O(q|V|)$ time.*

Proof. In $O(q|V|)$ time, we can compute all (\mathcal{S}, φ) -components and a data-structure such that we can find out in constant time to which components each top edge belongs (Lemma 3.3.13). Afterwards, we can iterate over all top edges, and for each such top edge e , we can connect the two components containing e . At the end, we so obtain the mountain connection tree. \square

Lemma 3.3.27. *Let $(G, \varphi, G', \varphi', \downarrow, \mathcal{S})$ be a mountain structure. Moreover, let C_1 and C_2 be (\mathcal{S}, φ') -components. Assume that C_1 and C_2 contain a common top edge of a crest separator $X \in \mathcal{S}$. Let U be the set of all vertices of the essential border of X . Then the set of common vertices of C_1 and C_2 is a subset of U . Moreover, there is no edge between a vertex in $C_1 \setminus U$ and a vertex in $C_2 \setminus U$.*

Proof. Since C_1 and C_2 are different, but share the top edge of X , they must be covered by different $(\{X\}, \varphi')$ -areas with respect to φ' . Hence the common vertices of C_1 and C_2 are a subset of the vertices in X . From the definition of a lowpoint we can conclude that either the C_1 -inner $(\{X\}, \varphi')$ -area or the C_2 -inner $(\{X\}, \varphi')$ -area cannot be incident to vertices of peeling number strictly smaller than that of the lowpoint of X , i.e., in particular, the set of common vertices in C_1 and in C_2 is indeed a subset of U . Finally note that G being planar and X being a crest separator implies that there are no edges between a vertex in $C_1 \setminus U$ and a vertex in $C_2 \setminus U$. \square

Combining the last lemma with the fact that the border edges of a crest separator contains exactly one top edge, we obtain the next corollary.

Corollary 3.3.28. *Let T be a mountain connection tree of a mountain structure $(G, \varphi, G', \varphi', \downarrow, \mathcal{S})$. For each pair of incident (\mathcal{S}, φ') -components C_1 and C_2 of T , there is a unique crest separator with its top edge belong to both (\mathcal{S}, φ') -components, and this crest separator is also the only crest separator that goes between C_1 and C_2 .*

Even if each (\mathcal{S}', φ') -component of a mountain structure $(G, \varphi, G', \varphi', \downarrow, \mathcal{S}'')$ is a mountain, it possibly does not contain a crest of (G', φ') . Let \mathcal{S} be the set of crest separators computed by the algorithm MS on the input $(G', \varphi', \downarrow)$. Since (\mathcal{S}', φ') -components not containing any crests of (G', φ') are not very useful for our algorithm, we define a *good mountain structure* to be a mountain structure $(G, \varphi, G', \varphi', \downarrow, \mathcal{S}')$ —thus, $\mathcal{S}' \subseteq \mathcal{S}$ —if the following properties hold:

- (a) Let H_1 and H_2 be two crests in (G, φ) , and let d be the depth of a ridge in (G', φ') with one endpoint in H_1 and the other in H_2 . Then, there is a crest separator $X \in \mathcal{S}'$ of height d going strongly between the crests and having among all such crest separators of \mathcal{S} as few as possible top vertices.
- (b) No crest separator in \mathcal{S}' contains a vertex of a crest in (G, φ) .
- (c) Each (\mathcal{S}', φ') -component contains exactly the vertices of one crest in (G, φ) .

An additional property of a good mountain structure is described next.

Lemma 3.3.29. *Given a good mountain structure $(G, \varphi, G', \varphi', \downarrow, \mathcal{S})$ and a crest separator $X \in \mathcal{S}$ with a horizontal top edge e , we can conclude that e is not face-adjacent to any vertex of peeling number $\text{height}(X) - 1$.*

Proof. Let C_1 and C_2 be the two (\mathcal{S}, φ') -components containing the top edge of X . For $i \in \{1, 2\}$, let u_i be a vertex of the crest H_i that is contained in C_i , and define P to be a u_1 - u_2 -ridge in (G', φ') . The definition of a good mountain structure implies that, for X as the only crest separator in \mathcal{S} going strongly between H_1 and H_2 , we must have $\text{depth}(P) = \text{height}(X)$ and that X must have a minimal number of top vertices among all crest separators of height $\text{depth}(P)$ going strongly between H_1 and H_2 and being constructed by the algorithm MS. Take $h = \text{height}(X)$ as well as v_1 and v_2 as the endpoints of e . Assume now that e is face-adjacent to a vertex v of peeling number $h - 1$. Let F be a connection face of e and v in $(G \oplus \{X\}, \varphi_{G \oplus \{X\}})$, and let A be the (X, φ) -area containing F . Take F' as the face in (G', φ') having e as boundary edge and being contained in A and hence in F . Lemma 3.3.6 and the existence of a vertex of peeling number $h - 1$ on the boundary of F implies that the three boundary vertices of F' consist, beside v_1 and v_2 , of a vertex v_3 with peeling number $h - 1$. Note that each u_1 - u_2 -path must cross X , in particular, this is true for P . Due to this and since e and the down edges of v_1 and v_2 are border edges of X contained in G' , P as a ridge of depth $\text{height}(X)$ must visit at least one of the vertices v_1 and v_2 . Moreover, one can choose an $i \in \{1, 2\}$ such that the down edge of v_i , an edge of P incident to v_i , the edge $\{v_i, v_3\}$ and the other edge of P incident to v_i appear around v_i in this order. Take v'_3 as the representant of v_3 around v_i , and let P' be the concatenation of the edge $\{v_i, v'_3\}$ and of the down path of v'_3 . The algorithm MS would then construct a crest separator X' induced by the down path of v_i and P' . Clearly X' also goes strongly between H_1 and H_2 , but has a lower number of top vertices than X . This contradicts the fact that we are given a good mountain structure. \square

In the remainder of this section, we consider an algorithm that, for a given a triconnected embedded ℓ -outerplanar graph (G, φ) with $G = (V, E)$, computes a good mountain structure $(G, \varphi, G', \varphi', \downarrow, \mathcal{S})$. For a simpler notation in the following we call a crest separator of a set $\tilde{\mathcal{S}}$ of crest separators a *largest crest separator* of $\tilde{\mathcal{S}}$ if it has among all crest separators in $\tilde{\mathcal{S}}$ a largest height and among all crest separators of largest height a maximal number of top vertices.

We start with the description of a preprocessing phase. In this phase, we first compute a down closure $(G', \varphi', \downarrow)$ of (G, φ) in $O(\ell|V|)$ time and start the algorithm MS on $(G', \varphi', \downarrow)$ (Lemmata 3.3.7 and 3.3.12). Note that, for the set \mathcal{S} of crest separators output by the algorithm MS, each $X \in \mathcal{S}$ has height at most ℓ . In $O(|V|)$ time, we next determine all crests of (G', φ') and test for each crest separator $X \in \mathcal{S}$, whether its top vertices belong to a crest of (G', φ') . If so, we remove X from \mathcal{S} . By Lemma 3.3.18, for the resulting set \mathcal{S}^* of crest separators, the tuple $(G, \varphi, G', \varphi', \downarrow, \mathcal{S}^*)$ is still a mountain structure. We initialize \mathcal{S}' with \mathcal{S}^* . We also construct the mountain connection tree of $(G, \varphi, G', \varphi', \downarrow, \mathcal{S}^*)$ in $O(\ell|V|)$ time (Lemma 3.3.26). However, since we defined the nodes of a mountain connection tree to be the (\mathcal{S}', φ') -components such an update would be too expensive. Instead of the mountain connection tree, we thus use a tree T that is obtained from the mountain connection tree by replacing each (\mathcal{S}', φ') -component C by a new node w and we say that C corresponds to w .

Before describing several updates of T in detail, let us remark that the main part of the algorithm consists of a traversal of T in a bottom-up manner. During this traversal, we mark nodes of T as finished—by rules described later—and we always process a so far unfinished node whose children are already marked as finished and that among all such nodes have the largest depth in T . When processing a node w , we possibly remove a crest separator X from the current set \mathcal{S}' of crest separators with the top edge of X belonging to the two (\mathcal{S}', φ') -components corresponding to w and to a neighbor w' of w in T . If so, by the replacement of \mathcal{S}' by $\mathcal{S}' \setminus \{X\}$, we merge the nodes w and w' in T to a new node w^* . We also mark w^* as finished only if w' is a child of w .

Some more preprocessing steps are necessary. In $O(|V|)$ time, we determine and store with each node w of T a value $\text{Crest}(w) \in \{0, 1\}$ that is set to 1 if and only if the (\mathcal{S}', φ') -component corresponding to w contains a vertex that is part of a crest of (G', φ') . Within the same time, we mark additionally each node as unfinished and store with each non-leaf w of T in $\text{MaxCrestSep}(w)$ a largest crest separator of the set of all crest separators going weakly between the two (\mathcal{S}', φ') -components corresponding to w and to a child of w . For each leaf w of T , we define $\text{MaxCrestSep}(w) = \text{nil}$. As a last step of our preprocessing phase, which also runs in $O(|V|)$ time, we initialize, for each node w of T , a value $\text{MaxCrestSep}^*(w)$ with nil . $\text{MaxCrestSep}^*(w)$ is defined analogously as $\text{MaxCrestSep}(w)$ if we restrict the crest separators to be considered only to those crest separators that go weakly between the two (\mathcal{S}', φ') -components corresponding to w and to one of the children of w that have been marked as finished. We next describe the processing of a node w during the traversal of T in detail. Keep in mind that \mathcal{S}' is always equal to the set of remaining crest separators of \mathcal{S}^* .

If w has a parent \tilde{w} , we first test whether the crest separator stored in $\text{MaxCrestSep}(\tilde{w})$ is equal to the crest separator going weakly between the two (\mathcal{S}', φ') -components corresponding to w and to \tilde{w} . If so and if \tilde{w} has a further child \hat{w} marked as unfinished beside w , we delay the processing of w and first process \hat{w} . Hence assume that the case above is not given. Next test, whether the (\mathcal{S}', φ') -component C corresponding to w contains a vertex belonging to a crest of (G', φ') , which is exactly the case if $\text{Crest}(w) = 1$. In this case, we mark w as finished. If there is a parent \tilde{w} of w , some extra work is necessary. Take X as the crest separator going weakly between the two (\mathcal{S}', φ') -components corresponding to w and to \tilde{w} , and replace $\text{MaxCrestSep}^*(\tilde{w})$ by a largest crest separator in $\{X, \text{MaxCrestSep}^*(\tilde{w})\}$.

Let us next consider the case where $\text{Crest}(w) = 0$. We then remove a largest crest separator X of the set of all crest separators going weakly between the two (\mathcal{S}', φ') -components corresponding to w and to a neighbor of w . X must be either $\text{MaxCrestSep}(w)$ or the crest separator going weakly between the two (\mathcal{S}', φ') -components corresponding to w and to its parent. If X is a crest separator going weakly between the two (\mathcal{S}', φ') -components corresponding to w and to a child \hat{w} of w , we mark the node w^* obtained from merging w and \hat{w} as finished and set $\text{Crest}(w^*) = 1$. This is correct since \hat{w} is already marked as finished. In addition, if there is a parent \tilde{w} of w , we replace $\text{MaxCrestSep}^*(\tilde{w})$ by the largest crest separator in $\{X', \text{MaxCrestSep}^*(\tilde{w})\}$ for the crest separator X' going strictly between the (\mathcal{S}', φ') -components corresponding to w and \tilde{w} .

Otherwise, X is the crest separator going weakly between the two (\mathcal{S}', φ') -components corresponding to w and to its parent \tilde{w} . In this case, we mark

the node w^* obtained from merging w and \tilde{w} as unfinished, set $\text{Crest}(w^*) = \text{Crest}(\tilde{w})$, and define the value $\text{MaxCrestSep}^*(w^*)$ as the largest crest separator in $\{\text{MaxCrestSep}^*(\tilde{w}), \text{MaxCrestSep}^*(w)\}$ or nil if this set contains no crest separator. If \tilde{w} beside w has another unfinished child, we define $\text{MaxCrestSep}(w^*)$ as the largest crest separator in $\{\text{MaxCrestSep}(\tilde{w}), \text{MaxCrestSep}(w)\}$. Otherwise, we take $\text{MaxCrestSep}(w^*)$ as the largest crest separator contained in $\{\text{MaxCrestSep}^*(\tilde{w}), \text{MaxCrestSep}(w)\}$ or nil if no crest separator is in this set.

Lemma 3.3.30. *After the processing of each node, the tuple $(G, \varphi, G', \varphi', \downarrow, \mathcal{S}')$ is a mountain structure.*

Proof. We already know that $(G, \varphi, G', \varphi', \downarrow, \mathcal{S}')$ is a mountain structure after the initialization of \mathcal{S}' with \mathcal{S}^* . Because of Lemma 3.3.17, this is also true after the removal of a crest separator X if the algorithm correctly computes the values of MaxCrestSep and MaxCrestSep^* . More precisely, we only need the correct value of $\text{MaxCrestSep}(w)$ when processing the node w , which implies that w is not already marked as finished. For this reason, it suffices to maintain the values of MaxCrestSep and MaxCrestSep^* only for these nodes that are marked as unfinished. It is easy to see that our algorithm always maintains the correct values of $\text{MaxCrestSep}^*(w)$ for all nodes marked as unfinished, where we use the fact that the parent of an unfinished node is also marked as unfinished. The same could be easily shown to be true for the values of $\text{MaxCrestSep}(w)$, except that the update rule in the case of merging w with its parent \tilde{w} seems to be a little bit more complicated. Thus, let us consider this case in detail. Recall that, when processing a node, we run an initial test whether the crest separator X separating the two (\mathcal{S}', φ') -components corresponding to w and to \tilde{w} is equal to MaxCrestSep . If now \tilde{w} has another unfinished child, this test guarantees that we have $X \neq \text{MaxCrestSep}(\tilde{w})$, and we correctly define $\text{MaxCrestSep}(w^*)$ to be the largest crest separator in $\{\text{MaxCrestSep}(\tilde{w}), \text{MaxCrestSep}(w)\}$. In the case, where \tilde{w} has no other unfinished child, all children of \tilde{w} , except for w , are already marked as finished, and $\text{MaxCrestSep}(w^*)$ is also correctly computed. \square

Lemma 3.3.31. *Given a triconnected ℓ -outerplanar embedded graph (G, φ) with $G = (V, E)$, a good mountain structure $(G, \varphi, G', \varphi', \downarrow, \mathcal{S}')$ can be computed in $O(\ell|V|)$ time. Within the same time bound, we can additionally compute the mountain connection tree of $(G, \varphi, G', \varphi', \downarrow, \mathcal{S}')$, all (\mathcal{S}', φ') -components, and for each such component C , the set T_C of top edges belonging to C as well as, for each top edge, the two (\mathcal{S}', φ') -components to which it belongs.*

Proof. We already have shown in the last lemma that during our algorithm we always maintain a mountain structure $(G, \varphi, G', \varphi', \downarrow, \mathcal{S}')$. We next want to show that it is a good mountain structure. Property (b) holds since at the beginning of our algorithm we replaced \mathcal{S} by \mathcal{S}^* .

For proving property (c), we want a node w of T to be marked as finished if and only if we can guarantee that the (\mathcal{S}', φ') -component C' corresponding to w contains at least one crest of (G', φ') —and since no component ever contains more than one crest, exactly one crest of (G', φ') . Note that our algorithm only marks a node w of T as finished if $\text{Crest}(w) = 1$ and that at the end of our algorithm we have no node with $\text{Crest}(w) = 0$. Consequently property (c) holds if our algorithm correctly updates the values $\text{Crest}(w)$ for all nodes w . Assume that, when processing a node w , we have already correctly computed the values

$\text{Crest}(w')$ for all children w' of w . In particular, since the children are already marked as finished, we must have $\text{Crest}(w') = 1$ for all children w' of w . Thus, it is easy to see that our algorithm correctly computes $\text{Crest}(w^*)$ after merging w with one of its neighbors to a new node w^* .

For proving property (a), let us consider two crests H_1 and H_2 of (G', φ') and vertices $s \in H_1$ and $t \in H_2$. We know from Lemma 3.3.15 that there is a crest separator $X \in \mathcal{S}$ with one of its top vertices being a vertex of lowest depth on an s - t -ridge. Hence, the top vertices of X are not part of a crest and $X \in \mathcal{S}^*$. Let C_1 and C_2 be the $(\mathcal{S}^*, \varphi')$ -components containing H_1 and H_2 . Then X must be one of the crest separators that goes weakly between C_1 and C_2 . Let w_1 and w_2 be the nodes of T such that C_1 corresponds to w_1 and C_2 corresponds to w_2 . If, during the processing of a node w on the path from w_1 to w_2 , we remove X going weakly between the two (\mathcal{S}', φ') -components corresponding to w and to a neighbor on the path from w_1 to w_2 , we have $\text{Crest}(w) = 0$ and therefore at least one other crest separator X' going weakly between the two (\mathcal{S}', φ') -components corresponding to w and the other neighbor of w on the path from w_1 to w_2 . Since X instead of X' was chosen for a removal, X' must have a height equal to $\text{height}(X)$ or less and, in the first case, additionally no more top vertices than X . Hence, by having X' instead of X in \mathcal{S}' , property (a) still holds after removing X .

Let us finally analyze the running time. As already remarked, the time for the preprocessing is $O(\ell|V|)$. Since, during the traversal of T , we do not explicitly update the (\mathcal{S}', φ') -components and merge nodes of T and since T has $O(|V|)$ nodes, we can process all nodes of T also in $O(\ell|V|)$ total time. \square

3.4 Separators in Planar Graphs

Lemma 3.4.1. *Let (T, B) be a tree decomposition for a graph $G = (V, E)$ of width k , and let $v_1, v_2 \in V$. If no bag of (T, B) contains both, v_1 and v_2 , there is a separator S of size at most k strongly disconnecting $\{v_1\}$ and $\{v_2\}$ in G .*

Proof. For each edge $\{w, w'\}$ of T with $B(w) \neq B(w')$, the set $B(w) \cap B(w')$ is a separator of size at most k . Let w_i , for $i \in \{1, 2\}$, be a node of T with $v_i \in B(w_i)$. Since $B(w_1) \neq B(w_2)$, one can find two consecutive nodes w and w' on the path from w_1 to w_2 such that $B(w) \neq B(w')$. The lemma follows by taking $S = (B(w) \cap B(w')) \setminus \{v_1, v_2\}$. \square

Lemma 3.4.2. *Let (G, φ) be an embedded graph of treewidth k , and let v_1 and v_2 be vertices of G with $\mathcal{N}_\varphi(v_2) - \mathcal{N}_\varphi(v_1) \geq k + 1$. Then, there is a separator of size at most k that strongly disconnects $\{v_1\}$ and $\{v_2\}$ in G .*

Proof. Let (T, B) be a tree decomposition of width k with a smallest number of bags containing both, v_1 as well as v_2 . If there is no such bag, then Lemma 3.4.1 guarantees the existence of a separator of size at most k strongly disconnecting $\{v_1\}$ and $\{v_2\}$. Hence, let us assume that there is at least one node w in T with its bag containing v_1 and v_2 . Since $|B(w)| \leq k + 1$, one number $i \in \{\mathcal{N}_\varphi(v_1) + 1, \mathcal{N}_\varphi(v_2) - 1\}$ exists with $B(w)$ containing no vertex of peeling number i . Of course, there is a separator Y consisting exclusively of a connected set of vertices of peeling number i with Y disconnecting $\{v_1\}$ and $\{v_2\}$. The nodes of T , whose bags contain at least one vertex of Y , induce a subtree T' of T because of property (TD2).

Let us root T such that w is a child of the root and such that the subtree T_w does not contain any node of T' . We then replace T_w by two copies T_1 and T_2 of T_w and connect T_1 and T_2 to the root of T . In addition, we define the bag of each node w' in T_1 to consist of those vertices of the bag $B(w')$ of (T, B) that are also contained in the connected component of $G[V \setminus Y]$ containing v_2 . The bag of each node w' in T_2 should contain the remaining vertices of $B(w')$. The replacement described above leads to a tree decomposition of width k with a lower number of bags containing v_1 as well as v_2 . Contradiction. \square

It should be no surprise that, if we are able to find a separator for two vertices v_1 and v_2 , then we can find a separator for two sets V_1 and V_2 , too. The details are described next.

Theorem 3.4.3. *Let (G, φ) be an embedded graph of treewidth $k > 1$ and let V_1 and V_2 be connected sets of vertices of G such that $\min_{v \in V_2} \mathcal{N}_\varphi(v) - \max_{v \in V_1} \mathcal{N}_\varphi(v) \geq k + 1$. Then, there exists a separator of size at most k that strongly disconnects V_1 and V_2 in G .*

Proof. We modify the graph $G = (V, E)$ such that we can apply Lemma 3.4.2. First of all, merge all vertices of peeling number smaller than $\max_{v \in V_1} \mathcal{N}_\varphi(v)$ to a vertex v_1 , and merge all vertices that belong to the same connected component as V_2 in the subgraph of G without any vertex of peeling number lower than $\min_{v \in V_2} \mathcal{N}_\varphi(v)$ to a vertex v_2 . Let $G' = (V', E')$ be the graph obtained. Note that the treewidth of G' is not larger than the treewidth of G by Lemma 1.4.10 since G' is a minor of G . Take an embedding φ' of G' with $\varphi'|_{G'[V' \setminus \{v_1, v_2\}]} = \varphi|_{G[V \setminus (V_1 \cup V_2)]}$ that embeds v_1 into the outer face of $\varphi|_{G[V \setminus (V_1 \cup V_2)]}$ and v_2 into the face of $\varphi|_{G[V \setminus (V_1 \cup V_2)]}$ containing the nodes of V_2 if we embed the nodes of V_2 with respect to φ . Next add two vertices v' and v'' as well as edges $\{v_1, v'\}$, $\{v', v''\}$, and $\{v'', v_1\}$ into the outer face of (G', φ') such that the area A having as boundary the triangle consisting of the vertices v_1, v', v'' and not containing the outer face is incident to all vertices and edges of $G'[V' \setminus \{v_1\}]$. Let (G'', φ'') be the embedded graph obtained.

By our construction the peeling numbers of all vertices part of both embedded graphs (G, φ) and (G'', φ'') differ by exactly $\max_{v \in V_1} \mathcal{N}_\varphi(v) - 1$. Moreover, v_1 has peeling number 1 in (G'', φ'') and v_2 has peeling number at least $k + 1$ in (G'', φ'') . Note that a tree decomposition for G' of width $k > 1$ can be extended to a tree decomposition (T, B) for G'' of the same width by simply connecting a new node w with $B(w) = \{v_1, v', v''\}$ by an edge to a node of T whose bag already contains v_1 . Thus G'' has also treewidth k and the last lemma guarantees us that there is a separator S of size k strongly disconnecting $\{v_1\}$ and $\{v_2\}$ in G'' . Then $S \setminus \{v', v''\}$ also strongly disconnects V_1 and V_2 in G . \square

3.5 Special Tree Decomposition for Mountains

For this section, assume that we are given an (\mathcal{S}, φ') -component C of a mountain structure $(G, \varphi, G', \varphi', \downarrow, \mathcal{S})$ and the extended (\mathcal{S}, φ') -component (C^+, φ_{C^+}) of C such that φ_{C^+} is an ℓ -outerplanar embedding for some $\ell \geq 2$. We denote by \mathcal{S}_C the set of crest separators with a top edge in C .

The goal of this section is to construct a tree decomposition for C^+ (and thus for C) of width $3\ell - 1$ that, for each $X \in \mathcal{S}_C$, has a bag different from

the bags chosen for the other crest separators containing all vertices of the essential border of X . We construct such a tree decomposition by modifying the algorithm of Bodlaender [15] for the construction of a tree decomposition of an ℓ -outerplanar graph. However, instead of C^+ we will construct a tree decomposition (T^*, B^*) for a graph C^* with some extra properties, but we choose C^* in such a way that a tree decomposition for C^* can be easily transformed into a tree decomposition for C^+ . In an intermediate step, we first construct an embedded graph $(\tilde{C}, \varphi_{\tilde{C}})$ that is obtained as follows:

For each crest separator $X = ((u_1, \dots, u_q), (v_1, \dots, v_q))$ of \mathcal{S}_C that either has a lowpoint u_p with $p > 2$ or no lowpoint, do the following²: If X has a lowpoint u_p and X does not enclose C , take $r = p - 1$, and otherwise $r = q$. Recall that there are virtual horizontal edges $\{u_i, x_i\}, \{x_i, y_i\}$ and $\{y_i, v_i\}$ in (C^+, φ_{C^+}) for all $i \in \{2, \dots, r\}$. More exactly, if X has two top vertices, we have edges $\{y_2, z_2\}, \{z_2, v_2\}$ instead of the edge $\{y_2, v_2\}$. We now add edges $\{x_i, y_{i+1}\}$ for all $i \in \{2, \dots, r-1\}$. Moreover, we add an edge $\{y_2, u_1\}$. Finally, if $u_1 \neq v_1$, we add an edge $\{z_2, v_1\}$, and, if X has a lowpoint u_p and does not enclose C , we add an edge $\{x_{p-1}, u_p\}$. These two latter edges are called *extra edges of X* . Note that because of the border edges of X being contained in C , there is a unique rooted combinatorial embedding $\varphi_{\tilde{C}}$ for the graph \tilde{C} obtained from the additions above. For a simpler notation, we call the vertices x_i and y_i ($i \in \{2, \dots, r\}$) defined above *down-connected* and *up-connected*, respectively. See for an example Fig. 3.5.1.

Lemma 3.5.1. $\mathcal{N}_{\varphi_{C^+}}(v) = \mathcal{N}_{\varphi_{\tilde{C}}}(v)$ for all vertices of C^+ , and the two embedded graphs (C^+, φ_{C^+}) and $(\tilde{C}, \varphi_{\tilde{C}})$ have only one and the same crest.

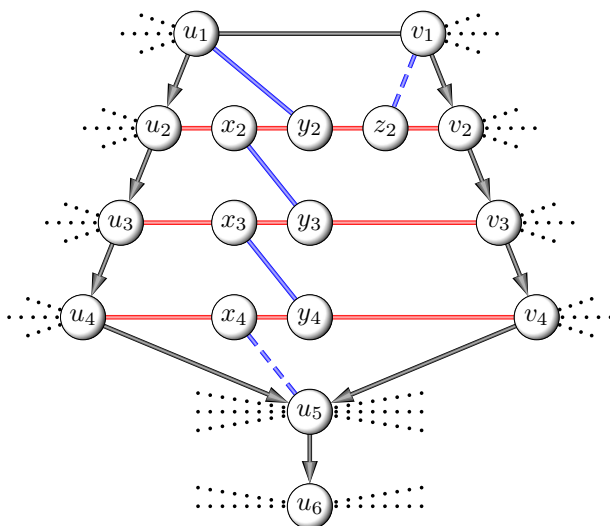


Figure 3.5.1: A crest separator X part of an extended component (C^+, φ^+) with its border edges and with the virtual horizontal edges. The blue edges are the new edges in \tilde{C} , and the dashed edges are the extra edges of X .

²The only crest separators, for which no further edges are added, are those having an essential border consisting only of a triangle.

Proof. We only introduced vertical edges. We already know from Lemma 3.3.1 that their introduction does not change any peeling number. Since (C^+, φ_{C^+}) is a mountain, (C^+, φ_{C^+}) has exactly one crest consisting of all vertices of largest peeling number, i.e., of peeling number ℓ . Again, because of the newly introduced edges connecting vertices of different peeling numbers, they cannot make a vertex part of a crest that was not belonging to any crest before the addition, and a maximal connected set of vertices in C^+ with peeling number ℓ is still connected after adding additional edges. Hence, after the addition of the new edges, we again have only one crest consisting of the vertices of peeling number ℓ . \square

It is much easier to work with a graph whose vertices have degree at most 3. Therefore in a second step, for each vertex v in \tilde{C} with $d \geq 4$ neighbors, we name its neighbors in clockwise order with u^1, u^2, \dots, u^d such that either $u^1 = v \downarrow$ holds or $\{u^1, v\}$ as well as $\{v, u^d\}$ are incident to the outer face of $(\tilde{C}, \varphi_{\tilde{C}})$. Moreover, we replace v by a path consisting of the vertices v^1, v^2, \dots, v^{d-2} in this order, each edge $\{v, u^i\}$ ($i \in \{2, \dots, d-1\}$) by the edge $\{v^{i-1}, u^i\}$, the edge $\{v, u^1\}$ by $\{v^1, u^1\}$, and the edge $\{v, u^d\}$ by $\{v^{d-2}, u^d\}$. A sketch of this construction is shown for one vertex in Fig. 3.5.2 and for the graph of Fig. 3.5.1 in Fig. 3.5.3, but ignore for the moment the difference between dashed and non-dashed edges.

The following observation shows that u^1 is well-defined even in the case of v having peeling number 1.

Observation 3.5.2. *No vertex of $(\tilde{C}, \varphi_{\tilde{C}})$ is an endpoint of more than two edges being incident to the outer face since this holds for (C^+, φ_{C^+}) because of C being a subgraph of a triconnected graph.*

We call the vertices v^1, \dots, v^{d-2} introduced for a vertex v with $\deg_{\tilde{C}}(v) = d \geq 4$ the *duplicates* of v . For technical reasons, we also call a vertex v part of \tilde{C} and C^* to be a *duplicate* of itself. Finally, for an edge $\{u, v\} \in \tilde{C}$, we call the edge $\{u', v'\}$ in C^* connecting duplicates u' of u and v' of v to be the *duplicate* of $\{u, v\}$. If $\{u, v\}$ is an extra, down, and virtual edge, the duplicates are also called *extra, down, and virtual edge*, respectively. Our choice of u^1 guarantees that the following three observations hold for the graph (C^*, φ^*) obtained after the second step.

Observation 3.5.3. $\mathcal{N}_{\varphi_{\tilde{C}}}(v) = \mathcal{N}_{\varphi_{C^*}}(v^1) = \dots = \mathcal{N}_{\varphi_{C^*}}(v^{d-2})$ holds for each vertex v of degree $d \geq 4$ in \tilde{C} that is replaced by vertices v^1, \dots, v^{d-2} in C^* , and $\mathcal{N}_{\varphi_{\tilde{C}}}(v) = \mathcal{N}_{\varphi_{C^*}}(v)$ holds for all v of degree at most 3 in \tilde{C} .

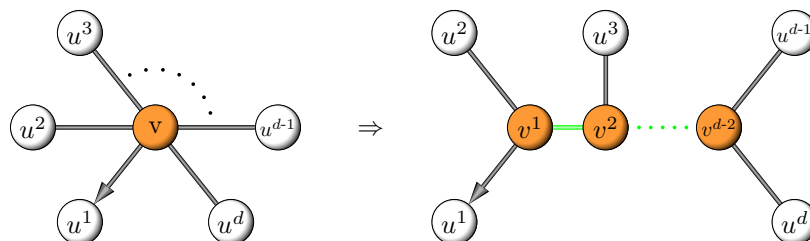


Figure 3.5.2: Splitting a vertex v with d neighbors into $d-2$ vertices v^1, \dots, v^{d-2} .

Observation 3.5.4. Since (C^*, φ^*) is a planar graph of maximal degree 3, no vertex of (C^*, φ^*) is incident to two or more vertices with a larger peeling number. Additionally, for each vertex of (C^*, φ^*) , the peeling numbers of each pair of its neighbors differ by at most 1.

Observation 3.5.5. Let $\{v, x\}$ be a virtual horizontal edge with v being a vertex of a crest separator $X = (L_1, L_2)$ and let v' be the vertex appearing immediately before v in L_1 or L_2 , respectively. Assume that v is split into two or more duplicates. We then can name the neighbors $\{u^1, \dots, u^d\}$ of v in clockwise order such that the following holds:

- If v has peeling number at least 2, u^1 is a duplicate of $v \downarrow$, and either $u^2 = x$ and u^3 a duplicate of v' or u^{d-1} is a duplicate of v' and $u^d = x$.
- If v has peeling number 1, either $u^1 = x$ and u^2 is a duplicate of v' or u^{d-1} is a duplicate of v' and $u^d = x$.

For an easier understanding of the last observation, consider the red dashed edges in Fig. 3.5.3—the meaning of the edges in other colors and the different types of edges is explained in the following.

Note that all virtual vertices have degree 3 in C' and are not split into two or more duplicates in C^* . Let (G_i, φ_i) be the embedded subgraph of C^* induced by the vertices of C^* with peeling number at least i such that $\varphi_i = \varphi^*|_{G_i}$. We next describe how to obtain a tree decomposition (T^*, B^*) for $C^* = (V^*, E^*)$

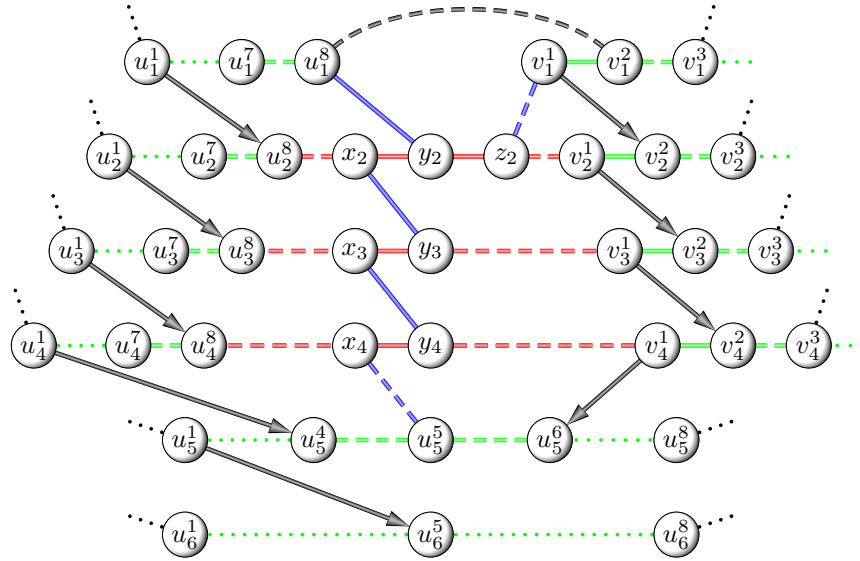


Figure 3.5.3: The figure shows the graph obtained from the graph of Fig. 3.5.1 after splitting all vertices of degree greater than 3. The superscripts of the names in the vertices should be considered as an example. Moreover, only the non-dashed edges are part of T after adding the edges of type (a), (b), and (c); more exactly, the black and blue non-dashed edges are of type (a), the green non-dashed edges are of type (b), and the red non-dashed edges are of type (c).

Note that some of the edges incident to $u_x \in \{u_1, \dots, u_6\}$ and $v_y \in \{v_1, \dots, v_4\}$ are incident to non-shown vertices u_x^i and v_y^j ($i, j \in \mathbb{N}$) and are thus not shown.

of small width. We begin with constructing a spanning tree $T = (W, F)$ for C^* as follows: Initially, we define T as the graph consisting of the vertices V^* and no edges. Next, we add edges of the following types into T .

- (a) all vertical edges of C^* , except for all extra edges of the crest separators.
- (b) all non-virtual horizontal edges $\{v, v'\}$ with v' being also incident to a virtual horizontal edge and adjacent to a vertex having a smaller peeling number than v' .
- (c) all virtual horizontal edges connecting two virtual vertices.

See again Fig. 3.5.3 for an example showing edges of all three types.

Observation 3.5.6. *Since T contains all edges of the types (a) and (b), by Obs. 3.5.5, each vertex being a duplicate of a vertex part of a crest separator $X \in \mathcal{S}_C$ and being incident to a virtual horizontal edge is connected in T to a duplicate of a vertex of X with a higher peeling number.*

For the last observation, note that u_2^8, u_3^8 , and u_4^8 as well as v_2^1, v_3^1 , and v_4^1 are part of a connected component in T containing a vertex with a higher peeling number in Fig. 3.5.3.

If afterwards T is not connected, we apply a postprocessing: We choose one connected component T_1 of T containing a vertex of largest peeling number, where we mean of course the peeling numbers of (C^*, φ^*) and not of T . Let i be the largest number for which there is a connected component of T different from T_1 containing vertices of peeling number i . Then choose a horizontal non-virtual edge e with its endpoints having peeling number i that connects T_1 with another connected component T_2 of T .

Lemma 3.5.7. *An edge e with the properties described above always exists.*

Proof. First note that, since $(\tilde{C}, \varphi_{\tilde{C}})$ is a mountain, the subgraph of \tilde{C} induced by the vertices of peeling number j is connected for each $j \in \{1, \dots, \ell\}$ and this also holds for the graph (C^*, φ_{C^*}) by construction. Note that T_1 also contains a vertex of peeling number i ; either i is the largest peeling number of all or the definition of i implies that T_1 contains all vertices of peeling number $i + 1$ and therefore—because of the addition of all edges of type (a)—at least one vertex of peeling number i . Note explicitly that this is true although the extra edges of the crest separators are missing. Since T_1 does not contain all vertices of peeling number i , there is at least one edge e in C^* with both endpoints having peeling number i connecting a vertex of T_1 to a vertex of another connected component T_2 of T . The lemma follows if we can show that e is not virtual.

Therefore, let us consider a virtual horizontal edge with both endpoints having peeling number i that is not contained in T and that connects a vertex u of T_1 and a vertex v of another connected component T_2 of T . We show that T_2 contains a vertex of peeling number $i + 1$, which contradicts our choice of i . Let us first assume that v is a virtual vertex. Because T contains all edges of type (c), either v or a neighbor of v is an up-connected virtual vertex in T_2 and hence T_2 contains a vertex of a higher peeling number than that of v . Finally assume that v is a non-virtual vertex. Since by our assumption v is incident to a virtual horizontal edge, v must be part of a crest separator, but cannot be a top vertex. By Obs. 3.5.6, T_2 contains a vertex of X that has peeling number $i + 1$. Contradiction. \square

Note that beside the edges of type (c) no virtual horizontal edges are added.

Observation 3.5.8. *In T , each down-connected virtual vertex v is only adjacent to one up-connected vertex, which has the same peeling number as v , and to possibly one vertex of a smaller peeling number.*

We next prove that T is indeed a tree.

Lemma 3.5.9. *T is a tree.*

Proof. T is connected by our construction rules and Lemma 3.5.7. To show that T contains no cycle, let us think of the edges introduced as being directed

- from a vertex with higher peeling number to a vertex with lower peeling number in case (a).
- to a vertex incident to a virtual horizontal edge and adjacent to a vertex of lower peeling number in case (b) (possibly, choose one of two possibilities).
- from the up-connected vertex to the other vertex in case (c).

We next show that the indegree of each vertex in the directed version of T is bounded by one before the postprocessing. By Obs. 3.5.4, this is true after adding only edges of type (a). A vertex v with an incoming edge of type (b) can clearly have no further incoming edges of the types (a) - (c). Since this also holds for a vertex with an incoming edge of type (c) by Obs. 3.5.8, we have $\text{indeg}_T(v) \leq 1$ for all vertices of T after adding all edges of types (a) - (c) to T .

Note that each path in the directed version of T only visit vertices with non-increasing peeling numbers. The only remaining possibility for a cycle in T is that there is a cycle K consisting of vertices of the same peeling number. However, such a cycle cannot contain an edge e of type (b) since, in \tilde{C} , one of its endpoints, beside edge e , is incident only to a vertical edge e_1 and to one virtual horizontal edge e_2 , which is not included in T , i.e., e_1 and e_2 both cannot be part of K . Since it is not possible to build a cycle consisting exclusively of edges of type (c), there is no cycle in T before the postprocessing. Since afterwards we only add edges between vertices of different components, we do not create cycles at all. \square

A tree decomposition (T^*, B^*) for $C^* = (V^*, E^*)$ is now constructed as follows: The vertex set V^* of T^* consists of both, the vertex set V^* and the edge set F of T . The edge set F^* of T^* is obtained by introducing for each edge $e = \{u_1, u_2\} \in F$ two edges $\{u_1, e\}$ and $\{u_2, e\}$ into T^* .

For each edge $e = \{u, v\} \in E^* \setminus F$, let us call the cycle consisting of edge e and the unique path from u to v in T^* to be the *fundamental cycle* of e . We define B^* in two steps: For each vertex $v \in V^*$ and each edge $e = \{u_1, u_2\} \in F$, add first v into $B^*(v)$ and u_1 and u_2 into $B^*(e)$. Second, for each edge $\{u_1, u_2\} \in E^* \setminus F$, add one of $\{u_1, u_2\}$ into all bags of the vertices in T that are part of the fundamental cycle of $\{u_1, u_2\}$. More precisely, if exactly one of u_1 and u_2 is a virtual vertex, add the non-virtual vertex into the bags of the fundamental cycle. For proving that (T^*, B^*) is a tree decomposition of small width, we need the statement of the next lemma.

Lemma 3.5.10. *For $i \in \{1, \dots, \ell\}$, the subgraph T_i of T consisting of all vertices of peeling number at least i is a tree.*

Proof. Assume for a contradiction that the lemma does not hold. Let us choose i as small as possible such that T_i is no tree. By Lemma 3.5.9, we know that $i > 1$. Thus, there must be two vertices u' and u'' of peeling number i in T such that the u' - u'' -path P in T contains a vertex of peeling number $i - 1$. Moreover, since we have chosen i as small as possible, P does not contain a vertex of peeling number $i - 2$. Let u_1, u_2 , and u_3 be three consecutive vertices of P with u_1 and u_2 having peeling number i and $i - 1$, respectively. Thus, $\{u_1, u_2\}$ is an edge of type (a). Obs. 3.5.4 applied to u_2 implies that $\{u_2, u_3\}$ cannot be an edge of type (a). Since u_2 cannot be incident to a vertex with peeling number $i - 2$ (again Obs. 3.5.4), $\{u_2, u_3\}$ could be an edge of type (b) only if u_3 is additionally incident to a horizontal virtual edge e_1 with a non-virtual endpoint and to an edge e_2 with an endpoint of peeling number $i - 2$. Therefore, neither e_1 nor e_2 can be part of P — e_1 is even not part of T . Hence, $\{u_2, u_3\}$ cannot be an edge of type (b). If $\{u_2, u_3\}$ is an edge of type (c), u_2 must be an up-connected virtual vertex. Then, either u_3 is a down-connected vertex or a virtual vertex being neither up- nor down-connected. In the first case, we can conclude by Obs. 3.5.8 that, in T , u_3 is only adjacent to u_2 and to a vertex of peeling number $i - 2$. In the second case, u_3 is only incident to $\{u_2, u_3\}$ in T . Hence we obtain a contradiction in both cases.

All of the remaining edges added into T are added to T in decreasing order with respect to the peeling numbers of their endpoints. Since u_2 has peeling number $i - 1$, before adding $\{u_2, u_3\}$ in the post processing phase, all vertices with peeling number i must be already connected. In particular, u' and u'' are connected. Contradiction. \square

Lemma 3.5.11. (T^*, B^*) is a tree decomposition of width $3\ell - 1$ for C^* .

Proof. It is not hard to see that (T^*, B^*) has all properties of a tree decomposition for C^* . All we have to prove is that (T^*, B^*) has width $3\ell - 1$. For each edge $e \in F$ and for each vertex $w \in W$, let $\mu(e)$ and $\nu(w)$ be the number of fundamental cycles that contain e or w , respectively. The values $\mu(e)$ and $\nu(w)$ are also called the *load* of e and w , respectively. Define $\mu = \max\{\mu(e) \mid e \in F\}$ and $\nu = \max\{\nu(w) \mid w \in W\}$. It is easy to see that the width of (T^*, B^*) is bounded by $\max\{\mu + 1, \nu\}$.

As in the last lemma, let $T_i = (V_i, E_i)$ ($i \in \{1, \dots, \ell\}$) be the subgraph of T consisting of all vertices of peeling number at least i , and let F_i be the set of edges in $E^* \setminus F$ that connect two vertices of peeling number i . Let us consider the increase of the load of the edges of T while adding the edges F_i into T . By Lemma 3.5.10, each edge $e \in F_i$ is incident to only one inner face of the embedding of $(V_i, E_i \cup F_i)$ inherited from φ_{C^*} . Since each edge of T_i is incident to at most two of these inner faces and since each vertex of T_i is adjacent to at most 3 vertices— T is a spanning tree of C^* , whose vertices have a degree of at most 3—the edges of F_i can increase the load of an edge of T by at most 2 and the load of a vertex by at most 3. Since this is true for all $i \in \{1, \dots, \ell\}$, $\mu \leq 2\ell$ and $\nu \leq 3\ell$, i.e., (T^*, B^*) is a tree decomposition of width $\min\{2\ell + 1, 3\ell\}$. The bound can be lowered to $\min\{2\ell + 1, 3\ell - 1\}$ by the fact that a vertex of peeling number 1 is incident to at most two inner faces in the embedding of $(V_1, E_1 \cup F_1)$ inherited from φ_{C^*} . \square

The tree decomposition (T^*, B^*) of width $3\ell - 1$ can easily be transformed into a tree decomposition (T, B) of C^+ by replacing, for each vertex v , its

duplicates in the bags of (T^*, B^*) by v . Here we use the fact that the duplicates induce a connected component in C^* and, thus, (TD2) holds for (T, B) , too.

Lemma 3.5.12. *For each crest separator $X \in \mathcal{S}$, there is a node w_X in (T^*, B^*) different from the corresponding nodes chosen for the other crest separators such that $B^*(w_X)$ contains a duplicate for each vertex of the essential border of X .*

Proof. First we consider the case that $X = ((u_1, \dots, u_q), (v_1, \dots, v_q))$ is a crest separator that has a lowpoint u_p with $p = 2$. Then, the essential border of X consists exactly of the vertices u_1, v_1 , as well as $u_2 = v_2$, and these three vertices induce a clique in C^+ . Assume for a moment that there is no bag of (T^*, B^*) containing a duplicate for each of the three nodes. Then, (T, B) also has no node whose bag contains $\{u_1, u_2, v_1\}$; a contradiction to Lemma 1.3.9.

For the remaining case, let Y be the set of virtual vertices introduced for a crest separator X , and let v be the virtual vertex adjacent to a top-vertex u of X in T (i.e., $\{u, v\}$ is no extra edge of X). For each vertex $u' \neq u$ of the essential border of X , there is a virtual vertex in Y that is connected by a virtual edge to a duplicate of u' , but this virtual edge has not been added to T . Note that these virtual edges are the only edges between a vertex of Y and a vertex not in Y . Thus, the nodes of one component of $T[W - \{u\}]$ consists exactly of all vertices in Y . Since the non-virtual endpoints of the virtual edges are added into all bags of their fundamental cycle, we can conclude the following: The bag $B^*(w)$ of the node w in T^* introduced for the edge $\{u, v\}$ must consist beside duplicates of u and v exactly of a duplicate of all vertices of the essential border. We choose $w_X = w$. \square

Concerning the running time it is easy to see that the embedded graph (C^*, φ^*) as well as the trees T^* and T can be constructed from (C^+, φ^+) in $O(|V|)$ time—recall that V denotes the vertex set of C^+ .

For the construction of (T^*, B^*) , note that we have to add one endpoint of an edge $e = \{u, v\}$ not part of T into all bags of the fundamental cycle of e . This fundamental cycle can be easily computed by traversing two pointers starting from u and v upwards in T^* such that the two pointers always have the same depth in T . Since in each bag visited by one of the two pointers we add a vertex, the time to construct (T^*, B^*) is linear in the total size of all bags of (T^*, B^*) , i.e., the running time for the construction of (T^*, B^*) can be bounded by $O(\ell|V|)$. Finally, it is easy to see that the transformation from (T^*, B^*) to (T, B) can be done in $O(\ell|V|)$ time, too.

As a corollary, we obtain the main result of this section.

Corollary 3.5.13. *Let $C = (V, E)$ be an (\mathcal{S}, φ') -component of a mountain structure $(G, \varphi, G', \varphi', \downarrow, \mathcal{S})$ such that the extended (\mathcal{S}, φ') -component (C^+, φ_{C^+}) of C has an ℓ -outerplanar embedding for some $\ell \geq 2$. Given (C^+, φ^+) , one then can construct a tree decomposition of width $3\ell - 1$ for C in $O(\ell^2|V|)$ time such that, for each crest separator $X \in \mathcal{S}$ with a top edge in C , there is a bag B_X different from the bags chosen for the other top edges that contains all vertices of the essential border of X .*

Proof. Construct a tree decomposition for C^+ and remove all vertices and edges not contained in C . Since C^+ may consist of $O(\ell|V|)$ vertices the running time is $O(\ell^2|V|)$ instead of $O(\ell|V|)$. \square

As a byproduct, we can also conclude a version of the well-known result of Bodlaender [15]—by the ideas of Bodlaender, one can improve the running time to $O(\ell|V|)$.

Theorem 3.5.14. *Given a graph $G = (V, E)$ with an ℓ -outerplanar embedding φ , a tree decomposition for G of width at most $3\ell - 1$ can be found in $O(\ell^2|V|)$ time.*

Proof. In $O(\ell|V|)$ time, construct first a good mountain structure $\mathcal{M} = (G, \varphi, G', \varphi', \downarrow, \mathcal{S})$ for $G = (V, E)$, the mountain connection tree T for \mathcal{M} , and for each top edge, the two (\mathcal{S}, φ') -components to which it belongs (Lemma 3.3.31). For each (\mathcal{S}, φ') -component $C = (V, E)$, compute then a tree decomposition for C as described in this section. Note that all these tree decompositions can be computed in $O(\ell^2|V|)$ total time by the last corollary. Finally, connect the tree decompositions (T_1, B_1) and (T_2, B_2) of two adjacent (\mathcal{S}, φ') -components C_1 and C_2 in T as follows: Let X be the common crest separator of C_1 and C_2 , and let w'_X and w''_X be nodes of T_1 and T_2 , respectively, such that $B(w'_X)$ and $B(w''_X)$ contain the essential border of X . Add an edge $\{w'_X, w''_X\}$.

By Lemma 3.3.25, we obtain a tree whose bags contain all vertices and edges of G , i.e., (TD1) holds; Lemma 3.3.27 implies that (TD2) is satisfied. \square

3.6 Shortcuts

As part of our main algorithm, we have to find out for a crest separator X of a mountain structure, whether some $(\{X\}, \varphi)$ -areas A are passed by some short alleys. We define these short alleys later as so-called pseudo shortcuts. For an efficient computation of such pseudo shortcuts, we want to compute them by independently constructing short alleys in different components of the mountain structure and by concatenating them. One problem is that an alley may pass from one component to another component by passing through a face that belongs only partly to each of the two areas of a crest separator since it also belongs partly to the other of the two areas. To avoid this situation, we want to add border edges into our graph. However, we want to maintain the length of shortest alleys, and we thus have to apply special rules for these border edges by introducing some kind of thin edges. More precisely, we extend the definition of alleys. For a shorter notation, from now on we call an object being either a vertex or an edge a *vedge*.

A *Z-skipping A-alley* in a graph (G, φ) with A being an area of (G, φ) and Z being a set of edges is a list P of pairwise disjoint vedges with each vedge being either a vertex of G or an edge of Z contained in G such that the following holds: Let E' be the edges of Z contained in P , and let (G^*, φ^*) be the embedded graph obtained from (G, φ) by splitting each edge $e \in Z$ into two edges with a new splitting vertex v_e . More precisely, *splitting an edge $e = \{u_1, u_2\}$ of an embedded graph (G, φ) into two edges* with a new *splitting vertex v_e* means

- to modify G by adding a new vertex v_e as well as replacing e by the two new edges $\{u_1, v_e\}$ and $\{v_e, u_2\}$, and
- to modify φ to obtain a rooted combinatorial embedding for the new graph: If $\varphi((u'_0, u_1)) = (u_1, u_2)$ and $\varphi((u_1, u_2)) = (u_2, u'_3)$ as well as $\varphi((u'_3, u_2)) = (u_2, u_1)$ and $\varphi((u_2, u_1)) = (u_1, u'_0)$ hold, redefine $\varphi((u'_0, u_1)) = (u_1, v_e)$,

$$\varphi((u_1, v_e)) = (v_e, u_2) \text{ and } \varphi((v_e, u_2)) = (u_2, u'_3) \text{ as well as } \varphi((u'_3, u_2)) = (u_2, v_e), \varphi((u_2, v_e)) = (v_e, u_1) \text{ and } \varphi((v_e, u_1)) = (u_1, u'_0).$$

Take P^* to be the list obtained from P after replacing each edge e of P by the vertex v_e . Then, P is a Z -skipping A -alley if and only if P^* is an A -alley in (G^*, φ^*) .

Note that the vedges of a Z -skipping A -alley in a graph (G, φ) are only allowed to consist of vertices and edges of G , whereas for technical reasons, Z may contain edges not part of G . The *length* of a Z -skipping A -alley $P = (s_1, \dots, s_q)$ —also denoted by $|P|$ —is defined as the number of subsequent pairs (s_i, s_{i+1}) ($i \in \{1, \dots, q-1\}$) with s_{i+1} being a vertex. The *inner vedges* of P are the elements s_2, \dots, s_{q-1} . For a vedge s , we often have to refer to s if it is a vertex or to both endpoints of s if s is an edge. Thus, for simplicity, we define the endpoint of a vertex s to be s itself. Note that for the length of a Z -skipping A -alley P , we only count vertices and this is what we mean by considering some edges—i.e., the edges in Z —as thin.

As for usual alleys, we also want to define crossings for Z -skipping A -alleys, which are a little bit more difficult to define. Let $P_1 = (u_1, \dots, u_p)$ and $P_2 = (v_1, \dots, v_q)$ be two Z -skipping A -alleys in an embedded graph (G, φ) with $G = (V, E)$, and let E' be the edges of Z belonging to P_1 or P_2 . Then P_1 and P_2 cross if the following holds: Let (G^*, φ^*) be the embedded graph obtained from (G, φ) by splitting each edge $e = \{u_1, u_2\} \in E'$ into two edges with a new splitting vertex v_e . Take P_1^* and P_2^* to be the alleys in (G^*, φ^*) obtained from P_1 and P_2 after replacing each edge e of these alleys by the vertex v_e . Then P_1 and P_2 cross in (G, φ) if and only if P_1^* and P_2^* cross in (G^*, φ^*) . If the crossing of P_1^* and P_2^* is non-planar, the crossing of P_1 and P_2 is also called *non-planar*. Otherwise, the crossing is called *planar*.

Definition 3.6.1 (s_1 - s_2 -connecting Z -skipping A -shortcut). Let (G', φ') be an embedded supergraph of an embedded (G, φ) with the same vertex set V , let Z be a subset of the edges of G' with Z including all these fence edges of an area A in (G', φ') that are not an edge of G , and let s_i , for $i \in \{1, 2\}$, be a fence vedge of A not being an edge of G . Let (G'', φ'') be the graph obtained from (G, φ) by adding the edges in Z into G and embedding them in the same way as φ' . An s_1 - s_2 -connecting Z -skipping A -shortcut of $(G, \varphi, G', \varphi')$ is a Z -skipping A -alley P from s_1 to s_2 in (G'', φ'') such that its length is as short as possible.

We next define a special kind of a shortcut. We therefore extend our definition of a crest alley. Let $X = (L_1, L_2)$ be a crest separator, and let s_1 and s_2 be vedges belonging to the essential border of X such that, for some i, j with $\{i, j\} = \{1, 2\}$, the endpoints of s_1 belong to L_i and the endpoints of s_2 belong to L_j . Then, we define the s_1 - s_2 -connecting crest alley of X to be the s_1 - s_2 -connecting alley obtained from a shortest crest alley of X from an endpoint of s_1 to an endpoint of s_2 —this crest alley from a vertex to a vertex is already defined—by adding s_1 in front of the crest alley if s_1 is an edge, and adding s_2 at the end of the alley if s_2 is an edge. More precisely, in the special case, where one of s_1 and s_2 is the lowpoint of X and where the other edge is incident to the lowpoint, the s_1 - s_2 -connecting crest alley should be (s_1, s_2) , and if both, s_1 and s_2 , are edges incident to the lowpoint and if they have different peeling

numbers, we define the s_1 - s_2 -connecting crest alley to be (s_1, v, s_2) with v being the lowpoint of X . These extra modifications are necessary since a crest alley from the lowpoint of X to itself is not defined as an alley only existing of the lowpoint of X . See for an example of the next definition Fig. 3.6.1.

Definition 3.6.2 (s_1 - s_2 -connecting Z -skipping pseudo A -shortcut of X). Let $X = (L_1, L_2)$ be a crest separator in an embedded graph (G, φ) with $G = (V, E)$, and let (G', φ') be an embedded supergraph of (G, φ) with the same vertex set that contains all border edges of X . Take A as an $(\{X\}, \varphi')$ -area, Z as a set of edges not being contained in G , and take Z' as the set of border edges of X not contained in G and being on the boundary of A .³ Let $s_1, s_2 \notin E$ be vedges of the essential border of X such that the endpoints of s_1 are contained in L_i and the endpoints of s_2 are contained in L_{3-i} . An s_1 - s_2 -connecting $Z \cup Z'$ -skipping A -shortcut P of $(G, \varphi, G', \varphi')$ is then called an s_1 - s_2 -connecting Z -skipping pseudo A -shortcut of X (with respect to $(G, \varphi, G', \varphi')$) if P has a strictly shorter length than the s_1 - s_2 -connecting crest alley of X . For a simpler notation, the word Z -skipping may be omitted if Z is the empty-set. If s_1 and s_2 are not of interest, P is called a (Z -skipping) pseudo A -shortcut of X . If also A is not of interest, P is called a pseudo shortcut of X .

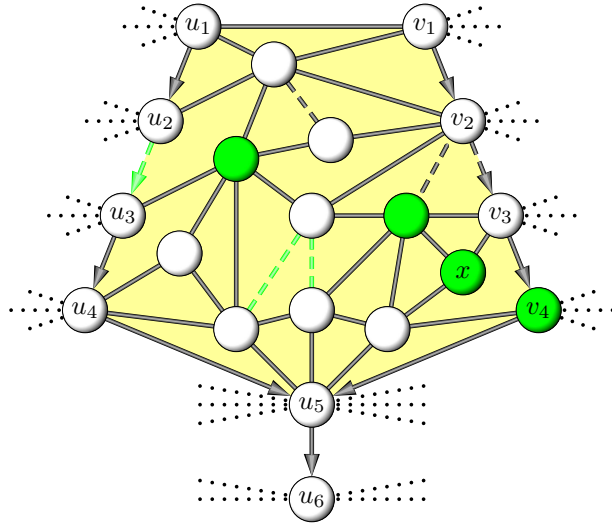


Figure 3.6.1: Let (G', φ') be the embedded graph sketched above, and let (G, φ) be the subgraph whose edges are shown non-dashed. For the crest separator $X = ((u_1, \dots, u_6)(v_1, \dots, v_6))$ with the yellow $(\{X\}, \varphi')$ -area A , there is a $\{u_2, u_3\}$ - v_4 -connecting pseudo A -shortcut P of X by the green vedges. This is the case, although the alley $(\{u_2, u_3\}, u_3, u_4, u_5, v_4)$ has also length $|P|$. Note that such a shortcut is not unique, e.g., we obtain another shortcut by replacing x with v_3 .

Note that although we consider $Z \cup Z'$ -skipping alleys, we refer to Z -skipping pseudo shortcuts since, given a crest separator X and an $(\{X\}, \varphi')$ -area, the set

³Not all border edges of X must be on the boundary of A since X may have a lowpoint.

Z' is uniquely defined and since the edges in Z' can only be used as endpoints of the pseudo shortcuts.

We later want to use a very special kind of pseudo shortcuts. For an area A and an alley P in an embedded graph (G, φ) , let us define a *maximum A -inner suballey* of P to be an A -suballey of P starting and ending in fence vedges of A that beside these two vedges does not consist of any further fence vedges of A . For a mountain structure $(G, \varphi, G', \varphi', \downarrow, \mathcal{S})$, we say that an A -pseudo shortcut P is *strong* if, for all crest separators $X \in \mathcal{S}$ and all $(\{X\}, \varphi')$ -areas A' covered by A , P has at most one maximum A' -inner suballey, and if so, this suballey is an A' -pseudo shortcut of X .

As shown by Lemma 3.6.4, the construction of a shortest alley visiting a vertex of a crest separator often allows us to follow the border edges of the crest separator instead of crossing the crest separator. Since statements about alleys consisting of vertices and edges are much more complicated than statements about alleys consisting of only vertices, let us first consider a simplified version.

Lemma 3.6.3. *Let $X = (L_1, L_2)$ be a crest separator of an embedded graph (G, φ) , and let (G', φ') be an embedded supergraph of (G, φ) with the same vertex set containing all border edges of X .*

- (a) *For each $i \in \{1, 2\}$, no alley with its endpoints v_1 and v_2 in L_i can have a shorter length than the v_1 - v_2 -connecting crest alley.*
- (b) *If X is of height ℓ and has $i \in \{1, 2\}$ top vertices, each pseudo shortcut of X with its endpoints being vertices can visit at most $i - 1$ vertices of peeling number ℓ and no vertex of peeling number larger than ℓ , where we mean the peeling numbers with respect to (G, φ) .*

Proof. Both (a) and (b) follow from the fact that the peeling numbers of two subsequent vertices of an alley can differ by at most 1. \square

Lemma 3.6.4. *Let $X = (L_1, L_2)$ be a crest separator of an embedded graph (G, φ) , and let (G', φ') be an embedded supergraph of (G, φ) with the same vertex set containing all border edges of X . Define Z to be a subset of the edges of G' without the edges of G .*

- (a) *Let s_1 as well as s_2 be a vedge part of the essential border of X such that all endpoints of s_1 and s_2 are in L_i ($i = 1$ or $i = 2$). Then, no s_1 - s_2 -connecting Z -skipping alley P in (G', φ') can have a shorter length than the s_1 - s_2 -connecting crest alley P^* .*

If X has a horizontal top edge not face-adjacent to a vertex of peeling number $h = \text{height}(X) - 1$, the same holds in the case, where one of $\{s_1, s_2\}$ is a horizontal top edge of X and only the endpoints of the other vedge are contained in L_i .

- (b) *If $X = (L_1, L_2)$ is a crest separator of height ℓ with $i \in \{1, 2\}$ top vertices, each Z -skipping pseudo shortcut P of X with respect to $(G, \varphi, G', \varphi')$ can visit at most $i - 1$ vertices of peeling number ℓ and no vertex of peeling number larger than ℓ , where we mean the peeling numbers with respect to (G, φ) .*

Proof. Let us define p_j ($j \in \{1, 2\}$) as the peeling number of an endpoint of s_j . More exactly, we take p_1 and p_2 in case (a) such that $r = |p_1 - p_2|$ is as low as possible, whereas p_1 and p_2 are both chosen as large as possible in case (b). For the latter case, let us additionally define P^* to be the s_1 - s_2 -connecting crest alley.

Assume in case (a) that $|P| < |P^*|$ and in case (b) that P visits at least i vertices of peeling number ℓ or larger. Note that $|P| < |P^*|$ holds in both cases since P is a pseudo shortcut in the latter case. If we remove from P and P^* all edges in Z , in (G, φ) , we obtain alleys Q and Q^* , respectively. Note that $|Q| < |Q^*|$ must hold.

Moreover, in case (a), the peeling numbers of the endpoints of Q^* differ by exactly r . The peeling numbers of Q differ by at least r . Here we also use the fact that in the case, where one of s_1 and s_2 is a horizontal top edge of X , this vedge is not face-adjacent to a vertex of peeling number $h = \text{height}(X) - 1$. Since the peeling numbers of two subsequent vertices of an alley can differ by at most 1, $|Q| \geq r = |Q^*|$, i.e., we obtain a contradiction in case (a).

In case (b), among all vertices of peeling number ℓ , choose v_1 as the first and v_2 as the last vertex in Q as well as v_1^* as the first and v_2^* as the last vertex in Q^* . In addition, let r_1 and r_2 be the difference between ℓ and the peeling number of the first and last vertex of Q^* , respectively. Then, the length of Q^* is exactly $r_1 + r_2 + i - 1$. The peeling numbers of the first vertex of Q and v_1 differ by at least r_1 and the peeling numbers of v_2 and the last vertex of Q differ by at least r_2 . Thus, Q has a length of at least $r_1 + r_2 + i - 1$, i.e., we obtain again a contradiction. \square

Corollary 3.6.5. *Take $(G, \varphi, G', \varphi', \downarrow, \mathcal{S})$ as a mountain structure, $X \in \mathcal{S}$ as a crest separator, and s_1 and s_2 as two vedges part of the border of X . If there is an s_1 - s_2 -connecting A -pseudo shortcut for an (X, φ') -area A , then there is also a strong s_1 - s_2 -connecting A -pseudo shortcut.*

Proof. Take P to be an s_1 - s_2 -connecting A -pseudo shortcut, and let A' be an (X', φ) -area for a crest separator $X' = (L_1, L_2)$ in \mathcal{S} such that A' is covered by A . If P contains a maximal A' -inner suballey not being an A' -pseudo shortcut, replace this suballey by a crest alley of X' (Lemma 3.6.4). If P has two maximal A' -inner suballeys P_1 and P_2 , there are two vedges s_1 and s_2 such that

- each is an endpoint of P_1 or P_2 ,
- for some $i \in \{1, 2\}$, all endpoints of s_1 and s_2 are in L_i , and
- s_1 , the inner vertices of one path in $\{P_1, P_2\}$, and s_2 appear in this order on P .

By Lemma 3.6.4.a, we can replace the subpath between s_1 and s_2 by the s_1 - s_2 -connecting crest alley of X without increasing the length. Thus, one path in $\{P_1, P_2\}$ is removed from P . \square

The next lemma later helps us to find some properties how pseudo shortcuts and crests interact.

Lemma 3.6.6. *Let \mathcal{S} be the set of crest separators constructed by the algorithm MS on the down closure $(G', \varphi', \downarrow)$ of a triconnected embedded graph (G, φ) , and let s and t be two vertices belonging to two different crests in (G', φ') . Let Z be*

a subset of the edges of G' without the edges of G , and let R be an s - t -ridge in (G', φ') of smallest depth having as few vertices of smallest peeling number as possible. Assume that there is a crest separator $X = (L_1, L_2) \in \mathcal{S}$ having the properties $P_R(X)$:

1. X crosses R in (G, φ) and the crossing is planar,
2. the height of X is equal to the depth of R .

If additionally the number of top vertices of X is as small as possible among all crest separators $X' \in \mathcal{S}$ with $P_R(X')$, then no Z -skipping pseudo shortcut P of X crosses R in $(G'[E \cup E_R \cup E_X \cup Z], \varphi'|_{G'[E \cup E_R \cup E_X \cup Z]})$, where E is the edge set of G , E_R the edge set of R , and E_X the set of border edges part of the essential border of X .

Proof. Take d to be the depth of R , i.e., d is also the height of X . Let us assume for a contradiction that there is a Z -skipping pseudo shortcut P of X that crosses R . Let us further assume for a moment that P contains an edge $e = \{u, v\}$ with both endpoints having peeling number at least d . Since P cannot start or end at a horizontal edge by the definition of a pseudo shortcut, e is an inner vedge of P . Recall that all edges of Z —i.e., also e —are added by the down closure. By our construction rules of a down closure an edge with both endpoints having peeling number d can only be added into a face F of (G, φ) for which the vertices of its boundary all have peeling number at least d . Hence there must be vertices of peeling number at least d in P appearing before and after edge e . Consequently, Lemma 3.6.4.b implies that P cannot have a shorter length than the crest alley with the same endpoints as P , since the crest alley consists of at most two vertices of peeling number d . Thus, P cannot contain e . Moreover, P cannot contain an edge with an endpoint larger than d since, in this case, P would also contain at least two vertices with peeling number d . Since P cannot contain an edge of R by the conclusions above, P and R must have at least one common vertex. Again by Lemma 3.6.4.b, we can conclude that P and R have exactly one common vertex v and that X must have two top vertices. Let s_1 and s_2 be the vedges part of P appearing immediately before v and after v , respectively. Note that these two vedges must appear on different sides of R . Lemma 3.6.4.b and the fact that P contains no edge with both endpoints having peeling number at least d implies that, for each $i \in \{1, 2\}$, at least one endpoint u_i of s_i has peeling number $d - 1$.

We next show that, in G' , v is adjacent to two vertices v_1 and v_2 on different sides of R both having peeling number $d - 1$. Take $v_1 = v \downarrow$. Choose $i \in \{1, 2\}$ such that u_i and v_1 are on different sides with respect to R . If $\{v, u_i\}$ is an edge in G' , take $v_2 = u_i$ and skip the rest of this paragraph. Otherwise, let F be an inner face of (G, φ) that is incident to v and s_i , i.e., that is incident to v and u_i —such a face must exist because v and u_i are face-adjacent in (G, φ) . Since (G', φ') contains all edges of R , there is an inner face F' of (G', φ') being covered by F that has v on its boundary. Moreover, we choose F' in such a way that u' and u'' either are part of R or leave R on the same side of R as u_i . Note that the boundary of F' consists of exactly three vertices v, u', u'' . By Lemma 3.3.6 at least one vertex on the boundary of F' has peeling number $d - 1$. Since v is not this vertex, we can choose v_2 to be one of the two vertices u' or u'' of peeling number $d - 1$. Note that no vertex of peeling number $d - 1$ can be part of R , and $\{v, v_2\}$ must leave R on a different side than $\{v, v_1\}$.

Hence we can define another crest separator $X' = (L'_1, L'_2)$, where L'_1 consists of the vertices of the down path starting in v (containing also v_1) and L'_2 consists of the vertex v and the vertices of the down path starting in the representant of v_2 around v . Since $P_R(X')$ holds and since X' has fewer top vertices than X has, we have again a contradiction. Thus, no Z -skipping pseudo shortcut P of X can cross R . \square

As a first step for the computation of pseudo shortcuts, we show next that one can efficiently compute shortest Z -skipping alleys in an embedded graph.

Lemma 3.6.7. *Let (G, φ) be an embedded graph with $G = (V^*, E)$, $Z \subseteq E$, and let A be an area in (G, φ) . For a given vedge $s_1 \in V^* \cup Z$, let $\delta(s_2)$ be the length of a Z -skipping A -alley from s_1 to s_2 for each vedge $s_2 \in V^* \cup Z$ that among all such Z -skipping A -alleys has shortest length. For all $q \in \mathbb{N}$, take S_q to be the set of all vedges $s_2 \in V^* \cup Z$ with $\delta(s_2) \leq q$. In $O(|S_{q+1}|)$ time, one then can compute the set S_q as well as the value $\delta(s_2)$ for each $s_2 \in S_q$. Moreover, within the same time one can construct a data structure that allows us to compute, for each $s_2 \in S_q$, the set of vertices part of a shortest Z -skipping A -alley P from s_1 to s_2 in the order in which they appear on P in $O(|P| + 1)$ time.*

Proof. A first problem we have to solve is that it is not so easy to consider Z -skipping A -alleys if these alleys contain edges. Therefore, we split each edge $e \in Z$ into two edges with a new splitting vertex v_e . In the embedded graph (G^*, φ^*) obtained, we search for a usual alley using the new vertices instead of the edges of Z . We denote the set of newly introduced vertices by V' and call each $v \in V'$ a *low-cost vertex*. Let m be the bijective function from $V^* \cup V'$ to $V^* \cup Z$ that maps a vertex $v \in V^* \cup V'$ to itself if $v \in V^*$ and to the edge e with $v_e = v$ otherwise. For a subset $V'' \subseteq V^* \cup V'$, $m(V'')$ is the union of $m(v)$ over all $v \in V''$. Define $s'_1 = m^{-1}(s_1)$. Our algorithm for computing the values $\delta(v)$ works as follows: The idea is to store with each vertex $v \in V^* \cup V'$ three values $d[v]$, $p[v]$, and $p'[v]$ with the following meaning: There is an A -alley from s'_1 to v in (G^*, φ^*) of length $d[v]$ whose last vertex before v is $p[v]$ and whose last vertex in V^* before v is $p'[v]$. More precisely, $p[v]$ and $p'[v]$ should be s'_1 if there is no vertex before v in $V^* \cup V'$ or no vertex in V^* , respectively. However, for an efficient implementation, we allow during our computation some of these values to be undefined. If we compare an undefined value with another value in the following description, we always consider the undefined value to be infinity (even an undefined value $p[v]$ or $p'[v]$). At the beginning of our algorithm we define the value $d[v]$, $p[v]$, and $p'[v]$ only for the vertex s'_1 with $d[s'_1] = 0$ and $p[s'_1] = p'[s'_1] = s'_1$. We also put s'_1 in an initially empty priority queue Q . All faces of (G^*, φ^*) are considered to be unmarked unless our algorithm explicitly mark them. In every step of our algorithm, we take a vertex u of Q whose current value $d[u]$ is of minimal size among all vertices in Q . We delete u from Q and, for each unmarked face F being incident to u and being contained in A , we mark F and relax over F from u . Relaxing over F from u means to relax from u over each vertex $v \in (V^* \cup V') \setminus \{u\}$ being incident to F , where *relaxing over v from u* means:

- to test whether $d[v] > d[u] + w(u, v)$ with $w(u, v) = 1$ if $u \in V^*$ and $w(u, v) = 0$ otherwise, and
- if so, to put v into Q and set $d[v] = d[u] + w(u, v)$, $p[v] = u$ as well as $p'[v] = u$ if $u \in V^*$ and $p'[v] = p'[u]$ otherwise.

We stop the algorithm after deleting the first vertex v from Q with $d[v] = q + 1$. At the end of the algorithm, for each v with $d[v] \leq q$, we output $d[v]$ for $m(v)$. We later show that $d[v] = \delta(m(v))$.

We next show that, if the algorithm sets the value $d[v]$ to a new value, there is indeed an s'_1 - v -connecting A -alley in (G^*, φ^*) of length $d[v]$ using $p[v]$ as last vertex in $V^* \cup V' \cup \{s'_1\}$ before v and using $p'[v]$ as last vertex in $V^* \cup \{s'_1\}$ before v . Let us define T to be the directed tree consisting of s'_1 and all vertices $v \in V^* \cup V'$ with $d[v] \neq \infty$. The vertices of T are connected by the directed edges $(p[v], v)$ for all $v \neq s'_1$. Assume that immediately before the relaxation over a vertex v from a vertex u (during the relaxation over a face F from u) the following is true: For each vertex v' in T , the vertices of the unique s'_1 - v' -path in T define an A -alley of length $d[v]$ in (G^*, φ^*) such that this path has vertices $p[v]$ and $p'[v]$ before v as last vertices in $V^* \cup V'$ and V^* , respectively, or has no such vertex if $p[v] = s'_1$ and $p'[v] = s'_1$, respectively. In particular, this means that T is a tree. If the relaxation now redefines $p[v] = u$, the only possibility to create a cycle in T is that v was an ancestor of u in T before the relaxation. However, this cannot be because we have $d[v] > d[u]$ before the relaxation. Therefore, T must still be a tree after the relaxation, and it is also easy to see that the properties of the function p and p' are maintained. The properties of T additionally imply that, for the output of the vertices of a Z -skipping A -alley of length $d[v]$ from s_1 to $m(v)$ in (G, φ) , we only have to print the reverse of the list $(m(u_1), \dots, m(u_{d[v]}))$ with $u_1 = v$, $u_{d[v]} = s'_1$ and $u_{i+1} = p'[u_i]$ for all $i \in \{1, \dots, d[v] - 1\}$. Note that having stored the array p' , we can easily construct the list $(m(u_1), \dots, m(u_{d[v]}))$ in $O(d[v])$ time.

For the correctness of our algorithm, it remains to show that, at the end of the algorithm, we have $d[m^{-1}(s)] = \delta(s)$ for all vedges $s \in V^* \cup Z$ with $\delta(s) \leq q$, and $d[m^{-1}(s)] \geq q + 1$ or $d[m^{-1}(s)]$ being undefined for all vedges s with $\delta(s) > q$. However, this proof is included as a special case of the proof of the next lemma, and it is thus not shown here.

We next want to analyze the running time for computing the values $\delta(s)$ for all $s \in S_q$. Since at each time of our algorithm, the values $d[v]$ for the vertices v in Q differ by at most 1, it is easy to implement the priority queue Q such that it supports the output of a vertex v with a minimal value $d[v]$ and the update of a value $d[v]$ after a relaxation in $O(1)$ time. Our algorithm only relaxes over faces that have as incident vertices only vertices $v \in V^* \cup V'$ with $\delta(m(v)) \leq q + 1$. The number of relaxations over a vertex $v \in m^{-1}(S_{q+1})$ (from vertices in $m^{-1}(S_q)$) is bounded by the number of faces in $(G^*[m^{-1}(S_{q+1})], \varphi|_{G^*[m^{-1}(S_{q+1})]})$ that are incident to v , i.e., is bounded by $O(\deg(v))$. Therefore, the total number of relaxations over a vertex is bounded by two times the number of edges, i.e., bounded by $O(|S_{q+1}|)$, and the whole algorithm runs in $O(|S_{q+1}|)$ time. \square

If we want to use the algorithm above to compute efficiently pseudo shortcuts for all crest separators, it is necessary to avoid visiting areas already considered during the construction of previous pseudo shortcuts for other crest separators and to use instead the already computed pseudo-shortcuts. We therefore introduce so-called forbidden areas.

Let (G', φ') be an embedded supergraph (G', φ') of an embedded graph (G, φ) with G and G' having the same vertex set V . Let A be an area in (G', φ') , and let \mathcal{A}^* be a set of subareas of A in (G', φ') such that no inner face

of (G', φ') is contained in more than one of the areas in \mathcal{A}^* . Moreover assume that the graph $G[A \dot{-} \mathcal{A}^*] = (V^*, E^*)$ is connected.⁴

Let Z be a subset of edges of G' without the edges of G , and let f be a partial function assigning a value in \mathbb{N}_0 to some pairs of vedges in $V^* \cup Z$ on the fence of at least one $A' \in \mathcal{A}^*$. Then, we define an (\mathcal{A}^*, f) -forbidden (Z -skipping) A -alley in (G, φ) as a list $P = (s_1, \dots, s_q)$ of vedges in $V \cup Z$ such that, for each pair of two consecutive vedges s_i and s_{i+1} in P , one of the following 3 cases as well as the so-called extended alley condition holds:

- (1) s_i and s_{i+1} are face-adjacent with a connection face in $A - \mathcal{A}^*$.
- (2) $\{s_i, s_{i+1}\}$ is a fence edge of at least one $A' \in \mathcal{A}^*$.
- (3) $f(s_i, s_{i+1})$ is defined.

The *extended alley condition* for P demands that, for each pair of consecutive vedges s_i and s_{i+1} of P with a connection face contained in $A - \mathcal{A}^*$, we can add the edge $\{s_i, s_{i+1}\}$ into a face of $A - \mathcal{A}^*$ and the graph obtained is still planar. Intuitively, (\mathcal{A}^*, f) -forbidden means that we want to forbid the alley to pass through the inner of any area $A' \in \mathcal{A}^*$. Instead, we want the alley to pass through a face in $A - \mathcal{A}^*$ in case (1), to follow the fence of at least one area $A' \in \mathcal{A}^*$ in case (2), or to skip an area $A' \in \mathcal{A}^*$ using the function f in case (3).

As for usual alleys we call the vedges s_2, \dots, s_{q-1} of an (\mathcal{A}^*, f) -forbidden (Z -skipping) A -alley $P = (s_1, \dots, s_q)$ the *inner vedges* of P . The weighted length $|P|$ of P is then defined as

$$\sum_{i=1}^{q-1} \begin{cases} 0 & \text{if case (1) or (2) holds and } s_{i+1} \text{ is an edge,} \\ 1 & \text{if case (1) or (2) holds and } s_{i+1} \text{ is a vertex,} \\ f(s_i, s_{i+1}) & \text{otherwise.} \end{cases}$$

For an efficient implementation of a partial function f , we store so-called *domain lists* that, for each vertex s' with at least one pair (s', s'') on which f is defined, consist of a list $\text{dom}_f(s')$ containing all vedges s with $f(s', s)$ being defined. Note that, for a (Z -skipping) A -alley being also an (\mathcal{A}^*, f) -forbidden (Z -skipping) A -alley, its usual length and its weighted length are equal, so that $|P|$ is well-defined.

It turns out that we sometimes need a special kind of (\mathcal{A}^*, f) -forbidden (Z -skipping) A -alleys. Intuitively, these special alleys are alleys that have the shortest length and have among those as much as possible consecutive vertices with a connection face in $A - \mathcal{A}^*$. Let us call a vertex s' an \mathcal{A}^* -idol of a vedge s if there is an area $A' \in \mathcal{A}^*$ such that $\{s, s'\}$ is a fence edge of A' or such that s is a fence edge of A' with endpoint s' . The idea is to prefer in a certain sense for each vedge its \mathcal{A}^* -idol—if it exists—as predecessor in the alleys. To be more exact, let us say that an s_1 - s_2 -connecting (\mathcal{A}^*, f) -forbidden (Z -skipping) A -alley P is *good* if it has the shortest weighted length among all s_1 - s_2 -connecting (\mathcal{A}^*, f) -forbidden (Z -skipping) A -alleys and if additionally one of the following three properties holds for each vedge s of P : Either s has no \mathcal{A}^* -idol, or the predecessor s' of s is an \mathcal{A}^* -idol of s , or the replacement of the s_1 - s' -connecting suballey of P by a shortest (\mathcal{A}^*, f) -forbidden (Z -skipping) A -alley from s_1 to an \mathcal{A}^* -idol of s increases the weighted length. See Fig. 3.6.2 for an example of the last definition.

⁴For the definition of $G[A \dot{-} \mathcal{A}^*]$ and of $A - \mathcal{A}^*$, see page 44.

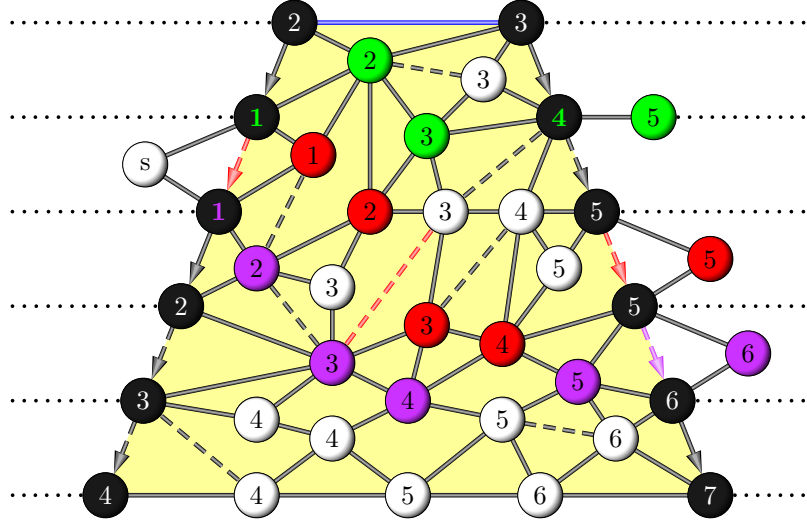


Figure 3.6.2: Let (G', φ') be the embedded graph sketched above and (G, φ) be the subgraph without the edges of Z that are drawn dashed. Take \mathcal{A}^* as the set containing the yellow area and $X = ((u_1, \dots, u_6), (v_1, \dots, v_6))$ as the crest separator shown by the black vertices and the blue top edge. The numbers inside the vertices denote the weighted distance of a Z -skipping alley from s . Note that the red alley is good, but the vertices v_2 and $\{v_4, v_5\}$ on the border of X have both an \mathcal{A}^* -idol that allows a rerouting of the green and the purple alleys using this \mathcal{A}^* -idol of v_1 and v_4 , respectively, without increasing the length.

For an s_1 - s_2 -connecting (\mathcal{A}^*, f) -forbidden (Z -skipping) A -alley P , we call a vertex s of P to be (\mathcal{A}^*, f) -essential if either $s = s_1$ or for the predecessor s' of s , the suballeys of P from s_1 to s' and from s_1 to s have different lengths. In particular, each vertex of P is essential. For the next lemma, recall that $G[A \dot{-} \mathcal{A}^*] = (V^*, E^*)$.

Lemma 3.6.8. *Take $\ell \in \mathbb{N}$, and let f be a function assigning $r \in \mathbb{N}$ pairs of vertices in $V^* \cup Z$ on the fence of at least one area $A' \in \mathcal{A}^*$ a value of at most 2ℓ . Assume that $f(s', s'') \geq 1$ for all vertices $s'' \in V^*$. Given the domain lists of f and a vertex $s_1 \in V^* \cup Z$, in $O((|V^*| + r) \log \ell)$ time, one then can compute a good s_1 - s -connecting (\mathcal{A}^*, f) -forbidden Z -skipping A -alley P_s for all vertices $s \in V^* \cup Z$. Within the same time one can construct a data structure that allows us to return for each vertex $s \in V^* \cup Z$ in constant time*

- the length of P_s and
- for each vertex s^+ on P_s , the predecessor of s^+ in P_s as well as the last (\mathcal{A}^*, f) -essential vertex before s^+ in P_s .

Proof. We introduce the set V' as the low-cost vertices, the function m , the graph G^* , and the vertex s'_1 as start vertex in the same way as in the proof of Lemma 3.6.7.

We also use arrays d , p , and p' , where initially set $d[s_1] = 0$, $p[s_1] = p'[s_1] = s_1$, and $d[v] = p[v] = p'[v] = \infty$ for all $v \in V^* \cup V'$ with $v \neq s_1$. In these arrays, for each vertex $v \in V^* \cup V'$ with $d[v] \neq \infty$, there is an s_1 - $m(v)$ -connecting

(\mathcal{A}^*, f) -forbidden Z -skipping A -alley $P^* = (s_1, \dots, s_r)$ ($r \in \mathbb{N}$) of length $d[v]$ in (G, φ) such that $p[v]$ is the last vertex part of $V^* \cup V'$ before $m(v)$ on P^* and such that $p'[v]$ is the last (\mathcal{A}^*, f) -essential vedge before $m(v)$ on P^* . Let f' be the function with $f'(u, v) = f(m(u), m(v))$ for each pair $u, v \in V^* \cup V'$ for which $f(m(u), m(v))$ is defined. Moreover, define $\text{dom}_{f'}(u) = m^{-1}(\text{dom}_f(m(u)))$ if $\text{dom}_f(m(u))$ is defined. We next run a modified version of the algorithm described in the proof of Lemma 3.6.7 with $q = |V|$; the details of the modifications are described now.

Whenever we delete a vertex $u \in V^* \cup V'$ from the priority queue Q , we relax over all faces not incident to an area of \mathcal{A}^* as usual with respect to the update of d and p . If during the relaxation over a face we relax over a vedge v from u and if we replace the current value $d[v]$ by $d[u] + w(u, v)$ with $w(u, v) \in \{0, 1\}$, we set $p'[v] = u$ if $d[p[u]] > d[p[p[u]]]$, and $p'[v] = p'[u]$ otherwise. We never relax over faces incident to u that are contained in an area of \mathcal{A}^* . Instead of this, we apply a so-called relaxation over the function f' from u and, if $u \in V^*$, additionally a so-called relaxation over the fence from u . *Relaxing over a function f' from u* means to test for each $v \in \text{dom}_{f'}(u)$ whether $d[v] > d[u] + f'(u, v)$. If so, we set $d[v] = d[u] + f'(u, v)$ and $p[v] = u$ as well as, if $d[p[u]] > d[p[p[u]]]$, $p'[v] = u$, or $p'[v] = p'[u]$ otherwise.

Relaxing over the fence from u means the following: For each vertex $v \in V^* \cup V'$ that has u as an \mathcal{A}^* -idol, we set $w(u, v) = 1$ if $v \in V^*$, or $w(u, v) = 0$ if $v \in V'$, and then we test whether $d[v] \geq d[u] + w(u, v)$. If so, we set $d[v] = d[u] + w(u, v)$, and $p[v] = p'[v] = u$ —note that we apply the changes even if $d[v]$ is equal to $d[u] + w(u, v)$, and this is what we meant with preferring an \mathcal{A}^* -idol of a vertex.

Whenever a relaxation over a function or over the fence decreases the value $d[v]$, we put v in our priority queue Q . Since the relaxation over function f' from u may set the value $d[v]$ to a value strictly larger than $d[u] + 1$, a vertex may now be added different times to Q . Thus, after finding a vertex in Q with smallest $d[v]$, we first test whether (another occurrence of) v was already deleted from Q in a previous step. If so, we delete the current occurrence of v and proceed with searching the next smallest vertex in Q .

For the same reason as described in the proof of Lemma 3.6.7, the graph T consisting of all vertices $v \in V^* \cup V'$ with $d[v] < \infty$ and of all edges $(p(v), v) \neq (s'_1, s'_1)$ maintains the property to be a tree when applying a relaxation over a face or over a function. If we possibly replace the value $p[v]$ by u during the relaxation over the fence from a vertex u , we must have $d[u'] < d[u]$ for all ancestors u' of u in T before the relaxation since u is a vertex. Thus, v cannot be an ancestor of u in T before the relaxation, and T maintains the property to be a tree. Since T is initially one vertex, i.e., a tree, T is always a tree.

If we define, for the vertices v in T , P_v to be the path from s_1 to v in T , then it is also easy to see that the number of vertices of P_v contained in V^* is always $d[v]$ and the last vertex before v on P_v is $p[v]$. Let P_v^* be the (\mathcal{A}^*, f) -forbidden Z -skipping alley in (G, φ) obtained from P_v by replacing each vertex u by $m(u)$. Note that $p'[v]$ is the last vertex v' before v with $m(v')$ being (\mathcal{A}^*, f) -essential on P_v^* .

To show that the computed alleys are of shortest length, let us define $\delta(s^*)$ as the length of an (\mathcal{A}^*, f) -forbidden Z -skipping A -alley of shortest weighted length from s_1 to s^* for each $s^* \in V^* \cup Z$. We next show that $d[v] \leq \delta(m(v))$ for all $v \in V^* \cup V'$. Note that this inequality implies that also $d[v] = \delta(m(v))$ holds since P_v^* has weighted length $d[v]$. Let v^* be a vedge with the smallest

value $\delta(v^*)$ for which, after deleting it from Q , we have $d[m^{-1}(v^*)] > \delta(v^*)$. Moreover, let $v = m^{-1}(v^*)$, and let P^* be an s_1 - v^* -connecting (\mathcal{A}^*, f) -forbidden Z -skipping A -alley in (G, φ) with weighted length $\delta(v^*)$. Take u^* as the vedge of P^* immediately before v and $u = m^{-1}(u^*)$. Since after processing v no value $d[v']$ of any vertex $v' \in V^* \cup V'$ is updated to a value strictly smaller than $d[v]$, $d[u]$ cannot be set to $\delta(u^*) \leq \delta(v^*) < d[v]$ after deleting v from Q . Hence, when processing v , u is already removed and $d[u] = \delta(u^*)$.

Let us first consider the case that $\{u^*, v^*\}$ is an edge on the boundary of at least one area $A' \in \mathcal{A}^*$. Then after the relaxation over the fence from u we have $d[v] \leq d[u] + 1 = \delta(u^*) + w(u, v)$, and thus $d[v]$ is set to $\delta(v^*)$.

Let us next consider the case that there is a connection face F of u^* and v^* contained in $A - \mathcal{A}^*$. If F is unmarked when relaxing over all unmarked faces incident to u , then after this relaxations $d[v] \leq d[u] + w(u, v) = \delta(u^*) + w(u, v)$ if $v \in V^*$, and $d[v] = d[u] = \delta(u^*)$ if $v \in V'$. In both cases $d[v]$ is set to $\delta(v^*)$. Otherwise, i.e., if F is marked, we already have relaxed over F when considering all unmarked faces of a vertex t removed before u from Q . Since afterwards no value is set to a value smaller than $d[t]$, we must have $d[t] \leq \delta(u^*)$ immediately after removing t from Q . Consequently, we again have $d[v] = \delta(v^*)$ after the relaxation over all faces incident to t .

We finally consider the case that the two cases above do not apply. We then must have $\delta(v^*) = \delta(u^*) + f(u^*, v^*)$, and hence $d[v] \leq d[u] + f'(u, v) = \delta(u^*) + f(u^*, v^*) = \delta(v^*)$ after relaxing over f' from u .

Let $v^* \in V^* \cup Z$. Then, for the output of the last (\mathcal{A}^*, f) -essential vedge before a vedge s^+ on the (\mathcal{A}^*, f) -forbidden Z -skipping A -alley P_v^* defined above with $v = m^{-1}(v^*)$, we simply return $p'[m^{-1}(s^+)]$. It only remains to show that the vertices of the path P_v^* defined above are the vertices of a good (\mathcal{A}^*, f) -forbidden Z -skipping A -alley. However, this must hold since—during the relaxation over the fence from some vertex $u \in V^*$ —we change $p[v]$ to u for each vertex $v \in V^* \cup V'$ with $m(u)$ being an \mathcal{A}^* -idol of $m(v)$ even if $d[v] = d[u] + w(u, v)$.

We next want to analyze the running time of the computation of the values $\delta(v)$. An update of a value $d[u]$ in Q may take up to $O(\log \ell)$ time since the priority queue Q may contain up to $O(\ell)$ different values because of the relaxation over f . Hence, it is easy to see that the total time needed for the relaxations over the fence from different vertices is bounded by $O(|E^*| \log \ell) = O(|V^*| \log \ell)$. The number of relaxations over vertices that are done during the relaxations over a face is bounded by $O(|V^*|)$ —similarly to the proof of Lemma 3.6.7. Since f is defined for exactly r pairs of vedges, the total running time of our algorithm is bounded by $O((|V^*| + r) \log \ell)$ time. \square

In the following, we apply the last lemma on a fixed good mountain structure $(G, \varphi, G', \varphi', \downarrow, \mathcal{S})$ with $G = (V, E)$. Keep in mind that G' is almost triangulated. We define Z to be the set of edges of G' without the edges of G . If we later construct alleys, the edges in Z prevent us from jumping from a non-boundary vertex in an (\mathcal{S}, φ') -component to a non-boundary vertex of another (\mathcal{S}, φ') -component. However, we have to show that the length of shortest alleys does not increase by the introduction of the extra edges in Z .

Lemma 3.6.9. *Let $(\tilde{G}, \tilde{\varphi})$ be an embedded graph obtained from (G, φ) by adding an edge e into a face of (G, φ) , and let Z' be a subset of the edges in G . For an area A of (G, φ) and two vedges s_1, s_2 in $V \cup Z'$, take P_G as Z' -skipping*

A -alley in (G, φ) and $P_{\tilde{G}}$ as $(Z' \cup \{e\})$ -skipping A -alley in $(\tilde{G}, \tilde{\varphi})$ both from s_1 to s_2 and both among all such alleys having shortest length in (G, φ) and $(\tilde{G}, \tilde{\varphi})$, respectively. Then, $|P_G| = |P_{\tilde{G}}|$.

Proof. For a Z' -skipping A -alley P of shortest length in (G, φ) , we can obtain a $Z' \cup \{e\}$ -skipping A -alley \tilde{P} in $(\tilde{G}, \tilde{\varphi})$ with the same length by possibly adding e into P , which does not increase the costs. Reversely, from a shortest Z' -skipping A -alley \tilde{P} in $(\tilde{G}, \tilde{\varphi})$ we obtain a $Z' \cup \{e\}$ -skipping A -alley \tilde{P} in (G, φ) of the same length by removing edge e if it exists in \tilde{P} . \square

Corollary 3.6.10. *Let A be an area of (G', φ') , let $\tilde{Z} \subseteq Z$ be a set containing all boundary edges of A , and let $\tilde{G} = (V, E \cup \tilde{Z})$. For $i \in \{1, 2\}$, take s_i as an edge of \tilde{Z} or as a vertex of G . If $P_{\tilde{G}}$ is a \tilde{Z} -skipping A -alley from s_1 to s_2 in $(\tilde{G}, \tilde{\varphi})$ and if $P_{G'}$ is a Z -skipping A -alley from s_1 to s_2 in (G', φ') such that both have shortest length, then we have $|P_{\tilde{G}}| = |P_{G'}|$.*

Proof. Iteratively apply Lemma 3.6.9. \square

Let A be an area in (G', φ') , and let \mathcal{A}^* be a set of subareas of A in (G', φ') such that no inner face of (G', φ') is contained in more than one of the areas in \mathcal{A}^* . Moreover, assume that the graph $G[A \dot{-} \mathcal{A}^*] = (V^*, E^*)$ is connected. Let B^* be the set of vedges not being an edge of E , but being part of the fence of at least one area $A' \in \mathcal{A}^*$. A function f is called (A, \mathcal{A}^*) -replacing if it is a partial function f from $B^* \times B^*$ to \mathbb{N}_0 such that the following holds:

- f is defined at least for each $(s', s'') \in B^* \times B^*$ for which there is an s' - s'' -connecting Z -skipping A' -alley P in (G', φ') for some $A' \in \mathcal{A}^*$ such that all alleys of the following type have a strictly larger length than $|P|$: Take the vertices of a path P from an endpoint of s' to an endpoint of s'' consisting completely of fence edges of an area $A'' \in \mathcal{A}^*$ in the order in which they appear on P and, if s' or s'' is an edge, add additionally s' in front of the alley and s'' at the end of the alley, respectively.
- if $f(s', s'')$ is defined, then $f(s', s'')$ is the length of a shortest s' - s'' -connecting Z -skipping A' -alley in (G', φ') for an $A' \in \mathcal{A}^*$ for which this length is minimal among all areas in \mathcal{A}^* .

Note that the definition implies that, if there is an s' - s'' -connecting Z -skipping A' -alley P in (G', φ') for some $A' \in \mathcal{A}^*$, then f is defined or there is an (\mathcal{A}^*, f) -forbidden Z -skipping A -alley P' from s' to s'' with $|P'| \leq |P|$ and the inner vedges of P' consist completely of vertices of the fence of one area in \mathcal{A}^* .

For the next lemma, let us fix A and \mathcal{A}^* with the properties above, and let $H = G'[A \dot{-} \mathcal{A}^*]$. Moreover, choose $\psi = \varphi'|_H$ and f to be an (A, \mathcal{A}^*) -replacing function.

Lemma 3.6.11. *Let s' and s'' be part of both $V \cup Z$ and H . If P is a shortest s' - s'' -connecting Z -skipping A -alley in (G', φ') , and if P^* is a shortest s' - s'' -connecting Z -skipping (\mathcal{A}^*, f) -forbidden A -alley in (H, ψ) , we have $|P| = |P^*|$.*

Proof. As above, let B^* be the vedges not in E that are part of the fence of at least one area $A' \in \mathcal{A}^*$. We first show that, for each Z -skipping A -alley $P_1 = (s_1, \dots, s_q)$ with $s_1 = s'$ and $s_q = s''$, we can construct a Z -skipping (\mathcal{A}^*, f) -forbidden A -alley P_2 in (H, ψ) from s' to s'' of length at most $|P_1|$. Let

$P'_1 = (s_i, s_{i+1}, \dots, s_j)$ be a maximal subsequence of consecutive vertices of P_1 such that, for each pair of vedges s_r and s_{r+1} with $r \in \{i, \dots, j-1\}$, there is neither a fence edge e of an area $A' \in \mathcal{A}^*$ equal to $\{s_r, s_{r+1}\}$ nor an inner face F in $A - \mathcal{A}^*$ incident to both, s_r and s_{r+1} . Then, there is a subsequence $P''_1 = (s_a, \dots, s_b)$ of P_1 with $s_a, s_b \in B^*$ such that P''_1 is a Z -skipping A' -alley for some $A' \in \mathcal{A}^*$. Assume for a moment that $f(s_a, s_b)$ is not defined. Then, there is an s_a - s_b -connecting (\mathcal{A}^*, f) -forbidden Z -skipping A -alley P''_2 with $|P''_2| \leq |P''_1|$ and with the inner vedges of P''_2 consisting completely of the vertices of the fence of an area in \mathcal{A}^* . We therefore replace P''_1 by P''_2 . Otherwise, i.e., if $f(s_a, s_b)$ is defined, we replace P''_1 by (s_a, s_b) . By the definition of f , $f(s_a, s_b) \leq |P''_1|$. Repeating this process we obtain a Z -skipping (\mathcal{A}^*, f) -forbidden A -alley P_2 in (H, ψ) from s' to s'' of weighted length at most $|P_1|$.

In the reverse direction, given a Z -skipping (\mathcal{A}^*, f) -forbidden A -alley $P_2 = (s_1, \dots, s_q)$ in (H, ψ) from $s_1 = s'$ to $s_q = s''$, we replace each pair (s_i, s_{i+1}) ($i \in \{1, \dots, q-1\}$) for which there is no connection face of s_i and s_{i+1} in A by a shortest s_i - s_{i+1} -connecting Z -skipping A' -alley for one $A' \in \mathcal{A}^*$ for which this length is minimal among all $A' \in \mathcal{A}^*$ to obtain a Z -skipping A -alley P_1 from s' to s'' of length at most $|P_2|$. \square

In the later parts of this chapter, we want to construct pseudo shortcuts only for a subgraph of G consisting of vertices with a high peeling number. For an $h \in \mathbb{N}$, we thus define $G(h)$ and $G'(h)$ to be in the following the subgraphs of G and G' induced by the set of all vertices with peeling number at least h , $\varphi(h) = \varphi|_{G(h)}$ and $\varphi'(h) = \varphi'|_{G'(h)}$. For a crest separator $X = ((u_1, \dots, u_q), (v_1, \dots, v_q))$ of height at least h and for $i \in \{1, \dots, q\}$ with u_i having peeling number h , we let $X(h) = ((u_1, \dots, u_i), (v_1, \dots, v_i))$. For a set \mathcal{S} of crest separators, we define $\mathcal{S}(h) = \{X(h) \mid X \in \mathcal{S} \text{ and } \text{height}(X) \geq h\}$ and, for an area A of (G', φ') , we also let $A(h)$ be the subset of A consisting of all faces of (G', φ') for which the vertices on the boundary all have peeling number at least h . For a crest separator $X \in \mathcal{S}$ of height at least h and for an $(\{X\}, \varphi')$ -area A , a *pseudo (A, h) -shortcut set for $X = (L_1, L_2)$* is a set consisting of exactly one s_1 - s_2 -connecting strong pseudo $A(h)$ -shortcut of X with respect to $(G(h), \varphi(h), G(h) \oplus X(h), \varphi'|_{G(h) \oplus X(h)})$ for each pair (s_1, s_2) for which an s_1 - s_2 -connecting pseudo $A(h)$ -shortcut exists. This also implies that, for some $j_1, j_2 \in \{1, 2\}$ with $j_1 \neq j_2$, each s_i ($i \in \{1, 2\}$) is either a vertex in L_{j_i} or a border edge of X not contained in G with both endpoints in L_{j_i} .

Lemma 3.6.12. *Let $C = (V_C, E_C)$ be an (\mathcal{S}, φ') -component, let $h \in \mathbb{N}$, let \mathcal{S}_C be the set of crest separators in \mathcal{S} whose top edges are part of C , let $X = (L_1, L_2)$ be a crest separator in \mathcal{S}_C , and let A_X be the C -inner $(\{X\}, \varphi')$ -area. Then, given for all $X' \in \mathcal{S}_C \setminus \{X\}$ a pseudo $(A_{X'}, h)$ -shortcut set $\mathcal{P}_{X'}$ of X' with $A_{X'}$ being the C -outer (X', φ') -area, we can compute a pseudo (A_X, h) -shortcut set \mathcal{P}_X of X in $O((\ell \log \ell)|V_C| + (\ell^3 \log \ell)|\mathcal{S}_C|)$ time with ℓ being the maximal height of a crest separator in \mathcal{S} .*

Proof. For the computation of the Z -skipping pseudo shortcuts, we want to use the algorithm described in the proof of Lemma 3.6.8 with A , \mathcal{A}^* , and f defined as follows: $A = A_X(h)$. \mathcal{A}^* is the union consisting, for each crest separator $X' \in \mathcal{S} \setminus \{X\}$ of the area $A_{X'}(h)$ (with $A_{X'}$ as defined in the lemma). The function f assigns to each pair s_1, s_2 of fence vedges of an area $A_{X'} \in \mathcal{A}^*$, for which there is an s_1 - s_2 -connecting Z -skipping $A_{X'}$ -pseudo shortcut P with respect to

$(G(h), \varphi(h), G'(h), \varphi'(h))$, the length of such a pseudo shortcut, i.e., $|P|$. Since by the assumptions of our lemma, we are given the $A_{X'}$ -pseudo shortcut sets for all $A_{X'} \in \mathcal{A}^*$ with respect to $(G(h), \varphi(h), G(h) \oplus X'(h), \varphi'|_{G(h) \oplus X'(h)})$ and since the length of these pseudo shortcuts is equal to the length of Z -skipping $A_{X'}$ -pseudo shortcuts with respect to $(G(h), \varphi(h), G'(h), \varphi'(h))$ connecting the same vedges (Corollary 3.6.9), it is trivial to initialize f . Observe that, if there is such a pseudo-shortcut, then there is exactly one crest separator X' with a top edge in C that has an s_1 - s_2 -connecting crest alley and there is exactly one area $A' \in \mathcal{A}^*$, namely $A' = A_{X'}$, having an s_1 - s_2 -connecting A' -pseudo shortcut.

Note that, for each pair of vedges s_1 and s_2 being part of the boundary of $A_{X'}$ with $X' \in \mathcal{S}_C$, $f(s_1, s_2)$ is never defined if all endpoints of these vedges are part of one list L_i with $i \in \{1, 2\}$ or if one of these vedges is a horizontal top edge. However in this case, because of Lemma 3.6.4.a (and additionally Lemma 3.3.29 if s_1 or s_2 is a horizontal top edge) an s_1 - s_2 -connecting Z -skipping $A_{X'}$ -alley does not have a strictly shorter length than the crest alley of X' , which is an s_1 - s_2 -connecting alley consisting only of fence vertices of $A_{X'}$. Thus, f is an (A, \mathcal{A}^*) -replacing function.

Applying Lemma 3.6.8 for a fence vedge s_1 of A , for each other vedge s_2 on the fence of A , but not belonging to the edges of E , we can obtain a good s_1 - s_2 -connecting Z -skipping (\mathcal{A}^*, f) -forbidden A -alley P' as well as its weighted length d . By Lemma 3.6.11, the length d is equal to the length of a shortest Z -skipping A -alley from s_1 to s_2 in $(G'(h), \varphi'(h))$, and by Corollary 3.6.10, equal to a shortest $(Z \cap \{s_1, s_2\})$ -skipping A -alley from s_1 to s_2 in $(G(h) \oplus X(h), \varphi'|_{G(h) \oplus X(h)})$. If and only if d is smaller than the length of the s_1 - s_2 -connecting crest alley of X , there is an s_1 - s_2 -connecting pseudo A -shortcut of X with respect to the tuple $(G(h), \varphi(h), G(h) \oplus X(h), \varphi'|_{G(h) \oplus X})$, and one such pseudo A -shortcut P being strong should be added to our pseudo (A_X, h) -shortcut set \mathcal{P}_X . Using the data structure of Lemma 3.6.8, i.e., the arrays p and p' , we can compute the sublist P^* of the (\mathcal{A}^*, f) -essential vedges of P' . Then P can be constructed from P^* as follows: For each vedge s of P^* , we test whether there is a connection face of $p[s]$ and s . If not, replace s by a $p[s]$ - s -connecting $A_{X'}$ -alley of length $f(p[s], s)$ for some crest separator $X' \in \mathcal{S}_C \setminus \{X\}$, namely, by a strong pseudo $A_{X'}(h)$ -shortcut of one of the precomputed pseudo $(A_{X'}, h)$ -shortcut sets. Moreover, if $p[s]$ then occurs twice, remove one occurrence. Note that either $p[s] = p'[s]$, or $p[s]$ is not (\mathcal{A}^*, f) -essential, i.e., $d[p[s]] = d[p'[s]]$ and since between $p[s]$ and $p'[s]$ are then only Z -skipping edges in P' , we can conclude that $p[s]$ and $p'[s]$ are face-adjacent in (G', φ') after removing all edges of $Z \setminus \{p[s], p'[s]\}$. Finally, remove all remaining inner edges of the alley obtained.

We next want to show that each pseudo-shortcut P computed above is indeed strong. Therefore, let us consider an area $A_{X'}$ for a crest separator $X' \in \mathcal{S}_C \setminus \{X\}$. If P has two maximum $A_{X'}$ -inner suballeys P_1 and P_2 , these suballeys are added because of consecutive vedges s'_i and s''_i of P^* with $f(s'_i, s''_i) = |P_i|$ ($i \in \{1, 2\}$). Take $X' = (L_1, L_2)$. Then, it is easy to see that we can find vedges s' and s'' both belonging to L_j for some $j \in \{1, 2\}$ such that s' , the vedges of one suballey in $\{P_1, P_2\}$, and s'' appear in this order on P —possibly some vertices are in between. Replacing the suballey P^+ of P' from s' to s'' by the s' - s'' -connecting crest alley \tilde{P} of X' cannot increase the length by Lemma 3.6.4.a. Then, P is no good s_1 - s_2 -connecting Z -skipping (\mathcal{A}^*, f) -forbidden A -alley since, for each vedge in \tilde{P} , its predecessor is also one of its idols, whereas this is not true

for each vedge of P^+ . Contradiction. Since, additionally, f is defined only for $A_{X'}$ -pseudo shortcuts with $X' \in \mathcal{S}_C$, P must be a strong pseudo A_X -shortcut.

Let us now analyze the running time of our algorithm. Since each pseudo $(A_{X'}, h)$ -shortcut set consists of at most ℓ^2 pseudo $A_{X'}(h)$ -shortcuts all having length at most $2 \cdot \text{height}(X') \leq 2\ell$, it is easy to construct the function f in $O(\sum_{X' \in \mathcal{S}_C \setminus \{X\}} |\mathcal{P}_{X'}| \ell) = O(\ell^3 |\mathcal{S}_C|)$ time. Since $G[A \dot{-} \mathcal{A}^*]$ consists of $|V_C| + O(\ell |\mathcal{S}_C|)$ vertices and since the number of all pseudo A' -shortcuts for all $A' \in \mathcal{A}^*$ sums up to $O(|\mathcal{S}_C| \ell^2)$, the total running time for the computation of the strong pseudo A_X -shortcuts of X from one fixed vedge $s_1 \in V^* \cup Z$ on the fence of A_X to all other vedges on the fence of A_X can be bounded by $O((\log \ell) |V_C| + (\ell^2 \log \ell) |\mathcal{S}_C|)$. For the computation of all strong pseudo A_X -shortcuts of X the running time has to be multiplied by ℓ .

We finally have to analyze the running time for the construction of the pseudo A -shortcuts P from a shortest Z -skipping (f, \mathcal{A}^*) -forbidden A -alley P' , or more precisely, from the data structure given by Lemma 3.6.8. Note that it is not necessary to compute P' explicitly. Recall that the data structure allows us to output the vedges of P^* in $O(|P^*| + 1)$ time. Since we have $f(s', s'') \geq 1$ for each pair of consecutive vedges on P^* , it is easy to construct P from P^* in $O(|P^*|) = O(|P|)$ time. \square

We call a pseudo A -shortcut of length ℓ an ℓ -long pseudo A -shortcut of a crest separator X . An A -pseudo shortcut is called h -high if it visits only vedges with their endpoints having peeling number at least h . For a mountain structure $(G, \varphi, G', \varphi', \downarrow, \mathcal{S})$ and $\ell \in \mathbb{N}$, we call a crest separator $X \in \mathcal{S}$ to be h -high ℓ -long pseudo shortcut free if, for each area A of the two $(\{X\}, \varphi')$ -areas, there is no h -high pseudo A -shortcut of length at most ℓ . An (\mathcal{S}, φ') -component C is called h -high ℓ -long pseudo shortcut free if, for each crest separator $X \in \mathcal{S}$ with a top edge in C , the C -inner (X, φ') -area A has no h -high ℓ -long pseudo A -shortcut for X and if C contains a vertex of peeling number $h - 1$.

Lemma 3.6.13. *Let $h \in \mathbb{N}$, and let $(G, \varphi, G', \varphi', \downarrow, \mathcal{S})$ be a good mountain structure with $G = (V, E)$ such that all crest separators of \mathcal{S} have height at most ℓ . In $O(|V| \ell^3 \log \ell)$ total time, we can compute a pseudo (A, h) -shortcut set for all $(\{X\}, \varphi')$ -areas A of all crest separators $X \in \mathcal{S}$. Within the same time, we can compute the set $\mathcal{S}' \subseteq \mathcal{S}$ consisting of all h -high ℓ -long pseudo shortcut free crest separators and the set of all h -high ℓ -long pseudo shortcut free (\mathcal{S}, φ) -components.*

Proof. Take $G = (V, E)$. Initially, compute the mountain connection graph T of $(G, \varphi, G', \varphi', \downarrow, \mathcal{S})$ (Lemma 3.3.26), and choose an arbitrary root of T . By the algorithm of Lemma 3.6.12, in a bottom-up traversal of T followed by a top-down traversal, we can compute all pseudo shortcut sets spending $O((\ell \log \ell) |V_C| + (\ell^3 \log \ell) |\mathcal{S}_C|)$ time per (\mathcal{S}, φ') -component C being a node C of T , where V_C and \mathcal{S}_C are defined as in Lemma 3.6.12. In detail, let X be a crest separator with its top edge being contained in an (\mathcal{S}, φ') -component C_1 and contained in the parent C_2 of C_1 in T . We then compute a pseudo (A, h) -shortcut set for the C_1 -inner $(\{X\}, \varphi)$ -area A of X during the bottom-up traversal, and an (A', h) -pseudo shortcut set for the C_1 -outer $(\{X\}, \varphi)$ -area A' of X during the top-down traversal. The running time over all (\mathcal{S}, φ') -components sums up to $O(|V| \ell^3 \log \ell)$. Moreover, given the pseudo (A, h) -shortcuts, it is

easy to determine all h -high ℓ -long pseudo shortcut free crest separators and (\mathcal{S}, φ) -components within the same time. \square

A *cyclic alley* is defined as a usual alley, except that the endpoints of a cyclic alley should be equal. We use many of the definitions introduced for alleys also for cyclic alleys as, for example, the definition of *inner vertices* or Z -skipping alleys. Let us explicitly mention that the *length* of a (Z -skipping) cyclic alley (s_1, \dots, s_q) is the number of consecutive pairs (s_i, s_{i+1}) of vedges of the alley with s_{i+1} being a vertex.

Let us define the *inner graph* of a cycle P in an embedded graph (G, φ) as an embedded graph (H, ζ) such that H is a maximal subgraph of G and such that exactly the vertices of P are incident to the outer face of $\zeta = \varphi|_H$. In addition, for a Z -skipping cyclic alley P in a triconnected embedded graph (G, φ) with $G = (V, E)$, let (G', φ') be the graph obtained by adding, for each pair of non-adjacent vertices u and v of P , an edge into a connection face of u and v in (G, φ) . Then, we define the *inner graph* of P in (G, φ) as the inner graph of P in (G', φ') . It is not hard to see that the inner graph of each cyclic alley in a triconnected embedded graph is unique. Moreover, the *strictly inner graph* is the maximal subgraph of the inner graph of P that contains no vertex on the boundary of P . We say that a cyclic alley P *encloses* a vertex set U if the vertices in U are part of the strictly inner graph of P . We also say in the following that a graph is *smaller* than another graph if it has a smaller number of vertices or the same number of vertices, but a smaller number of edges. Let X be a crest separator in a triconnected embedded graph (G, φ) , let φ' be an embedding of the supergraph of $G \oplus \{X\}$ with $\varphi'|_G = \varphi$, and let A and A' be the two $(\{X\}, \varphi')$ -areas. Define Z to be the set of border edges of X not already contained in G . Then, for a Z -skipping alley P in $(G \oplus \{X\}, \varphi'|_{G \oplus \{X\}})$, we define an *X -crossing sequence* to be a Z -skipping suballey (s, v_1, \dots, v_p, t) of P such that s and v_1 have only a connection face in one of the two $(\{X\}, \varphi')$ -areas, say area A , and v_p and t have no connection face in A , whereas for each pair v_i, v_{i+1} with $i \in \{1, \dots, p-1\}$, there are connection faces for v_i and v_{i+1} in both $(\{X\}, \varphi')$ -areas. This implies that the Z -skipping suballey of P from v_1 to v_p is a crest alley of X . Otherwise, for at least one $i \in \{1, \dots, p-1\}$, the vertices v_i and v_{i+1} are part of a crest alley P' of X with v_i and v_{i+1} being no consecutive vertices of P' . Thus, at least one vertex of the suballey of P' from v_i to v_{i+1} is disconnected by $\{v_i, v_{i+1}\}$ from at least one vertex with peeling number 1, which is a contradiction to the triconnectivity of G .

For a cyclic alley P in a triconnected embedded graph (G, φ) , for a crest separator X in (G, φ) , for the set Z of the border edges of X not contained in G , and for an embedding φ' of $G \oplus \{X\}$, let $P(G, X, \varphi')$ be a Z -skipping cyclic alley in $(G \oplus X, \varphi')$ such that

- P can be obtained from $P(G, X, \varphi')$ by the removal of all edges in Z , and
- P and $P(G, X, \varphi')$ have the same inner graph.

Intuitively, P and $P(G, X, \varphi')$ are the same alleys, but considered in different graphs G and $G \oplus \{X\}$. In an embedded graph, we call an alley *h -high* if all endpoints of its vedges have peeling number at least h .

For the next lemma and the next corollary, let us consider a situation, where X is a crest separator in a triconnected embedded graph (G, φ) with $G = (V, E)$

such that the top edge of X is incident to no vertex with a smaller peeling number. Take Z as the set of border edges of X not contained in G , φ' as an embedding of $G \oplus \{X\}$ and U as a connected subset of vertices of V incident to one $(\{X\}, \varphi')$ -area A , but not containing any vertex of X . In addition, let $P = (u_1, \dots, u_p)$ be an h -high cyclic alley enclosing U that is minimal with respect to the lexicographic ordering of the tuple (size, size of the inner graph).

Lemma 3.6.14. *$P(G, X, \varphi')$ has at most two X -crossing sequences.*

Proof. Let $X = (L_1, L_2)$. If the lemma does not hold, there are at least four X -crossing sequences and we can find an $i \in \{1, 2\}$ for which there are two X -crossing sequences $Q = (s, v_1, \dots, v_p, t)$ and $Q' = (s', v'_1, \dots, v'_{p'}, t')$ such that the endpoints of all vedges $v_1, \dots, v_p, v'_1, \dots, v'_{p'}$ are contained in L_i . We fix i and choose Q and Q' as described above with the additional constraint that the peeling numbers of the vedges v_1, \dots, v_p are as small as possible and that the peeling numbers of Q' are as small as possible among the crossing sequences with peeling numbers larger than that of the vedges v_1 and v_p . Let us choose $u \in \{v_1, v_p\}$ and $u' \in \{v'_1, v'_{p'}\}$ such that the difference of the peeling numbers of u and u' are as small as possible. Our choice of Q guarantees that the face enclosed by $P(G, X, \varphi')$ after adding edges between all subsequent non-adjacent vertices of $P(G, X, \varphi')$ contains the crest alley of X between u and u' . Let us define P_1 and P_2 to be the two h -high Z -skipping cyclic alleys that we obtain from $P(G, X, \varphi')$ by replacing the Z -skipping suballey from u to u' in the case of P_1 and from u' to u in the case of P_2 by the crest alley from u to u' and from u' to u , respectively. Since U is connected and does not contain any vertex of X , one of P_1 and P_2 must enclose U . Moreover, by Lemma 3.6.4.a the length of P_1 as well as the length of P_2 cannot be larger than the length of $P(G, X, \varphi')$. By our choice of Q , Q' , P_1 , and P_2 , it is easy to see that the inner graphs of P_1 and P_2 must be subgraphs of the inner graph of $P(G, X, \varphi')$. Let A be one of the two (X, φ') -areas. Since there is no connection face in A for one pair of $\{(s, v_1), (v_p, t)\}$ as well as for one pair of $\{(s', v'_1), (v'_{p'}, t')\}$ and since there is no connection face in the $(\{X\}, \varphi')$ -area $A' \neq A$ for each of the remaining pairs, the inner graphs of P_1 and P_2 must be real subgraphs of the inner graphs of $P(G, X, \varphi')$, i.e., contain less vertices. Hence, for one of P_1 and P_2 , the removal of the edges in Z would lead to an h -high Z -skipping cyclic alley enclosing U of length at most $|P| = |P(G, X, \varphi)|$ with a smaller inner graph than P . Contradiction. \square

Corollary 3.6.15. *If there is a pair u and v of two consecutive vedges of $P(G, X, \varphi')$ such that there is no connection face in $(G \oplus \{X\}, \varphi'|_{G \oplus \{X\}})$ contained in A , then $P(G, X, \varphi')$ contains an h -high pseudo A' -shortcut of X for the $(\{X\}, \varphi')$ -area $A' \neq A$ as a Z -skipping suballey.*

Proof. By our choice of P , the Z -skipping alley $P(G, X, \varphi')$ has at least one X -crossing sequence. Since the number of crossings must be even, we can conclude from Lemma 3.6.14 that $P(G, X, \varphi')$ has exactly two X -crossings sequences $Q = (s, v_1, \dots, v_p, t)$ and $Q' = (s', v'_1, \dots, v'_{p'}, t')$. Let us choose $u \in \{v_1, v_p\}$ and $u' \in \{v'_1, v'_{p'}\}$ such that the crest alley from u to u' has shortest length. Take P' to be the one of the Z -skipping suballeys from u to u' and from u' to u of $P(G, X, \varphi')$ that contains two subsequent vertices for which there is no connection face in A .

Assume for a moment that, for some $i \in \{1, 2\}$, u and u' both have all their endpoints in L_i . Since the top edge of X is incident to no vertex with a smaller peeling number, we know from Lemma 3.6.4.a that the length of P' is at least the length of the u - u' -connecting crest alley P'' of X . By replacing the Z-skipping suballey P' of P by P'' and removing all edges from the resulting alley we would obtain an h -high cyclic alley P^* enclosing U of length at most $|P|$ that has a smaller inner graph than P ; contradiction.

The only remaining case is that the endpoints of u and u' belong to different sides of the crest separator. Again, the replacement of P' by the crest alley from u to u' would lead to an h -high Z-skipping cyclic alley enclosing U with a smaller inner graph. The only reason for P' being part of P is that P' is a pseudo shortcut of X . \square

For the next two lemmata, let us consider a fixed path \tilde{P} in a mountain connection tree T of a good mountain structure $(G, \varphi, G', \varphi', \downarrow, \mathcal{S})$. To get an easier intuition, let us assume that the vertices of P are ordered from left to right. For some $r \in \mathbb{N}$, take C_0, \dots, C_r as the (\mathcal{S}, φ') -components being the nodes of P from left to right. Define X_i ($1 \leq i \leq r$) to be the crest separator whose top edge belongs to both C_{i-1} and C_i . The uniqueness of the crest separators X_1, \dots, X_r follows from Corollary 3.3.28. Let A_0 and A_r be the $(\{X_0\}, \varphi')$ -area covering C_0 and the $(\{X_r\}, \varphi')$ -area covering C_r , respectively. In addition, for $i \in \{1, \dots, r-1\}$, let A_i be the $(\{X_i, X_{i+1}\}, \varphi')$ -area, which covers C_i . Let H_i be the crest of (G, φ) contained in C_i ($i \in \{0, \dots, r\}$). For a simpler notation, we call the pseudo A -shortcuts for a crest separator X_i with A being the C_i -outer or the C_i -inner (X_i, φ') -area the *left* and the *right pseudo shortcuts* for X_i , respectively.

In the next lemma, we consider a very special kind of cyclic alleys. Let $X = ((u_1, \dots, u_q), (v_1, \dots, v_q))$ be a crest separator in (G, φ) , and let P be an s_1 - s_2 -connecting pseudo shortcut of X . Let P' be the crest alley of shortest length from an endpoint of s_2 to an endpoint of s_1 . The *cyclic alley* of (X, P) is then the cyclic alley obtained from concatenating the vertices of P and the vertices of P' in the order in which they appear on P and P' . Explicitly note that we do not exclude the case that s_1 and s_2 are equal to the lowpoint of X , and that in this case the s_1 - s_2 -connecting crest alley visits the top vertices of X .

Lemma 3.6.16. *For each left (right) pseudo shortcut P of a crest separator X_i with $i \in \{1, \dots, r\}$, the cyclic alley of (X_i, P) encloses the crest contained in C_{i-1} (in C_i).*

Proof. Let P' be the cyclic alley of (X_i, P) . Note that X_i is the only crest separator in \mathcal{S} strongly going between H_{i-1} and H_i . Due to this fact and since $(G, \varphi, G', \varphi', \downarrow, \mathcal{S})$ is a good mountain structure such that the border edges of X are contained in G' , we can conclude that property $P_R(X_i)$ of Lemma 3.6.6 holds for each ridge R connecting a vertex of H_{i-1} with a vertex of H_i . Hence we know that P cannot cross R . This shows that P' must enclose at least some vertices of H_{i-1} (of H_i).

Let h_{i-1} and h_i be the peeling numbers of the vertices in H_{i-1} and H_i , respectively. Note that X_i being the only crest separator strongly going between H_{i-1} and H_i implies that $\text{height}(X_i) < \min\{h_{i-1}, h_i\}$. By Lemma 3.6.4.b, P can visit at most one vertex of peeling number $\text{height}(X_i)$ and no vertex of a

peeling number larger than $\text{height}(X_i)$. Hence P' cannot disconnect two vertices of H_{i-1} (or of H_i) weakly and must enclose H_{i-1} (H_i). \square

Lemma 3.6.17. *Assume that X_1 has no lowpoint with a peeling number $\geq h$, that X_r has no h -high ℓ -long right pseudo shortcut, and that each X_i ($i \in \{2, \dots, r\}$) has an h -high ℓ -long left shortcut L_i being an A -alley for the C_0 -outer $(\{X_1\}, \varphi')$ -area A . Then X_1 cannot have an h -high ℓ -long right shortcut.*

Proof. Assume for a contradiction that the lemma does not hold. First note that no crest separator $X \in \{X_1, \dots, X_r\}$ can have a lowpoint with a peeling number $\geq h$. Otherwise X would enclose either all (\mathcal{S}, φ') -components left from X or all (\mathcal{S}, φ') -components right from X . This would imply that either X_1 or X_r has also lowpoint of peeling number at least h , where in the latter case this lowpoint is a right pseudo shortcut of X_r . Hence, we would obtain a contradiction to the conditions of the lemma. Let P_1 be an h -high ℓ -long right pseudo shortcut of X_1 of shortest length such that the inner graph I_1 of the cyclic alley P_1^* of (X_1, P_1) is as small as possible among all such alleys. Define $i \in \{1, \dots, r\}$ to be as small as possible such that P_1 is an $A_1 \cup \dots \cup A_i$ -alley. By the conditions of the lemma, we must have $i \leq r - 1$. Take P_2 as an h -high ℓ -long left pseudo shortcut of X_{i+1} with shortest length such that the inner graph I_2 of the cyclic alley P_2^* of (X_{i+1}, P_2) is as small as possible. Lemma 3.6.16 and the fact that a suballey L of P_1 is an h -high ℓ -long right pseudo shortcut of X_i imply that the strictly inner graphs of P_1^* and P_2^* both contain H_i . Consequently, the alleys P_1 and P_2 must cross at least two times.

Intuitively, we next interchange the suballeys of P_1 and P_2 between the first and last crossing of the two alleys. We then observe that the new right pseudo shortcut P_1' of X_1 does not contain any vertex of I_2 , except for some vertices of P_2 . This implies that H_i is not enclosed by the cyclic alley of (X_1, P_1') , which is a contradiction to Lemma 3.6.16.

Next we describe the interchange of the two suballeys in detail. Take $P_1 = (u_1, \dots, u_p)$ and $P_2 = (v_1, \dots, v_q)$. Let a be the smallest index such that (i) u_a is also part of P_2 or (ii) u_a is not contained in P_2 , but a vertex on the boundary of a face F_1 in $(G \oplus \{X_i, X_{i+1}\}, \varphi|_{G \oplus \{X_i, X_{i+1}\}})$ with four vertices $u_a, v_b, u_{a+1}, v_{b+1}$ appearing around F_1 in this series for some $b \in \{1, \dots, q - 1\}$. Note that the definition of a is unique since we have chosen P_1 and P_2 as the shortest h -high ℓ -long pseudo shortcuts. Set $u = u_a$ in case (i) and $u = u_{a+1}$ in case (ii). Similarly, let c be the largest index such that (i) u_c is also part of P_2 or (ii) u_c is not part of P_2 , but there exists a vertex part of P_2 such that this vertex is on the boundary of a face F_2 in $(G \oplus \{X_i, X_{i+1}\}, \varphi|_{G \oplus \{X_i, X_{i+1}\}})$ with four vertices $u_{c-1}, v_{d-1}, u_c, v_d$ appearing around F_2 in this series for some $d \in \{2, \dots, q\}$. Set $u' = u_c$ in case (i) and $u' = u_{c-1}$ in case (ii). We then replace the suballey of P_1 from u to u' by a suballey from a vertex v to a vertex v' of P_2 , where v and v' is chosen as follows: Take v from $\{v_b, v_{b+1}\}$ and v' from $\{v_{d-1}, v_d\}$ such that the distance between v and v' is as small as possible. Afterwards, if $u = u_a$, redefine $v = u_a$. Similarly, if $v' = u_c$, redefine $v' = u_c$.

Since P_1 and P_2 were chosen as h -high ℓ -long pseudo shortcuts of shortest length, it is clear that the suballeys of P_1 from u to u' and of P_2 from v to v' must have the same length. Since P_2^* weakly disconnects the vertices of I_2 from the remaining vertices, it is easy to see that the new h -high ℓ -long right pseudo shortcut P_1' of X_1 contains some vertices of P_2^* , but no other vertex of I_2 , i.e.,

also no vertex of H_i . Thus—as already observed—we obtain a contradiction to Lemma 3.6.16. \square

Lemma 3.6.18. *Assume that X_1 has no h -high ℓ -long left pseudo shortcut and that X_r has no h -high ℓ -long right pseudo shortcut. Then one of the crest separators X_1, \dots, X_r is h -high ℓ -long pseudo shortcut free.*

Proof. Note that no crest separator $X \in \{X_1, \dots, X_r\}$ can have a lowpoint with a peeling number $\geq h$. Otherwise, X would enclose either all (\mathcal{S}, φ') -components left from X or all (\mathcal{S}, φ') -components right from X . This would imply that either X_1 or X_r has also a lowpoint of peeling number at least h , which then is a left pseudo shortcut of X_1 and a right pseudo shortcut of X_r , respectively. Hence, we would obtain a contradiction to our assumptions.

W.l.o.g. all crest separators X_i ($i \in \{2, \dots, r\}$) have an h -high ℓ -long left shortcut. Otherwise, it suffices to prove the lemma for a shorter sequence of crest separators. Since X_2, \dots, X_r have a left pseudo shortcut, they also have a strong left pseudo shortcut (Corollary 3.6.5). Since X_1 has no left h -high ℓ -long pseudo shortcut and because of Lemma 3.6.4, all strong left pseudo shortcuts of X_2, \dots, X_r are A -alleys for the C_0 -outer $(\{X_1\}, \varphi')$ -area A . We apply Lemma 3.6.17 and conclude that X_1 cannot have an h -high ℓ -long right pseudo shortcut. Consequently, X_1 is h -high ℓ -long pseudo shortcut free. \square

Corollary 3.6.19. *Let $(G, \varphi, G', \varphi', \downarrow, \mathcal{S})$ be a good mountain structure, let \mathcal{H} be the set of all crests in (G, φ) that are contained in an h -high ℓ -long pseudo shortcut free (\mathcal{S}, φ') -component, and let \mathcal{S}' be the set of all h -high ℓ -long pseudo shortcut free crest separators of \mathcal{S} . Then, for each pair of crests $H', H'' \in \mathcal{H}$, there is a crest separator $X \in \mathcal{S}'$ strongly going between H' and H'' .*

Proof. We choose C_0, \dots, C_r as the (\mathcal{S}, φ') -components on the path in the mountain connection tree T of $(G, \varphi, G', \varphi', \downarrow, \mathcal{S})$ from the (\mathcal{S}, φ') -component containing H' to the (\mathcal{S}, φ') -component containing H'' . For $i \in \{1, \dots, r\}$, take X_i as the crest separator with a top edge part of both C_{i-1} and C_i .

Then by Lemma 3.6.18, there must be a crest separator $X \in \mathcal{S}'$ strongly going between H' and H'' . \square

It turns out that our algorithm for computing a tree decomposition often wants to disconnect vertices of high peeling numbers from vertices of low peeling numbers. In the following, we therefore designate sometimes a face-connected subset of the vertices of an embedded graph (G, φ) as the *coast* of (G, φ) if this subset consists of at least all vertices of peeling number 1. Take $G = (V, E)$. In this case, we define a *coast separator* for a set $U \subseteq V$ to be a set Y strongly disconnecting U from the coast. The coast separators are very useful for the construction of a tree decomposition of small width. One of the reasons for the introduction of pseudo shortcuts in this section is that they later help us to construct coast separators more efficiently. In the next two lemmata, we show that the vertices of a coast separator build a cyclic alley in certain cases.

Lemma 3.6.20. *Let V_1 and V_2 be the vertex sets of connected subgraphs of a triangulated triconnected embedded planar graph (G, φ) . Let Y be a vertex set strongly disconnecting V_1 and V_2 that among all such separators has minimal size. Then Y is a cycle in G and, for each vertex v of Y , at most two other vertices of Y are adjacent to v .*

Proof. Note that G must contain more than 3 vertices since otherwise either G is not triangulated or there is no separator in G that strongly disconnects two vertices of G .

For simplicity, we search for a separator in a minor G' of G obtained by merging the vertices of V_1 to a new vertex s and all vertices of V_2 to a new vertex t . Note that Y is a cycle in G if and only if it is a cycle in G' and that there is also a triangulated embedding of G' . By Lemma 2.2.4, there are $|Y|$ internally vertex-disjoint paths $P_1, \dots, P_{|Y|}$ in G' from s to t , i.e., each of them visits exactly one vertex of Y . Note also that, since G is triconnected, there are three vertex-disjoint paths from a vertex in V_1 to a vertex in V_2 . Thus, $|Y| \geq 3$.

We first show that each vertex u of Y is incident to exactly two other vertices in Y in G' and, hence, also in G . Assume for a moment that u is incident to at most one other vertex of Y . In this case, we can reroute the path P_i ($1 \leq i \leq |Y|$) having u as inner vertex by replacing u by some neighbors of u appearing clockwise around u such that the new path visits no vertex of Y —a contradiction to the fact that Y disconnects s and t . Let us now assume that u is incident to at least three other vertices v_1, v_2 , and v_3 of Y . Since no path P_i ($1 \leq i \leq |Y|$) can use an edge incident to u and another vertex of Y , we have a $K_{3,3}$ as a minor in G' : Each vertex of $\{s, t, u\}$ is connected by a path to each vertex of $\{v_1, v_2, v_3\}$ such that all paths are pairwise internally vertex disjoint. Contradiction.

Thus, the subgraph of G induced by the vertices of Y is a set of cycles C_1, \dots, C_q . Clearly one of these cycles must already disconnect V_1 and V_2 . Since Y was chosen as a separator of minimal size, we must have $q = 1$. \square

Lemma 3.6.21. *Let (G, φ) be a triconnected embedded graph $G = (V, E)$, and let $H_1 = (V_1, E_1)$ and $H_2 = (V_2, E_2)$ be connected subgraphs of G . If Y is a separator strongly disconnecting V_1 and V_2 such that Y among all separators strongly disconnecting V_1 and V_2 has as few vertices as possible, then there is a cyclic alley in (G, φ) whose vertices consist exactly of the vertices in Y . Moreover, for every vertex v in Y , at most two vertices in Y are face-adjacent to v .*

Proof. We first show that there is at least one cyclic alley with the properties described in the lemma. We construct a triangulated supergraph G^+ of (G, φ) by subsequently inserting edges into a face F with a boundary of more than three vertices as follows: If the boundary contains a vertex of Y , we add an edge with one endpoint being in Y . Otherwise, we add an arbitrary edge. After the insertions Y is still a separator of minimal size strongly separating V_1 and V_2 in G^+ . Y is a cycle in G^+ by Lemma 3.6.20, i.e., the vertices in Y induce a cyclic alley in (G, φ) .

It remains to show that, for every vertex v in Y , there are at most two vertices in Y being face-adjacent to v . Assume that this is not the case, and let (v_1, \dots, v_p) be an ordering of the vertices in Y corresponding to one cyclic alley. Choose v_i with $1 < i < p$ as a vertex that beside v_{i-1} and v_{i+1} is also face-adjacent to a vertex v_j with $j \notin \{i-1, i, i+1\}$. Since H_1 and H_2 are connected, one of the vertex sets $v_{\max\{i,j\}}, \dots, v_p, v_1, \dots, v_{\min\{i,j\}}$ or $v_{\min\{i,j\}}, \dots, v_{\max\{i,j\}}$ disconnects V_1 and V_2 strongly and is of smaller size than Y . Contradiction. \square

We say that a (Z -skipping) cyclic alley P represents a coast separator Y if the vertex set of P is exactly Y . Let (G, φ) be a triconnected embedded graph.

Take $G = (V, E)$. Then, the *inner graph* of a coast separator Y for a subset $U \subset V$ with Y being of minimal size among all coast separators for U is the inner graph of a cycle representing Y . By Lemma 3.6.21 the inner graph of such a coast separator is well-defined. A coast separator Y for a vertex set $U \subseteq V$ is called *absolutely-minimal* if it has the smallest size among all coast separators for U and if, among those, the inner graph of the coast separator has as few vertices as possible.

Lemma 3.6.22. *Let (G, φ) be a triconnected embedded graph with $G = (V, E)$ with a coast U_1 , let $\ell \in \mathbb{N}$, and let $U_2 \subset V$. Assume that $G[U_2]$ is connected. Then, given the set $N_G(U_1) \setminus U_1$, the graph $G[V \setminus U_1]$, and the embedding $\varphi|_{G[V \setminus U_1]}$, one can test in $O(\ell|V \setminus U_1|)$ time whether there is a separator of size at most ℓ strongly disconnecting U_1 and U_2 . If such a separator exists, within the same time bound, one can also construct an absolutely-minimal coast separator Y for U_2 as well as the inner graph $(I, \varphi|_I)$ of Y .*

Proof. Let $G^* = (V^*, E^*)$ be the graph obtained from G by merging the vertices in U_1 to a new vertex t and the vertices of U_2 to another new vertex s . G^* can be constructed in $O(|V \setminus U_1|)$ time from $G[V \setminus U_1]$ as follows: Connect the vertices part of $N_G(U_1) \setminus U_1$ to a new vertex t , and replace the vertices in U_2 by a new vertex s that is connected to all vertices in $N_G(U_2) \setminus U_2$. By Lemma 2.2.4, either there are $\ell + 1$ internally vertex-disjoint s - t -paths in G^* or there is a set of size at most ℓ disconnecting s and t in G^* . Moreover, either the paths or the separator can be found in $O(\ell|V^*|) = O(\ell|V \setminus U_1|)$ time using G^* . Remember that the algorithm for solving the problem above first constructs the vertex-disjoint-to-edge-disjoint version G^+ of G^* —see for a quick reminder Fig 2.2.5. For each vertex v of G^* , let (v', v'') be the directed edge of G^+ connecting the two vertices introduced for v . For some $r \in \{3, \dots, \ell + 1\}$, the algorithm first constructs a path P_i^{++} in the residual graph of G^+ and $P_1^{++}, \dots, P_{i-1}^{++}$ for each $1 \leq i \leq r$. Then, these paths are used to compute edge-disjoint s'' - t' -paths P_1^+, \dots, P_r^+ in G^+ . Subsequently, these paths are transformed into r internally vertex-disjoint s - t -paths P_1^*, \dots, P_r^* in G^* . More exactly, remember that a path P_i^* ($1 \leq i \leq r$) is obtained from P_i^+ by applying the backward vertex transformation.

Note explicitly that $r < \ell + 1$ if and only if the algorithm returns a separator Y of size r . Thus, if a separator Y is found, Y has minimal size. By our construction of G^* , Y is then a separator for U_1 and U_2 of minimal size in (G, φ) .

It remains to determine the inner graph of Y and to show that the inner graph of Y has as few vertices as possible among all coast separators of minimal size. Let S^+ be the subset of vertices reachable from s'' in the residual graph of G^+ and $P_1^{++}, \dots, P_r^{++}$, and let $S^* = \{u \mid \{u', u''\} \cap S^+ \neq \emptyset\}$. The vertices in $U_2 \cup S^* \setminus \{s\}$ are contained in the inner graph of all coast separators for U_2 of size r in (G, φ) . This can be shown as follows: Clearly, the vertices in U_2 must be contained in all inner graphs of coast separators for U_2 . For the remaining vertices, assume that there is a vertex $v \in S^* \setminus \{s\}$ for which there is a coast separator Y' for U_2 of size r in (G, φ) with the inner graph of Y' not containing v . Note that in G^+ there is a path P^+ edge-disjoint from P_1^+, \dots, P_r^+ leading from s'' to the vertex v' introduced for v . Our construction of G^+ implies that there is an s - v^* -path P^* in G^* internally vertex-disjoint to P_1^*, \dots, P_r^* . Since

all these paths must visit at least one vertex of Y' , we must have $|Y'| \geq r + 1$. Contradiction.

We next want to construct an absolutely-minimal coast separator Y for U_2 in (G, φ) . Let Y^+ consist of the union of the last vertex of P_i^+ contained in S^+ over all $i \in \{1, \dots, r\}$. Then, all paths from a vertex in S^+ to t' visit a vertex of Y^+ . We set $Y = \{u \mid \{u', u''\} \cap Y^+ \neq \emptyset\}$. By the construction of P_i^* from above, Y consists of the last vertex of P_i^* contained in S^* . Moreover, all paths from a vertex in S^* to t in G^* visit a vertex in Y . Hence, Y defines a coast separator for U_2 of size r in (G, φ) . Since $|Y| = r$, Y is of minimal size. We next show that Y is an absolutely-minimal coast separator for U_2 in (G, φ) . We already know that each inner graph of a coast separator of minimal size contains all vertices of $U_2 \cup S^* \setminus \{s\}$. It therefore suffices to show that only vertices in $U_2 \cup S^* \setminus \{s\}$ belong to the inner graph I of Y in (G, φ) . Otherwise, let v_0 be a vertex of I not contained in $U_2 \cup S^* \setminus \{s\}$. Then, v_0 cannot belong to the two connected components C_1 and C_2 of $G[V - Y]$ containing U_1 and U_2 , respectively. Consequently, $G[V - Y]$ contains another connected component C_3 containing v_0 . Since G is triconnected, there are three internally vertex-disjoint paths Q_1, Q_2, Q_3 from v_0 to a vertex of U_1 which implies that there are three internally vertex-disjoint paths from v_0 to three different vertices v_1, v_2 , and v_3 of Y that beside these vertices visit only vertices of C_3 . If we take the three paths in $\{P_1^*, \dots, P_k^*\}$ that visit v_1, v_2 , and v_3 , respectively, the three subpaths of these paths from s to the vertices in $\{v_1, v_2, v_3\}$, the three subpaths from the vertices in $\{v_1, v_2, v_3\}$ to t , and the three paths Q_1, Q_2 , and Q_3 define a $K_{3,3}$ in G^* . Note that G^* is a minor of the planar graph obtained from the planar graph G by adding edges into connecting-faces of the face-connected coast U_1 as long as U_1 is not connected. Thus, G^* must be still planar by Obs. 1.4.9, which is a contradiction to the fact that we found a $K_{3,3}$. Therefore our assumption that I contains vertices not part of $U_2 \cup S^* \setminus \{s\}$ must be wrong.

To sum up, we can compute an absolutely-minimal coast separator Y for U_2 as well as the inner graph I of Y in $O(\ell|V \setminus U_1|)$ time. Given the embedding $\varphi|_{G[V \setminus V_1]}$ of $G[V \setminus V_1]$, we can additionally construct the embedding of $\varphi|_I$ in the same time. \square

3.7 A Linear-Time Algorithm

Given $\alpha \in \mathbb{R}$ with $0 < \alpha \leq 1$, a triconnected graph $G = (V, E)$ of treewidth k , a $(k + \lfloor \alpha k \rfloor + 2)$ -outerplanar rooted combinatorial embedding φ of G , and a coast consisting of all vertices with peeling numbers at most $\lfloor \alpha k \rfloor + 1$, the algorithm below constructs a good mountain structure $(G, \varphi, G', \varphi', \downarrow, \mathcal{S})$ as well as a set \mathcal{P} of cyclic alleys with several good properties together with their inner graphs. Let \mathcal{H} be the set of crests in (G, φ) of height exactly $k + \lfloor \alpha k \rfloor + 2$.

(mountain structure, set) **function** $\text{decompose}_{k, \alpha}(\text{graph } G = (V, E))$

Step 1: In $O(k|V|)$ time, compute first a good mountain structure $\mathcal{M} = (G, \varphi, G', \varphi', \downarrow, \mathcal{S})$, second all (\mathcal{S}, φ') -components of \mathcal{M} , third, for each (\mathcal{S}, φ') -component C , the set T_C of top edges belonging to C , fourth the mountain connection tree $T = (W, F)$ of \mathcal{M} , and fifth, for each top edge, the two (\mathcal{S}, φ') -components to which it belongs (Lemma 3.3.31). Initialize $\mathcal{P} = \emptyset$.

Step 2: Define $h = \lfloor \alpha k \rfloor + 2$. Determine in $O(|V|k^3 \log k)$ time, for all crest separators $X \in \mathcal{S}$ and all $(\{X\}, \varphi')$ -areas A of X , a pseudo (A, h) -shortcut set of X , the set $\mathcal{S}' \subseteq \mathcal{S}$ consisting of all h -high k -long pseudo shortcut free crest separators, and the h -high k -long pseudo shortcut free (\mathcal{S}, φ) -components (Lemma 3.6.13). Moreover, compute the set $\mathcal{H}^+ \subseteq \mathcal{H}$ of the crests of (G', φ') contained in an h -high k -long pseudo shortcut free (\mathcal{S}, φ') -component in $O(|V|)$ time.

Step 3: Compute all (\mathcal{S}', φ') -components in $O(|V|)$ time.

Step 4: For each (\mathcal{S}', φ') -component $C = (V_C, E_C)$ obtained in Step 3, test in $O(|V_C|)$ time whether it contains a crest $H \in \mathcal{H}^+$, which then is unique by Corollary 3.6.19. If so, define the coast U_1 as the union over all vertices v with $\mathcal{N}_\varphi(v) \leq h - 1$ as well as over all vertices in $V \setminus V_C$. Then compute in $O(|V_C|)$ time the subset of vertices in V_C that have peeling number h and are adjacent to a vertex of peeling number $h - 1$ or that have peeling number at least h and are boundary vertices of C . We later show that this set is exactly the set $N_G(U_1) \setminus U_1$. Afterwards, we can easily construct the graph $G[V \setminus U_1]$ again in $O(|V_C|)$ time and then apply Lemma 3.6.22 with $U_2 = H$ to test in $O(k|V_C|)$ time whether there is a separator Y for U_1 and H of size at most k . If this is not the case, output that the treewidth is larger than k and stop the whole algorithm. Otherwise, compute the following in $O(k|V_C|)$ time: an absolutely-minimal coast separator Y for H , the inner graph I of Y with respect to (G, φ) , and the embedding $\varphi|_I$ (again Lemma 3.6.22).

Step 5: For each coast separator Y computed in Step 4, construct a cyclic alley P representing Y , i.e., with vertex set Y , and add P to \mathcal{P} . Store the inner graph I computed for Y as inner graph of P . Since each vertex of Y is face-adjacent to only two other vertices of Y —as shown in the proof of Lemma 3.6.21—this step runs also in $O(|V|)$ time.

Step 6: Determine the set \mathcal{H}' of all crests in \mathcal{H} that are enclosed by one of the cyclic alleys $P \in \mathcal{P}$. This can be done in $O(|V|)$ total time since different cyclic alleys in \mathcal{P} are constructed in different (\mathcal{S}', φ') -components.

Step 7: Iterate over each maximal connected subtree T^* of the mountain connection tree T that has only the (\mathcal{S}, φ') -components containing a crest $H \in \mathcal{H} \setminus \mathcal{H}'$ as nodes and that, for each h -high k -long pseudo shortcut free crest separator $X \in \mathcal{S}$, contains at most one (\mathcal{S}, φ') -component including the top edge of X . For each such subtree T^* of T , apply the Steps (a)-(e):

- (a) Initially, consider T^* as undirected tree without any marked nodes, and initialize a set W^* as the set of the (\mathcal{S}, φ') -components being nodes in T^* . As long as there is a node in W^* with exactly one unmarked neighbor in T^* , run (b). Afterwards proceed with (c).
- (b) Take an (\mathcal{S}, φ') -component C with exactly one unmarked neighbor C' in T^* . Delete C from W^* (but not from T^*). Next try to find a crest separator X and a shortcut L of X as follows:

In a time linear in the number of edges of C , test by the use of the precomputed pseudo shortcut sets if there is a crest separator with a top edge in C enclosing C . If so, take X as this crest separator and L as the lowpoint of X . Otherwise, take X as the crest separator, whose top edge belongs to C and to C' . Again, use the precomputed pseudo shortcut

sets, and determine in $O(1)$ time, whether X has a strong h -high k -long A -pseudo shortcut L for the C -inner $(\{X\}, \varphi')$ -area.

If and only if a shortcut L is found, mark C , and, for the crest H contained in C , define P_H to be the cyclic alley of (X, L) , and replace the edge $\{C, C'\}$ by a directed edge (C, C') .

- (c) For each remaining unmarked node C of T^* , do the following: Choose a crest separator $X \in \mathcal{S}$ that has a top edge belonging to C as well as a strong h -high k -long A -pseudo shortcut L for the C -inner $(\{X\}, A)$ -area. For the crest H contained in C , define P_H to be the cyclic alley of (X, L) .
- (d) Let \tilde{T} be the graph consisting of the nodes in T^* and the directed edges constructed in Step 7(b), which is an *intree*—i.e., a directed tree whose edges are all directed from a node to its parent—as shown in Lemma 3.7.2. Denote the root of \tilde{T} by R . Choose C to be the (\mathcal{S}, φ') -component of maximal depth being enclosed by a crest separator X with a top edge in C with X also enclosing R , or let $C = R$ if no such (\mathcal{S}, φ') -component exists. Note that C is uniquely defined since each (\mathcal{S}, φ') -component C' can be enclosed by at most one crest separator with a top edge in C' . For the crest $H \in \mathcal{H}$ contained in C , add P_H to \mathcal{P} . Moreover, compute the inner graph $I = (V_I, E_I)$ of P_H and all contained crests in $O(|V_I|)$ time. This is done by marking all vertices of P_H as seen and then by starting a *DFS*. Here we use the fact that Lemma 3.6.16 guarantees that H is enclosed by P_H so that the *DFS* indeed determines the correct inner graph. During the *DFS*, determine additionally all crests in $\mathcal{H} \setminus \mathcal{H}'$ contained in I . Initialize now \mathcal{T}' as the forest obtained from \tilde{T} by deleting all (\mathcal{S}, φ') -components containing a crest of $\mathcal{H} \setminus \mathcal{H}'$ contained in I . Note that R is one of the deleted (\mathcal{S}, φ') -components.
- (e) Until all nodes are deleted from \mathcal{T}' , repeat: Choose an (\mathcal{S}, φ') -component C that is a root of one intree in \mathcal{T}' , and add P_H to \mathcal{P} for the crest H contained in C . Moreover, compute the inner graph $I = (V_I, E_I)$ of P_H in $O(|V_I|)$ time (similar as in Step d). Afterwards, delete all (\mathcal{S}, φ') -components containing a crest of $\mathcal{H} \setminus \mathcal{H}'$ contained in I as nodes from \mathcal{T}' .

The running time of Step 7 is analyzed in Corollary 3.7.6.

Step 8: Return the mountain structure \mathcal{M} as well as the set \mathcal{P} of cyclic alleys and the inner graphs of these cyclic alleys.

Informally, we want to show in Corollary 3.7.6 that the algorithm above constructs a set of coast separators of size $3k + 2$ enclosing all highest crests such that different coast separators 'are part of' different (\mathcal{S}, φ') -components. We begin with some auxiliary lemmata.

Lemma 3.7.1. *Taking a graph G of treewidth at most k as input, the algorithm above constructs, for each crest in \mathcal{H}^+ , in Step 4 an absolutely-minimal coast separator. Moreover, the cyclic alleys added in Step 7 to \mathcal{P} , have length at most $3k + 2$.*

Proof. Let C be an (\mathcal{S}', φ') -component containing a crest $H \in \mathcal{H}^+$. Let U'_1 be the set of vertices of peeling number at most $h - 1$, where $h = \lfloor \alpha k \rfloor + 2$ as

in the algorithm. Then, by Theorem 3.4.3 there is a separator of size at most k strongly separating U'_1 and H in G . Let Y be an absolutely-minimal coast separator for H in (G, φ) if we let U'_1 instead of $U_1 = U'_1 \cup (V \setminus V_C)$ be the coast of G . Note explicitly that U'_1 and H indeed induce a connected component in G . By Lemma 3.6.21, we already know that the subgraph of G induced by the vertices of Y contains a cyclic alley P with vertex set Y . We next show that P is an alley in $(C, \varphi|_C)$. Assume for a contradiction that this is not true. Since the crest separators in \mathcal{S}' are h -high k -long pseudo shortcut free, they cannot have a lowpoint of peeling number at least h . Due to this, there is at least one crest separator $X \in \mathcal{S}'$ such that the alley $P(G, X, \varphi')$ has at least one X -crossing sequence. Because the number of crossing sequences must be even, Lemma 3.6.14 implies that $P(G, X, \varphi')$ has exactly two X -crossing sequences $Q = (s, v_1, \dots, v_p, t)$ and $Q' = (s', v'_1, \dots, v'_p, t')$. Let us choose $u \in \{v_1, v_p\}$ and $u' \in \{v'_1, v'_p\}$ such that the crest alley from u to u' has shortest length. Take P' to be the suballey from either u to u' or from u' to u such that P' contains two consecutive vertices for which there is no connection face in the C -inner $(\{X\}, \varphi')$ -area A . Hence, we know from Corollary 3.6.15 that P' is an h -high k -long pseudo A' -shortcut of X for the $(\{X\}, \varphi')$ -area $A' \neq A$. However, this cannot be the case since X as a crest separator in \mathcal{S}' does not have any h -high k -long pseudo shortcuts. Thus, $Y \subseteq V_C$, and Y therefore remains to be an absolutely-minimal coast separator of size at most k if we let now U_1 instead of U'_1 be coast of G . Hence the algorithm finds an absolutely-minimal coast separator in Step 4.

Note finally that the length of the cyclic alleys computed in Step 7 is bounded by the length of an h -high k -long pseudo shortcut plus the number of vertices v of X with $\mathcal{N}_\varphi(v) \geq h$, i.e., bounded by $k + 2(k + 1) = 3k + 2$. \square

Lemma 3.7.2. *For a fixed subtree T^* considered in Step 7, the forest \tilde{T} consists of exactly one intree.*

Proof. Note that the algorithm assigns to each marked node of T^* exactly one outgoing edge. \tilde{T} cannot contain any cycle—even after making all edges of \tilde{T} undirected—since T^* does not contain any cycle. Hence, \tilde{T} is indeed a tree immediately after its construction if we can show that after the marking process in Step 7 only one node of T^* remains unmarked. Assume that there is more than one unmarked node at the end of Step 7(b). For $i \in \{1, 2\}$, let w_i be the i th node that remains unmarked after deleting it from W^* . This means that w_i has only one unmarked neighbor w'_i and that the crest separator X_i with its top edge part of the two (\mathcal{S}, φ') -components C_{w_i} for w_i and $C_{w'_i}$ for w'_i does not have an h -high k -long pseudo shortcut being contained in the C_{w_i} -inner $(\{X_i\}, \varphi')$ -area. It is also not hard to see that, by our subsequent choices of unmarked nodes with exactly one unmarked neighbor, the maximal subtree of T^* containing w_i , but not w'_i , must consist of w_i and additionally only of marked nodes. Hence, for all i, j with $\{i, j\} = \{1, 2\}$, w_i is not contained in the C_{w_j} -inner $(\{X_j\}, \varphi')$ -area. Consequently, by Lemma 3.6.18, there must be an h -high k -long pseudo shortcut free crest separator in \mathcal{S} with a top edge belonging to two consecutive nodes of the path in T^* from C_{w_1} to C_{w_2} . This is a contradiction to the construction of the trees T^* in Step 7. \square

Let $\mathcal{M} = (G, \varphi, G', \varphi', \downarrow, \mathcal{S})$ be the good mountain structure, let T be a mountain connection tree for \mathcal{M} , and \mathcal{P} be the set of coast separators con-

constructed by the algorithm above for a graph G . In addition, let us define m as the mapping such that, for each cyclic alley $P \in \mathcal{P}$, $m(P)$ is the set consisting of all (\mathcal{S}, φ') -components containing a crest being enclosed by P .

Lemma 3.7.3. *For each $P \in \mathcal{P}$, the graph G_P consisting of the vertices and edges of all (\mathcal{S}, φ') -components in $m(P)$ contains the inner graph I_P of P as a subgraph.*

Proof. Let us first consider a cyclic alley P that is constructed in Step 7 as a cyclic alley of (X, L) for a crest separator $X \in \mathcal{S}$ with a lowpoint of peeling number at least h and a pseudo shortcut L that consists exactly of this lowpoint. The inner graph of P then consists exactly of the vertices of the (\mathcal{S}, φ') -components enclosed by X and these components are exactly the (\mathcal{S}, φ') -components contained in $m(P)$.

Let us next consider a cyclic alley $P \in \mathcal{P}$ not considered in the previous paragraph. If P was constructed in Step 5, let C_1 be the (\mathcal{S}, φ') -component containing the crest of \mathcal{H}^+ enclosed by P . Otherwise, if P is the cyclic alley of (X, L) for a crest separator $X \in \mathcal{S}$ and a strong h -high k -long pseudo shortcut L , let C_1 be the (\mathcal{S}, φ') -component that contains the top edge of X and is contained in the $(\{X\}, \varphi')$ -area for which L was constructed. Assume that there is some vertex v of I_P not part of G_P . This implies that there is an (\mathcal{S}, φ') -component C_2 containing v such that the crest H in C_2 is not enclosed by P . Let $X' \in \mathcal{S}$ be the crest separator with a top edge in C_2 such that C_1 is part of the C_2 -outer $(\{X'\}, \varphi)$ -area. Since, by our choice of C_2 the vertices of the C_2 -inner $(\{X'\}, \varphi)$ -area cannot be completely enclosed by P , $P(G, X', \varphi')$ contains a pair (u, v) of two consecutive vertices such that there is no connection face in $(G \oplus \{X'\}, \varphi'|_{G \oplus \{X'\}})$ contained in the C_2 -outer $(\{X'\}, \varphi')$ -area. Note that $P(G, X', \varphi')$ is either an absolutely-minimal coast separator without any vertices of peeling number lower than h or a cyclic alley containing a strong h -high k -long pseudo shortcut L of X' . Thus, $P(G, X', \varphi')$ must contain a pseudo shortcut in the C_1 -outer $(\{X'\}, \varphi')$ -area by Corollary 3.6.15 or by the definition of a strong pseudo shortcut. Lemma 3.6.16 then implies that P must enclose H . This contradiction shows that the vertex v defined above cannot exist. \square

Lemma 3.7.4. *For each $P \in \mathcal{P}$, $m(P)$ induces a connected subgraph of T .*

Proof. Let H_0 be the crest for which P is constructed, and let C_0 be the (\mathcal{S}, φ') -component containing H_0 . If there is an (\mathcal{S}, φ') -component C' with a crest H' that is enclosed by P , let C_0, \dots, C_r ($r \in \mathbb{N}$) in this order be the (\mathcal{S}, φ') -components of a C_0 - C' -path in T —in particular, $C_r = C'$. In addition, for $1 \leq i \leq r$, let H_i be the crest contained in C_i , and let X_i be the crest separator of \mathcal{S} with its top edge belonging to C_{i-1} and C_i .

Let us first consider the case that P was constructed in Step 7 as a cyclic alley of (X, L) for a crest separator $X \in \mathcal{S}$ with a lowpoint of peeling number at least h and a pseudo shortcut L that consists exactly of this lowpoint. Then, $m(P)$ consists exactly of the (\mathcal{S}, φ') -components enclosed by X and they clearly induce a connected subtree of T .

In all other cases, since P encloses beside H_0 also the crest H' , a suballey P' of $P(G, X_1, \varphi')$ must be a strong pseudo shortcut for the C_0 -outer $(\{X_1\}, \varphi')$ area of X_1 . Here we use the fact that P is either an absolutely-minimal coast separator or a cyclic alley for a pair (X, L) with X being a crest separator and

L being a strong pseudo shortcut. Let q be the minimal index in $\{1, \dots, r\}$ such that P' is completely contained in the C_0 -inner $(\{X_q\}, \varphi')$ -inner area A_q of X_q , or let $q = r + 1$ if no such separator exists. Then, for all crest separators X_i with $1 \leq i < q$, $P(G, X_i, \varphi')$ must contain an h -high k -long A_i -shortcut of X for the C_i -inner $(\{X_i\}, \varphi)$ -area A_i . By Lemma 3.6.16, H_i is then enclosed by P . If $q < r$, the only possibility for P to enclose H' and to be an A_q -alley is that X_{q+1} has a lowpoint. In this case, X_{q+1} as well as P encloses all (\mathcal{S}, φ') -components X_{q+1}, \dots, X_r . Hence, C_0, \dots, C_r are all contained in $m(P)$. We can conclude that $m(P)$ must be connected. \square

Lemma 3.7.5. $m(P) \cap m(P') = \emptyset$ for all pairs of different cyclic alleys $P, P' \in \mathcal{P}$.

Proof. Let us assume that there are two cyclic alleys P and P' in \mathcal{P} both enclosing a common crest.

We first consider the case that one of the two cyclic alleys, say P , is constructed in Step 5 for some crest $H \in \mathcal{H}^+$. Then P' must be constructed in Step 7 since different cyclic alleys constructed in Step 5 are contained in different (\mathcal{S}', φ') -components. Let H' be the crest for which P' was constructed and let C' be the (\mathcal{S}', φ') -component containing H' . Let C_0, \dots, C_r ($r \in \mathbb{N}$) in this order be the (\mathcal{S}, φ') -components of a C' - C -path in T —in particular, $C_0 = C'$ and $C_r = C$. In addition, for $1 \leq i \leq r$, let H_i be the crest contained in C_i , and let X_i be the crest separator of \mathcal{S} with its top edge belonging to C_{i-1} and C_i . Let us first consider the case that X_1 has a lowpoint. Then the crest alley constructed for H' must be a cyclic alley of (X, L) with X being a crest separator $X \in \mathcal{S}$ with a top edge in C' that has a lowpoint of peeling number at least h and with L being a pseudo shortcut that consists exactly of this lowpoint. If $X \neq X_1$, H' and H are in the same $(\{X\}, \varphi')$ -area, i.e., X beside H' must also enclose H which is a contradiction to the fact that H is contained in an h -high k -long pseudo shortcut free (\mathcal{S}, φ') -component. Otherwise, i.e., if $X = X_1$, L must be a pseudo shortcut for the C' -inner $(\{X_1\}, \varphi')$ -area. Hence P' is also contained in this area. If $P(G, X_i, \varphi')$ is not completely contained in the C' -outer $(\{X_1\}, \varphi')$ -area, Corollary 3.6.15 implies that a suballey of $P(G, X_i, \varphi')$ is a pseudo-shortcut for the C' -inner $(\{X_1\}, \varphi')$ -area and then encloses H' by Lemma 3.6.16. However, this could not be because of $H' \notin \mathcal{H}'$. We can conclude that P is contained in the C' -inner and P' in the C' -outer $(\{X_1\}, \varphi')$ -area and therefore they cannot both enclose the same crest.

It remains to consider the case that both, P and P' are constructed in Step 7. By the argumentation above, note that a cyclic alley \tilde{P} constructed in Step 7, $m(\tilde{P})$ does not contain an (\mathcal{S}, φ') -component with a crest in \mathcal{H}' . Moreover, the construction of each tree T^* in Step 7 implies that, for two (\mathcal{S}, φ') -components adjacent in T belonging to different subtrees T^* constructed in Step 7, the crest separator going between these (\mathcal{S}, φ') -components is h -high ℓ -long pseudo shortcut free. We hence can conclude together with Lemma 3.7.4 that all (\mathcal{S}, φ') -components of $m(P)$ are contained in one tree T^* . Thus, P and P' must be added into \mathcal{P} during the Steps 7(d) and 7(e) for the same tree T^* . We next want to show that different cyclic alleys added to \mathcal{P} during the traversal of \tilde{T} in the Steps 7(d) and 7(e) cannot contain the same crests. Since $m(P)$ is a connected subtree of T^* for each $P \in \mathcal{P}$ (Lemma 3.7.4), all we have to show is the following: Whenever we construct a cyclic alley P_H for a crest H with P_H being added to \mathcal{P} —in which case the (\mathcal{S}, φ') -component C containing H

must be the current root of \mathcal{T}' —then P_H is completely contained in the C -inner $(\{X\}, \varphi')$ -area of the crest separator X separating C from the parent of C in \tilde{T} . This is clear, if C is not enclosed by a crest separator: In this case P_H is chosen as the cyclic alley of (X, L) for an A -pseudo shortcut L with A being the C -inner $(\{X\}, \varphi')$ area. Otherwise, P is chosen as a crest separator X' enclosing C . If in this case, P is not contained in the C -inner $(\{X\}, \varphi')$ -area, X' must enclose all ancestors of C in \tilde{T} , in particular, X' encloses also the root of R of \tilde{T} . But in this case C must have been already deleted in Step 7(d). Contradiction. \square

Corollary 3.7.6. *Let $\alpha \in \mathbb{R}$ with $0 \leq \alpha \leq 1$. Assume that we are given a $(k + \lfloor \alpha k \rfloor + 2)$ -outerplanar rooted combinatorial embedding φ for a triconnected graph $G = (V, E)$ of treewidth k with the coast of G consisting of all vertices of peeling number at most $\lfloor \alpha k \rfloor + 1$. In $O(|V|k^4/\alpha \log k)$ time, the algorithm above then constructs a good mountain structure $\mathcal{M} = (G, \varphi, G', \varphi', \downarrow, \mathcal{S})$, a mountain connection-tree T for \mathcal{M} , a set of cyclic alleys \mathcal{P} and, for each $P \in \mathcal{P}$, the inner graph I of P and the corresponding embedding $\varphi|_I$ such that the following properties hold:*

- (i) *There is an injective mapping m from the cyclic alleys in \mathcal{P} to the (\mathcal{S}, φ') -components such that, for each $P \in \mathcal{P}$, $m(P)$ induces a connected subgraph of T and such that the union G_P of the (\mathcal{S}, φ') -components in $m(P)$ contains the inner graph I_P of P as a subgraph.*
- (ii) *For each crest H of height exactly $k + \lfloor \alpha k \rfloor + 2$ in (G, φ) , there is a cyclic alley $P \in \mathcal{P}$ with its vertex set Y being a coast separator for H of size at most $3k + 2$.*

Proof. We use the algorithm described at the beginning of this section and first analyze its running time. Except Step 7, we have already analyzed the running time of each of the steps in the description of the algorithm. The sum of the running time of all these steps is bounded by $O(|V|k^3 \log k)$. Let us next analyze the running time for Step 7. Note that the computation of all subtrees T^* of T in Step 7 can be done in $O(|W|) = O(|V|)$ total time, Step 7(b) and Step 7(c) take $O(k|V_C|)$ time for each (\mathcal{S}, φ') -component C considered, where V_C is the vertex set of C . Since we have $m(P) \cap m(P') = \emptyset$ for all pairs of cyclic alleys in \mathcal{P} with $P \neq P'$, the running time of Step 7(d) and Step 7(e) sums up to $O(k|V|)$. In summary, Step 7 runs in $O(k|V|)$ time; and the total running time is bounded by $O(|V|k^3 \log k)$.

We next show that the set \mathcal{P} of cyclic alleys computed by the algorithm has all desired properties. Property (i) follows from Lemmata 3.7.3-3.7.5.

Concerning property (ii), Lemma 3.7.1 and the definition of \mathcal{H}' imply that all crests $H \in \mathcal{H}'$ are enclosed by a cyclic alley in \mathcal{P} consisting of at most k vertices. Each remaining crest in $\mathcal{H} \setminus \mathcal{H}'$ is contained in an (\mathcal{S}, φ') -component being a node in one of the subtrees T^* considered in Step 7. Since an (\mathcal{S}, φ') -component is removed from T' in step Step 7(d) only if its crest is enclosed by a cyclic alley $P \in \mathcal{P}$, the coast separators added in Step 7 to \mathcal{P} must enclose all crests in $\mathcal{H} \setminus \mathcal{H}'$ and are of size at most $3k + 2$ (Lemma 3.7.1). \square

We now combine our results to compute, for a rational number α with $0 < \alpha \leq 1$ and for a connected embedded planar graph (H^*, ψ^*) with treewidth k ,

a tree decomposition of width $9k + 3\lfloor \alpha k \rfloor + 9$. More precisely, we solve a more general problem that we call the *inner tree decomposition problem* (ITD). In this problem, we are given a quintuple $(k, \alpha, (H^*, \psi^*), (H, \psi), Y)$, where either Y is the vertex set of a cyclic alley P of length at most $3k + 2$ in (H^*, ψ^*) , H is the inner graph of P in (H^*, ψ^*) and $\psi = \psi^*|_H$, or $(H, \psi) = (H^*, \psi^*)$ and $Y = \emptyset$. The goal is to compute a tree decomposition for H of width $9k + 3\lfloor \alpha k \rfloor + 9$ with one bag containing at least all vertices of Y .

For solving this problem, the idea is to construct some cyclic alleys in (H, ψ) , to compute the inner graph (H', φ') for each such alley P , and to recursively solve ITD on $(k, \alpha, (H^*, \psi^*), (H', \varphi'), Y_P)$ with Y_P being the vertex set of P . Then, a tree decomposition (T', B') is computed for the graph H'' obtained from H by deleting all computed inner graphs. Moreover, (T', B') should contain a bag with exactly the vertices of Y_P for each computed cyclic alley P . If the precomputed cyclic alleys were chosen appropriately, we can guarantee that H'' is ℓ -outerplanar for a small value of ℓ . One can therefore compute (T', B') easily, put the vertices of Y into all bags of (T', B') , and finally connect (T', B') with the recursively constructed tree decompositions to a tree decomposition of H with all desired properties. However, for an efficient implementation, it is not really possible to implement a subroutine using the whole graph (H^*, ψ^*) as input parameter since the vertices of the subgraph (H^*, ψ^*) , which are considered in a current call of our subroutine, would then also be part of the subgraphs taken as input for previous calls. Instead of this, we use one global representation of our original graph (H^*, ψ^*) and a subroutine $\text{TreeComp}_{k,\alpha}$ accessing this global graph representation.

For $j \in \mathbb{N}$ and for a cyclic alley P in an embedded graph (G, φ) separating a vertex set U from all vertices with peeling number 1, the j -*inner graph* of P with respect to (G, φ) is the subgraph of G induced by the vertices v of the inner graph I of P with $\mathcal{N}_{\varphi|_I}(v) \leq j$. For a subgraph H of our original graph H^* , let $H(q)$ be subgraph of H induced by all vertices v with $\mathcal{N}_{\psi^*|_H}(v) \leq q$. Then for solving ITD on the instance $(k, \alpha, (H^*, \psi^*), (H, \psi), Y)$ for a subgraph H of H^* and $\psi = \psi^*|_H$, we call a subroutine $\text{TreeComp}_{k,\alpha}$ that takes the tuple $((H(q), \psi|_{H(q)}), Y)$ as input, where $q = k + \lfloor \alpha k \rfloor + 2$.

Our main algorithm constructs in a preprocessing phase before the first call of $\text{TreeComp}_{k,\alpha}$ the graph $(H^*(q), \psi^*|_{H^*(q)})$ in a time linear in the number of vertices of H^* . This is done by inserting an additional vertex s_1 into the outer face of H^* and applying Lemma 3.6.7, where we take $Z = \emptyset$ and A as the area consisting of all inner faces. We then obtain a tree decomposition for H^* by calling $\text{TreeComp}_{k,\alpha}((H^*(q), \psi^*|_{H^*(q)}), \emptyset)$.

We next describe the routine $\text{TreeComp}_{k,\alpha}$ on an input $((H, \psi), Y)$, where we assume that Y is a cyclic alley in H^* with $|Y| \leq 3k + 2$ and that (H, ψ) is the q -inner graph of Y in (H^*, ψ^*) .

tree decomposition **function** $\text{TreeComp}_{k,\alpha}(\text{emb. graph } (H, \psi), \text{cyclic alley } Y)$

Step 1: Take U as the vertex set of H . In $O(|U|)$ time, determine for H the biconnected components and the block-cutpoint tree of H . In $O(|V'|)$ time, compute for each biconnected component $H' = (V', E')$ of H the triconnected components of H' by constructing the SPQR tree for H' . For example, one can use an implementation given by Gutwenger and Mutzel [45]. Beside the edges of H' , each triconnected component may contain additional virtual edges; however, since each triconnected component is a minor of H' , the treewidth

of each triconnected component is bounded by the treewidth of H' . Again in $O(|V'|)$ time, for each triconnected component $K = (V_K, E_K)$, choose an embedding ζ with $\zeta|_{H'[V_K]} = \psi|_{H'[V_K]} = \psi^*|_{H'[V_K]}$, and compute the peeling numbers of the vertices in K with respect to ζ .

Step 2: For each triconnected component K with at most 3 vertices, compute a tree decomposition consisting of only one bag containing the vertices of K and of Y in $O(k)$ time. Moreover, for each triconnected embedded component (K, ζ) with more than 3 vertices and no vertex v with $\mathcal{N}_\zeta(v) = q$, construct a tree decomposition for K of width $3(q-1) - 1$ in $O(q^2|V_K|)$ time (Corollary 3.5.13). In $O(q|V_K|)$ time, put additionally the vertices of Y into all bags of the tree decomposition so that the treewidth increases to $3q + 3k - 2$.

Step 3: For each triconnected embedded component (K, ζ) containing a vertex v with $\mathcal{N}_\zeta(v) \geq q$, determine the embedded graph (G, φ) that can be obtained from (K, ζ) by merging, for each maximal connected component consisting only of vertices v with $\mathcal{N}_\zeta(v) \geq q$, its vertices to a single vertex. In detail, determine the connected components of the subgraph of K induced by all vertices of peeling number of exactly q , and merge each such component to one vertex. This construction can be done in $O(|V_K|)$ time and does not increase the treewidth since G is a minor of K . We later show that G is still triconnected. Take $G = (V, E)$ and apply Steps (a)-(f) to (G, φ) .

- (a) Construct a good mountain structure $\mathcal{M} = (G, \varphi, G', \varphi', \downarrow, \mathcal{S})$ and a set \mathcal{P} of cyclic alleys of length at most $3k + 2$ such that both properties of Corollary 3.7.6 hold, where the coast of (G, φ) consists of all vertices v with $\mathcal{N}_\varphi(v) \leq \lfloor \alpha k \rfloor + 1$. This can be done in $O(|V|k^3 \log k)$ time. Within the same time bound, compute additionally for each $P \in \mathcal{P}$ the inner graph I in (G, φ) of P , the embedding $\varphi|_I$ as well as the mapping m that maps each $P \in \mathcal{P}$ to the set consisting of all (\mathcal{S}, φ') -components containing a crest enclosed by P .
- (b) For each (\mathcal{S}, φ') -component $C = (V_C, E_C)$, construct a tree decomposition for C of width $3q - 1 = 3k + 3\lfloor \alpha k \rfloor + 5$ in $O(q^2|V_C|)$ time such that, for each crest separator $X \in \mathcal{S}$ with a top edge in C , there is a bag $B_{C,X}$ containing all vertices part of both, C and the essential boundary of X (Corollary 3.5.13).
- (c) In $O(|V|)$ time, determine the graph G^* obtained from G by removing, for each cyclic alley $P \in \mathcal{P}$, the vertices not part of P , but part of the inner graph I of P in (G, φ) . Subsequently, construct a tree decomposition (T', B') of width $(3q - 1) + 2(3k + 2) = 9k + 3\lfloor \alpha k \rfloor + 9$ for the graph G^* such that Y is part of all bags and such that, for each cyclic alley $P \in \mathcal{P}$, there is a bag containing all vertices of P . More precisely, this is done as follows:
 - In total time $O(k|V|)$, for each cyclic alley $P \in \mathcal{P}$ and each (\mathcal{S}, φ') -component $C \in m(P)$, remove the vertices in G , but not in G^* , from all bags of the tree decomposition for C constructed in Step (b), and then add the vertices of P to all bags of the tree decomposition. Note that each bag contains afterwards at most $(3q - 1) + 3k + 2$ vertices.

- In $O(k|V|)$ time, for each (\mathcal{S}, φ') -component C , add additionally the set Y to each bag of the tree decomposition for C . After that, each bag contains at most $(3q - 1) + 2(3k + 2)$ vertices since $|Y| \leq 3k + 2$.
 - For each pair of (\mathcal{S}, φ') -components C_1 and C_2 being adjacent in the mountain connection tree of \mathcal{M} and hence having a common top edge e of a crest separator $X \in \mathcal{S}$, connect the bags $B_{C_1, X}$ and $B_{C_2, X}$ by an edge. This step takes $O(|V|)$ time. By Lemma 3.3.25, we obtain a tree after the edge insertions. Moreover, $(\text{TD2})_v$ is true for all vertices v not part of a $P \in \mathcal{P}$ by Lemma 3.3.27 and true for the remaining vertices v by Lemma 3.3.27 and Corollary 3.7.6(i). Thus, we obtain a tree decomposition (T', B') of width $3q - 1 + 2(3k + 2)$ with the desired properties.
- (d) In total time $O(|V_K|)$, compute for each cyclic alley $P \in \mathcal{P}$ the inner graph I' of P in (K, ζ) and its embedding $\zeta|_{I'}$. I' and $\zeta|_{I'}$ can be obtained from the inner graph I of P in (G, φ) and from the corresponding embedding $\varphi|_I$ constructed in Step (a) by demerging all vertices v with $\mathcal{N}_\zeta(v) = q$.
- (e) For each cyclic alley $P \in \mathcal{P}$, construct the q -inner graph $I(q)$ of P in (H^*, ψ^*) . Given the embedded inner graph $(I, \psi^*|_I)$ of P in (H^*, ψ^*) , and the vertices v of I with $\mathcal{N}_{\psi^*|_I}(v) = 1$, it would be easy to determine $I(q)$ in a time linear in the number of vertices in the $(q + 1)$ -inner graph (Lemma 3.6.7). Unfortunately, we are not given the whole embedded graph $(I, \psi^*|_I)$. However, we are given the global embedding ψ^* of H^* and, because of Step (d), the embedding $\zeta|_{I'}$ of the inner graph I' of P in (K, ζ) . The faces of $\psi^*|_I$ adjacent to the vertices v of I with $v \notin U$ are exactly the faces in (H^*, ψ^*) , the faces adjacent to a vertex v with peeling number $\mathcal{N}_{\psi^*|_I}(v) < q$ are exactly the faces in $(I', \zeta|_{I'})$. Moreover, for each vertex v with peeling number $\mathcal{N}_{\psi^*|_I}(v) = q$, the list of the faces around v in clockwise order with respect to $(I, \psi^*|_I)$ can be split into a list of consecutive faces in (H^*, ψ^*) and into a list of consecutive faces in $(I', \zeta|_{I'})$. This allows us to apply Lemma 3.6.7 without constructing $\psi^*|_I$ explicitly; we use instead the embeddings ψ^* and $\zeta|_{I'}$ to determine the faces around each vertex v when this is necessary for the algorithm in the proof of Lemma 3.6.7.
- (f) For each q -inner graph $I(q)$ of a cyclic alley $P \in \mathcal{P}$ in (H^*, ψ^*) , call recursively $\text{TreeComp}_{k, \alpha}((I(q), \psi^*|_{I(q)}), Y_P)$ with Y_P being the vertex set of P to obtain a tree decomposition for I with a bag containing Y_P . Connect this bag by an edge to a bag containing Y_P in the tree decomposition (T', B') constructed in Step (c). This leads to a tree decomposition of width $9k + 3\lfloor \alpha k \rfloor + 9$ for (G, φ) or, more precisely, for the graph obtained from G by decontracting all vertices of peeling number q and by inserting all vertices and edges of the inner graphs in (H^*, ψ^*) for all cyclic alleys $P \in \mathcal{P}$. In the following, denote this decontracted graph by $D(K)$. Apart from the recursive call, this step needs $O(|V_K|)$ time.

Step 4: For each pair of triconnected components K_1 and K_2 in the same biconnected component H' of H with K_1 and K_2 being connected by an edge in the SPQR tree for H' and for the set S of common vertices of K_1 and of K_2 , add an edge between a bag B_1 and a bag B_2 both containing the vertices of

S , where B_i ($i \in \{1, 2\}$) should be part of the tree decomposition constructed for $D(K_i)$. Observe that such bags exist since, by the definition of triconnected components in a biconnected graph, the two common vertices of two adjacent triconnected components in an SPQR tree must be adjacent in both of the two triconnected components. Note also that SPQR trees are defined in such a way that, for two triconnected components containing a common set S of vertices, the path connecting the two components in the SPQR tree consists completely of triconnected components that also contain all vertices of S . Hence, a tree decomposition (T'', B'') of width $9k + 3\lfloor \alpha k \rfloor + 9$ is obtained for each biconnected component H' of H , or more precisely, for the graph obtained from H' by inserting all vertices and edges of the inner graphs in (H^*, ψ^*) for all cyclic alleys $P \in \mathcal{P}$. Similarly to Step 3(f), denote this graph by $D(H')$. It is easy to see that the total running time of this step is $O(|U|)$.

Step 5: In $O(|U|)$ time, for each cutpoint u of H , create a bag B_u only containing u and, for each biconnected component H' containing u , insert an edge between B_u and a bag containing u in the tree decomposition of width $9k + 3\lfloor \alpha k \rfloor + 9$ for $D(H')$. By the definition of blockcut-point trees, a tree decomposition (T''', B''') is found for H , or more precisely, for the graph obtained from H by inserting all vertices and edges of the inner graphs in (H^*, ψ^*) for all cyclic alleys $P \in \mathcal{P}$. Similarly to the previous steps, let us denote this graph by $D(H)$, and return (T''', B''') as a tree decomposition for $D(H)$.

For the correctness of the algorithm above, it remains to show that merging the vertices of peeling number q in a triconnected embedded q -outerplanar graph (K, ζ) leads to a triconnected graph G . Hence, assume for a contradiction that there are two vertices u_1 and u_2 in $G = (V, E)$ with $\{u_1\}$ and $\{u_2\}$ being disconnected in G by a separator of size at most 2. Let S be such a separator of minimal size. If the vertices of S all have peeling number at most $q - 1$ and therefore also belong to K , S would be also a separator in K . Thus, at least one vertex $v \in S$ is a vertex of peeling number q in (G, φ) . Moreover, there is a cycle P in G enclosing v consisting exclusively of vertices with peeling number $q - 1$ in (G, φ) . Note that each vertex in $N_G(v)$ is part of P . Since there is at most one vertex of P part of $S \setminus \{v\}$, P and also $N_G(v) \setminus S$ remain to be connected in $G[V - S]$. This implies that v is adjacent only to the vertices of one connected component of $G[V - S]$. This means that $S \setminus \{v\}$ also disconnects $\{u_1\}$ and $\{u_2\}$, which is a contradiction to the minimality of S .

Hence, if we call $\text{TreeComp}_{k,\alpha}((H^*(q), \psi^*|_{H^*(q)}), \emptyset)$, we obtain a tree decomposition for $H^* = D(H^*(q))$ of width $9k + 3\lfloor \alpha k \rfloor + 9$. Let us next analyze the running time.

Lemma 3.7.7. *Let α be a real number with $0 < \alpha \leq 1$. For an embedded graph (H^*, ψ^*) with $H^* = (V^*, E^*)$ of treewidth k , $\text{TreeComp}_{k,\alpha}((H^*(q), \psi^*|_{H^*(q)}), \emptyset)$ runs in $O(|V^*|k^3 \min(1/\alpha, k) \log k)$ time.*

Proof. We first determine the running time for solving ITD on an instance $(k, \alpha, (H^*, \psi^*), (H, \psi^*|_H), Y)$, or more precisely, the running time for calling $\text{TreeComp}_{k,\alpha}((H(q), \psi^*|_{H(q)}), Y)$. Let U —as in the algorithm—be the vertex set of $H(q)$, and let U' be the vertex set of $H(2q + 1)$. Apart from Step 3(a) and apart from the recursive call of $\text{TreeComp}_{k,\alpha}$ in Step 3(f), the running time of each step of $\text{TreeComp}_{k,\alpha}$ is of size $O(q^2|U^*|) = O(k^2|U^*|)$ with U^* being the set of vertices of the subgraphs considered in this step. Hence apart from

Step 3(a) and the recursive calls, we obtain a running time of size $O(k^2|U'|)$. Note that Step 3(e) is the reason for writing $O(k^2|U'|)$ instead of $O(k^2|U|)$. The running time of Step 3(a) summed up over all triconnected components of one recursion depth is $O(|U|k^3 \log k) = O(|U'|k^3 \log k)$. By each recursive call of $\text{TreeComp}_{k,\alpha}$, the peeling number of each fixed vertex with respect to (H, ψ) decreases by at least $\lfloor \alpha k \rfloor + 1$. This means that each vertex is considered in at most $O(\min(1/\alpha, k))$ recursive calls. Therefore, the total running time of the algorithm can be bounded by $O(|V^*|k^3 \min(1/\alpha, k) \log k)$. \square

We now come to our main result:

Theorem 3.7.8. *Let α be a real number with $0 < \alpha \leq 1$. Given an embedded graph (G, φ) of treewidth k , one can compute a tree decomposition of width $9k + 3\lfloor \alpha k \rfloor + 9$ in $O(|V^*|k^3 \min(1/\alpha, k) \log k)$ time.*

Chapter 4

Outerplanarity Index

4.1 Another Complexity Parameter

Known algorithms solving NP-hard graph problems to optimality do not run in polynomial time and require mostly exponential running time. Given a graph, let n be the number of its vertices. Using the treewidth as an additional parameter k , we can observe that several NP-hard graph problems are computable in a time that is polynomial in n and that is exponential only in k .

Definition 4.1.1 (fixed-parameter algorithm). A *fixed-parameter algorithm* is an algorithm with a running time $O(f(\ell) \cdot n^{O(1)})$, where n is the size of the input instance, f is a function, and ℓ is an additional parameter.

For denoting more explicitly that we want to use a parameter to measure the running time, we call it a *complexity parameter*. The treewidth of a graph is such a complexity parameter which is unfortunately NP-hard to determine on general graphs [4]. Moreover, on other graph classes, e.g., the planar graphs, polynomial-time algorithms for finding a tree decomposition of smallest width are unknown. We next introduce on planar graphs a new complexity parameter, which can be computed in polynomial time.

Definition 4.1.2 (outerplanarity index). The *outerplanarity index* of a graph G is the smallest ℓ such that G is ℓ -outerplanar.

Observe that the outerplanarity index is a very natural parameter on planar graphs to use for fixed-parameter algorithms. Baker [7] has shown that many NP-hard problems can be solved on ℓ -outerplanar n -vertex graphs G in $O(c^\ell n)$ time for a constant c if an ℓ -outerplanar embedding of G is given.

In this chapter we focus on an algorithm that determines the outerplanarity index ℓ of an n -vertex graph in $O(n^2)$ time improving an $O(\ell^3 n^2)$ -time algorithm for the same problem from Bienstock and Monma [12]. Additionally, we see in Section 4.6 that giving up optimality allows a further improvement of the running time. In detail, a linear-time algorithm for the outerplanarity index is presented, which has an approximation ratio of 4. Using the approximation algorithm and Baker's technique, we can solve many NP-hard problems to optimality on ℓ -outerplanar graphs in $O(c^\ell n)$ time for some constant c (e.g.,

maximum independent set in $O(8^{4\ell}n)$ time). Thus, this approach as well as the algorithm of Section 3.7 are the fastest algorithm for many NP-hard problems for planar graphs whose outerplanarity index ℓ is within a constant factor of their treewidth.

Using the ratcatcher algorithm of Seymour and Thomas [83] and the results of Gu and Tamaki [43], one can obtain a tree decomposition of width $O(\ell)$ for an ℓ -outerplanar graph in $O(n^3)$ time. However, using the new algorithm presented in this chapter and Theorem 3.5.14, we obtain the following result.

Corollary 4.1.3. *Given an n -vertex graph G with outerplanarity index ℓ a tree decomposition for G with treewidth $3\ell - 1$ and $12\ell - 4$ can be found in time $O(\ell n^2)$ and $O(\ell^2 n)$, respectively.*

By using Bodlaender's ideas [15] and the new algorithm of this chapter one can improve the latter running time from $O(\ell^2 n)$ to $O(\ell n)$.

4.2 Treewidth of ℓ -Outerplanar Graphs

For $\ell \in \mathbb{N}$, let \mathcal{G}_ℓ be the graph class consisting of all ℓ -outerplanar graphs. Before an algorithm is presented for computing the outerplanarity index, let us analyze the exact treewidth of the graph class \mathcal{G}_ℓ . The upper bound follows from Lemma 3.5.14, i.e., all graphs in \mathcal{G}_ℓ have treewidth at most $3\ell - 1$. The exact lower bound is common knowledge; however, it seems to the author that no one has published it. An easy lower bound can be obtained by the combination of Lemma 1.4.5 and 1.4.6 since the $((2\ell+2) \times (2\ell))$ -grid is an example of an ℓ -outerplanar graph having treewidth 2ℓ . We next show that there are ℓ -outerplanar graphs of larger treewidth.

Lemma 4.2.1. *For all $\ell, n \in \mathbb{N}$ with $n \geq 2\ell^3 + 6\ell^2$, there exists an ℓ -outerplanar graph G with n vertices and treewidth $\geq 3\ell - 1$.*

Proof. Let G be the graph obtained from a $(2\ell \times 2\ell)$ -grid G_1 and a $(2\ell(\ell + 1) \times \ell)$ -grid G_2 by connecting, for all $i \in \{0, \dots, 2\ell - 1\}$, the rightmost vertex of the $(i + 1)$ -th row of G_1 by $\ell + 1$ edges with the leftmost vertices of the $(i \cdot (\ell + 1) + 1)$ th, \dots , $((i + 1) \cdot (\ell + 1))$ th row of G_2 (see Fig. 4.2.1). Since G_1 has only 2ℓ rows and since G_2 has only ℓ columns, G is ℓ -outerplanar. For proving that the treewidth of G is at least $3\ell - 1$, we consider the one-robber- $(3\ell - 1)$ -cops-game on G . Let us define an *extended row* as a tuple (r, e, s) of a row r of G_1 connected by edge e to a row s of G_2 . Note that, for each row r of G_1 , there are $\ell + 1$ extended rows containing r . The robber wins the game since he can move such that, before and after each round, one of the two following invariants holds:

- (I1) At most $2\ell - 1$ cops are in G_1 and the robber is in a cop-free extended row.
- (I2) At most $\ell - 1$ cops are in G_2 and the robber is in a cop-free column of G_2 .

For the next conclusions, always keep in mind that there are in total $3\ell - 1$ cops. On the one hand, if there are at most $2\ell - 1$ cops in G_1 , there is at least one cop-free row r_1 in G_1 . If there is no cop-free extended row containing r_1 , then $\ell + 1$ cops must be in G_2 and there is another cop-free row r_2 in G_1 . If

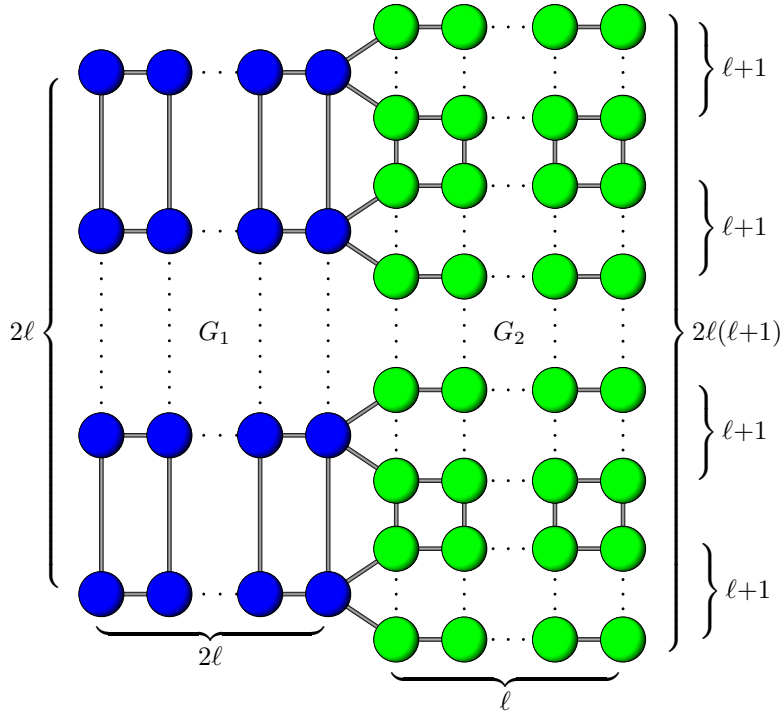


Figure 4.2.1: The graph G consisting of two grids G_1 and G_2 .

this row again is not part of a cop-free extended row, there is a third row r_3 in G_1 that now definitively must be part of a cop-free extended row. This means that if there are at most $2\ell - 1$ cops in G_1 , there is an extended cop-free row. On the other hand, at least one column of G_2 is cop-free if there are at least 2ℓ cops in G_1 . Thus, the robber can find an initial position such that (I1) or (I2) holds before the first round. Let us now analyze a fixed round of the game.

Assume that (I1) holds before the round. If one cop lifts his helicopter and wants to move such that after the round again there will be at most $2\ell - 1$ cops in G_1 , the robber moves along his cop-free extended row to a current cop-free column of G_1 and then along this column to an extended row being cop-free after the movement of the cop. Otherwise at least 2ℓ cops are in G_1 after the round. Then the robber moves along his extended row to a column of G_2 being cop-free with respect to the next positions of the cops.

Let us next analyze what happens if (I2) holds before the round. Then the robber stays in a cop-free column of G_2 . If after the next movement of the cops there are still at most $\ell - 1$ cops in G_2 , then there will be a new cop-free column c . Then the robber can move along his current column to a (non-extended) cop-free row of G_2 and then along this row to column c . Otherwise there will be a cop-free extended row after the next movement of the cops, and the robber can move along his current column to this row.

Hence the robber has a winning strategy and this proves the lemma for $n = 2\ell^3 + 6\ell^2$. For $n > 2\ell^3 + 6\ell^2$ just connect the endpoint of a simple path of length $n - (2\ell^3 + 6\ell^2 + 1)$ by an edge to an arbitrary vertex of G . \square

Corollary 4.2.2. *The graph class \mathcal{G} consisting of all ℓ -outerplanar graphs has exactly treewidth $3\ell - 1$.*

Having the knowledge that a graph class \mathcal{G} consisting of all ℓ -outerplanar graphs has treewidth $3\ell - 1$ does not mean that we have a tree decomposition of width $3\ell - 1$. Before we can construct a tree decomposition by the use of Lemma 3.5.14, we need for each graph $G \in \mathcal{G}$ an ℓ -outerplanar embedding.

4.3 Ideas of the Algorithm

One way to solve a graph problem is to consider the problem on increasing subgraphs of a given graph G . Such an approach is easily possible if we have the tree decomposition for G . However, we want to determine the outerplanarity index for a graph G without the knowledge of a tree decomposition for G . The idea will be to solve the problem by removing some subgraph C from G , solve the problem recursively on the remaining graph G' as well as on C , and finally use the solution obtained to solve the problem on G . Unfortunately, the outerplanarity index of G is not a function of the outerplanarity indices of G' and C alone. However, one can determine the outerplanarity index of G by a computation of the so-called *weighted outerplanarity index* of G' and C , which is a small generalization of the outerplanarity index that additionally takes into account the weighted outerplanarity indices of 6 different graphs, each of which is a supergraph of G' with a few additional edges. It turns out that by one recursive call one can compute the weighted outerplanarity indices of the 6 graphs simultaneously. For the time being let us consider only the outerplanarity index.

Also to find the outerplanarity index we need the definition of peeling. See possibly again page 47. Note that a peeling can be considered intuitively as a process that removes the vertices of a graph in successive steps, each of which removes all vertices incident to the outer face. In this context, the peeling index of an embedded graph (G, φ) is the minimal number of steps of a peeling to remove all vertices. Let us call an embedding *optimal* (*c*-*approximative*) if it is a rooted embedding φ of a planar graph G such that the peeling index of (G, φ) equals (is bounded by c times) the outerplanarity index of G .

Suppose that we are given a triconnected planar graph G . Then, the combinatorial embedding of G is unique and can be found in linear time [93, 23, 66]. Let F be a face of the combinatorial embedding of G . Hence, the rooted embedding φ of G with F chosen as the outer face is also unique. Obviously, we can determine the peeling index ℓ of (G, φ) in linear time. Let φ^{OPT} be an optimal embedding of G . Define F^{OPT} as the outer face of φ^{OPT} and ℓ^{OPT} as the peeling index of $(G, \varphi^{\text{OPT}})$. If the distance of two faces F' and F'' is taken to be the minimal number $q \in \mathbb{N}$ of faces $F' = F_1, \dots, F_q = F''$ such that two consecutive faces have a common edge on their boundary, the distance in φ^{OPT} from F to F^{OPT} and from F^{OPT} to any other face is at most ℓ^{OPT} . Consequently, $\ell \leq 2\ell^{\text{OPT}}$. By Lemma 3.2.3, a combinatorial embedding of a graph with n vertices has at most $2n$ faces. Thus, we can simply iterate over all faces of G and find an optimal embedding in that way.

Lemma 4.3.1. *The peeling index of an arbitrary embedding of a triconnected planar graph G is at most twice as large as the smallest possible peeling index of an embedding for G .*

Corollary 4.3.2. *Given a triconnected planar graph G , a 2-approximative and an optimal embedding of G can be obtained in linear and in quadratic time, respectively.*

4.4 Extended Peelings

In this section, we define extended peelings, the weighted outerplanarity index, and a special kind of peninsulas. Then a description is given for computing the weighted outerplanarity index on a graph G by removing this special kind of peninsula from G and solving the problem recursively.

For a rooted embedding φ , let us call a vertex or edge *outside* (with respect to φ) if it is incident to the outer face of φ . Let $G = (V, E)$ be a connected embedded graph with weight function $r : V \cup E \rightarrow \mathbb{N}$, and let φ be a rooted embedding of G . Define $Out(\varphi)$ as the set of vertices outside with respect to φ . If $\mathcal{S} = Out(\varphi)$ and $u, v \in \mathcal{S}$ are two different vertices, $\mathcal{S}_{u \rightarrow v}^\varphi \subseteq \mathcal{S}$ is the maximal set of vertices part of the series from u to v appearing clockwise around the outer face of φ such that this series contains u as well as v only once. In particular, this means that, if $u = v$, $\mathcal{S}_{u \rightarrow v}^\varphi = \{u\}$. Given a directed edge $e = (u, v)$, we say that an embedding φ' is obtained from φ by *adding e clockwise* if φ' is a (planar) rooted embedding of $(V, E \cup \{e\})$ such that $Out(\varphi') = Out(\varphi)_{v \rightarrow u}^\varphi$ and all inner faces of φ are also inner faces of φ' . By iterating over a set of directed edges we can *add a set clockwise* into an embedded graph. The embedding obtained does not depend on the order in which edges are added since several edges can be added clockwise only if they do not interlace. Let us say that a set of directed edges is *embeddable* in φ if the set can be added clockwise into φ .

Let E^+ be a set of directed edges being embeddable in φ , and let φ^+ be the embedding obtained. The *extended peeling* \mathcal{P} of the quadruple (G, φ, r, E^+) is the (unique) list of vertex sets (V_1, \dots, V_t) with $t \in \mathbb{N}$, $\bigcup_{i=1}^t V_i = V$, and $V_t \neq \emptyset$ such that V_i are exactly the vertices outside with respect to $\varphi^+|_{G[V \setminus (V_1 \cup \dots \cup V_{i-1})]}$. In other words, each set V_i ($1 \leq i \leq t$) contains all vertices that are incident to the outer face obtained after the removal of the vertices in V_1, \dots, V_{i-1} in the initial embedding φ^+ . Thus, we also say that the *peeling step i* removes the vertices in V_i . We also denote t as the *total peeling number* of \mathcal{P} and, if $E^+ = \emptyset$, also of φ . Note that the embedding φ restricted to the vertices and edges of the subgraph $G[V \setminus \bigcup_{j=1}^i V_j]$ is a unique embedding. For a vertex $v \in V_i$, we call $i + r(v)$ the *extended peeling number* of v with respect to \mathcal{P} also denoted by $\mathcal{P}(v)$. For a set $V' \subseteq V$, we let $\mathcal{P}(V') = \bigcup_{v \in V'} \mathcal{P}(v)$. Moreover, the *extended peeling number* of an edge $\{u, v\}$ with $u \in V_i$ and $v \in V_j$ ($1 \leq i, j \leq t$) is $\mathcal{P}(\{u, v\}) = \min\{i, j\} + r(\{u, v\})$, and $\mathcal{P}(G) = \max\{\max_{v \in V} \mathcal{P}(v), \max_{e \in E} \mathcal{P}(e)\}$. To have a simpler notation in the following, there is a slight difference in the definition of a peeling number and an extended peeling number for a non-horizontal edge. For avoiding any confusion, we consider only extended peelings in the remainder of this chapter.

The *weighted outerplanarity index* of G with respect to a weight function r is the minimal extended peeling number of the extended peelings of $(G, \varphi', r, \emptyset)$

over all rooted embeddings φ' of G . Note that the outerplanarity index of G is the weighted outerplanarity index in the special case $r \equiv 0$.

Recall that, if H is a peninsula of G , then either $H = G$ or each H -attached component C of G has at most two vertices in common with H , i.e., $|\mathcal{R}_H(C)| \leq 2$. An *outer component* C of a peninsula H of G is defined in φ as H itself or as an H -attached component of G such that C contains an edge outside with respect to φ .

Definition 4.4.1 (inherited embedding of induced graphs). Let (G, φ) be an embedded graph and H be a peninsula of G .

For the induced graph $H(G)$, an *inherited embedding* $\varphi|_{H(G)}$ is a rooted embedding ψ of $H(G)$ that can be obtained from φ by iteratively applying one of the following modifications:

- remove a vertex of degree 0,
- remove an edge $\{u, v\}$, i.e., if $\varphi((u', u)) = (u, v)$, $\varphi((v, u)) = (u, u'')$, $\varphi((v', v)) = (v, u)$, and $\varphi((u, v)) = (v, v'')$ holds before the removal, redefine $\varphi((u', u)) = (u, u'')$ and $\varphi((v', v)) = (v, v'')$,
- merge two edges $\{u, x\}$ and $\{x, v\}$ with a common endpoint x of degree 2 to one edge $\{u, v\}$, i.e., if $\varphi((u', u)) = (u, x)$, $\varphi((x, u)) = (u, u'')$, $\varphi((v', v)) = (v, x)$, and $\varphi((x, v)) = (v, v'')$ holds before the removal, redefine $\varphi((u', u)) = (u, v)$, $\varphi((v, u)) = (u, u'')$, $\varphi((v', v)) = (v, u)$, and $\varphi((u, v)) = (v, v'')$.

See Fig. 4.4.1 for an example. If an induced embedding is obtained by merging a set E' of edges to a virtual edge $e = \{u, v\}$, we say that e is *mapped* to the cyclic position around u of the edge in E' incident to u . Informally, e is embedded as the path consisting of the edges in E' . Apart from choosing between two possible outer faces, an inherited embedding of an induced peninsula is unique if, for each virtual edge $\{u, v\}$, the set of edges to whose cyclic position $\{u, v\}$ can be mapped around u is a set of consecutive edges around u , none of which is part of the peninsula. Let us call a peninsula H *good* if this is the case.

In the rest of this section, we consider different properties of extended peelings and of inherited embeddings. For the next observation, recall that the inner graph of a cycle S in an embedded graph (G, φ) is an embedded graph (C, ζ) such that C is a maximal subgraph of G and such that exactly the vertices of S are outside with respect to $\zeta = \varphi|_C$.

Observation 4.4.2. Let (G, φ) and (G', φ') be embedded graphs, let r be a weight function for both graphs, let E^+ be a set of directed edges embeddable in both embedded graphs, and let S be a vertex set inducing a cycle in G as well as in G' . Moreover, take (C, ζ) as the inner graph of S in (G, φ) , $V(C)$ as the vertex set of C , (C', ζ') as the inner graph of S in (G', φ') and $V(C')$ as the vertex set of C' . If $G[V(C)]$ and $G'[V(C')]$ are the same graphs as well as $\varphi|_{G[V(C)]}$ and $\varphi'|_{G'[V(C')]}$ are the same embeddings and if, for extended peelings \mathcal{P} of (G, φ, r, E^+) and \mathcal{P}' of (G', φ', r, E^+) , $\mathcal{P}(v) - \mathcal{P}'(v)$ is the same for all $v \in S$, then $\mathcal{P}(v) - \mathcal{P}'(v)$ is the same for all $v \in C$.

Let (G, φ) be a connected embedded graph with a weight function r and let $H = (V_H, E_H)$ be a good peninsula with an edge $\{u^*, v^*\}$ outside with respect

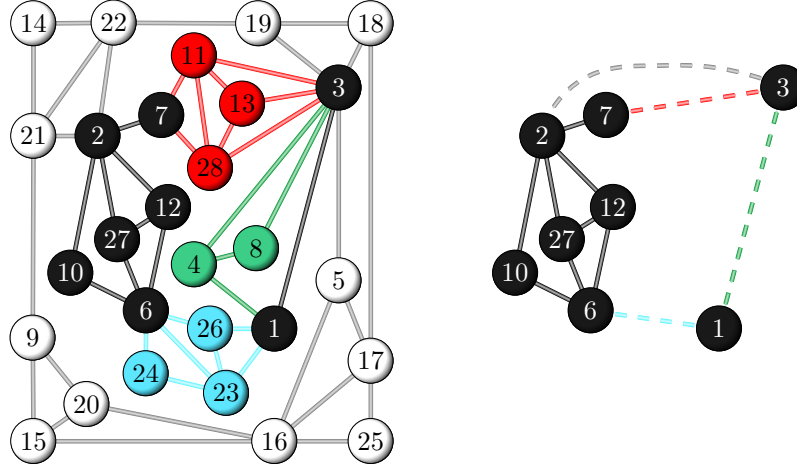


Figure 4.4.1: The left side shows an embedded graph (G, φ) with a peninsula H (black), an outer component of H in G (all white vertices and vertices 2 and 3) and three further H -attached components of G . On the right side, we see an inherited embedding $\varphi|_{H(G)}$ with the virtual edges drawn dashed.

to φ . We now want to compare the extended peeling of (G, φ, r, E^+) with the extended peeling of an inherited embedding of $(H(G), \varphi|_{H(G)}, r, E^+)$ for some r and E^+ . Moreover, if we consider the peeling as a process iteratively removing the vertices of smallest peeling number, what happens with an H -attached component $C = (V_C, E_C)$ of G during the extended peeling of (G, φ, r, E^+) ?

Define $\mathcal{L}(V^+)$ for a set $V^+ = \{u\}$ as the list of edge sets $(\{(u, u)\}, \{\})$ and for a set $V^+ = \{u, v\}$ as $(\{(u, v), (v, u)\}, \{(u, u)\}, \{(v, v)\}, \{\}, \{(v, u)\}, \{(u, v)\})$. Fig. 4.4.2 shows the directed edges of $\mathcal{L}(\{u, v\})$. Take E^+ as a set in the list $\mathcal{L}(\{u^*, v^*\})$ and $\mathcal{P} = (V_1, V_2, \dots, V_t)$ as the extended peeling of (G, φ, r, E^+) . Choose u and v such that $\{u, v\} = \mathcal{R}_H(C)$ —possibly $u = v$. If $u \in V_i$ and $v \in V_j$ ($1 \leq i, j \leq t$), take $q = \min\{i, j\}$. Note that C is also a peninsula. Define $\mathcal{S} = \text{Out}(\varphi|_C)$ and H' as the outer component of C that is a supergraph of H and that contains $\{u^*, v^*\}$. Let \mathcal{P}' be the extended peeling of $(H'(G), \varphi|_{H'(G)}, r', E^+)$, where r' is an arbitrary weight function such that r' is equal to r for all vertices of $H'(G)$. The following two properties are proved below.

Property 1. $V_q \cap V_C$ equals one of the six sets below. In other words, if we ignore the vertices not in C , peeling step q of \mathcal{P} removes one of these six sets: 1. $\{u, v\}$, 2. $\{u\}$, 3. $\{v\}$, 4. \mathcal{S} , 5. $\mathcal{S}_{u \rightarrow v}^{\varphi|_C}$ or 6. $\mathcal{S}_{v \rightarrow u}^{\varphi|_C}$. The remaining vertices of \mathcal{S} are removed in peeling step $q + 1$.

Property 2. The removal of C from G if $|\mathcal{R}_H(C)| = 1$ and the replacement of C by the (virtual) edge $\mathcal{R}_{H'}(C)$ in G if $|\mathcal{R}_H(C)| = 2$ does not change the extended peeling numbers in H' , i.e., $\mathcal{P}(s) = \mathcal{P}'(s)$ for all s being a vertex or an edge of H' .

For each of the six sets of Property 1, Fig. 4.4.2 shows an example of a situation just before the peeling step q . In each example, the black edge corresponds

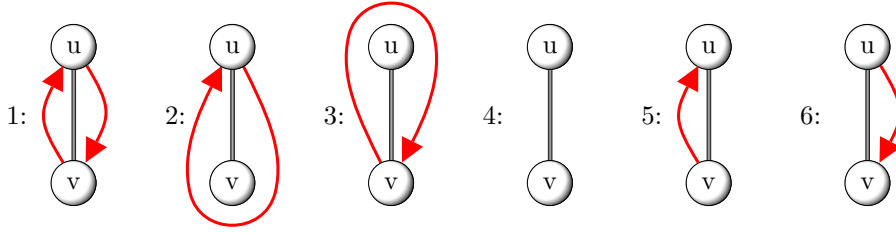


Figure 4.4.2: Possibilities for an edge to be outside, e.g., 1: Only the two endpoints are outside.

to C , and each directed edge corresponds to an undirected path in $G[V_q \cup \dots \cup V_i]$ using no edge of C .

Proof of Property 1. Consider the two cases $|\mathcal{R}_H(C)| = 1$ and $|\mathcal{R}_H(C)| = 2$. In the first case—i.e., $u = v$ —observe that if one vertex of $\mathcal{S} \setminus \{u\}$ is removed in peeling step q , then all vertices in \mathcal{S} are removed in that peeling step because there is no edge from $\mathcal{S} \setminus \{u\}$ to H . Consequently, $V_q \cap V_C$ equals $\{u\}$ or \mathcal{S} .

In the second case, we argue in the same way that if a vertex in $\mathcal{S}_{u \rightarrow v}^{\varphi|_C} \setminus \{u, v\}$ (in $\mathcal{S}_{v \rightarrow u}^{\varphi|_C} \setminus \{u, v\}$) is removed, all vertices in $\mathcal{S}_{u \rightarrow v}^{\varphi|_C}$ (in $\mathcal{S}_{v \rightarrow u}^{\varphi|_C}$) are then removed. Finally, note that the remaining vertices in \mathcal{S} are outside after peeling step q . \square

Proof of Property 2. Starting with the initial embedding $\varphi_0^{\mathcal{P}} = \varphi$ and $\varphi_0^{\mathcal{P}'} = \varphi|_{H'(G)}$ and carrying out in parallel the extended peelings \mathcal{P} and \mathcal{P}' , let us compare the embeddings $\varphi_i^{\mathcal{P}}$ and $\varphi_i^{\mathcal{P}'}$ obtained after $i \in \mathbb{N}$ peeling steps. By induction on i , we can observe that if a C -internal face is one that is incident only to vertices of C , there is the following bijection f between the non C -internal faces of $\varphi_i^{\mathcal{P}}$ and the faces of $\varphi_i^{\mathcal{P}'}$: Each face F in $\varphi_i^{\mathcal{P}}$ is mapped to the face whose boundary vertices are those of F , except for the vertices in C . Thus, we can conclude that a face F of $\varphi_i^{\mathcal{P}}$ is the outer face if and only if the corresponding face $f(F)$ of $\varphi_i^{\mathcal{P}'}$ is the outer face, i.e., a vertex in H' is outside after the same number of peeling steps in \mathcal{P} and in \mathcal{P}' . \square

Note that, for some embedding ζ of C , (C, ζ) is the inner graph of \mathcal{S} , and we thus can apply Observation 4.4.2. Informally, the extended peeling numbers of vertices in C are determined by the extended peeling numbers of vertices in \mathcal{S} . Because of that and Property 1, a set of edges $E' \in \mathcal{L}(\{u, v\})$ exists such that, for the extended peeling \mathcal{P}'' of $(C, \varphi|_C, r, E')$, we have $\mathcal{P}(v) = \mathcal{P}''(v) + q - 1$ for all $v \in C$. If we again consider a peeling as a process, then E' can be determined during the extended peeling \mathcal{P}' by observing

- which of the vertices are in $\mathcal{R}_H(C)$ and,
- if $|\mathcal{R}_H(C)| = 2$, which sides of the virtual edge $\mathcal{R}_H(C)$ are outside just before a first vertex in $\mathcal{R}_H(C)$ is removed.

For that reason, let us define E' as the *induced set of extra edges* with respect to \mathcal{P}' and to $\mathcal{R}_H(C)$. Figure 4.4.2 shows all possibilities for how a virtual edge (black) can be outside.

The removal and replacement described in Property 2, applied to all H -attached components of G in φ , results (after the removal of multiple edges) in an inherited embedding $\varphi|_{H(G)}$. For some function r' , let \mathcal{P} and \mathcal{P}^* be the extended peelings of (G, φ, r, E^+) and of $(H(G), \varphi|_{H(G)}, r', E^+)$, respectively. Informally, our goal is to peel the components independently. Thus, we want that, for each vertex s or edge s being in both G and $H(G)$, the extended peeling numbers of s with respect to \mathcal{P} and with respect to \mathcal{P}^* should coincide for each embeddable set E^+ of directed edges. Therefore, we define $r' = r$, except for the following two changes. The changes enable us to take into consideration the extended peeling numbers of all H -attached components to the computation of the extended peeling \mathcal{P}^* . The first change is to extend the domain of the weight function r' to the virtual edges and to define r' on the extension as follows: The weight r' of a virtual edge e is $\mathcal{P}^{**}(C^*) - 1$, where C^* is the union of all H -attached components with virtual edge e , \mathcal{P}^{**} is the extended peeling of $(C^*, \varphi|_{C^*}, r, E')$, and E' is the induced set of extra edges with respect to the extended peeling $(H(G), \varphi|_{H(G)}, 0, E^+)$ and to e . The second change is to increase $r'(v)$ by -1 plus the total peeling number of the embedding of an H -attached component C' with $\mathcal{R}_H(C') = \{v\}$ ($v \in V_H$). Leave $r'(v)$ unchanged if no such C' exists. Denote the first change the *extending change* w.r.t. $(G, H(G), \varphi|_{H(G)}, r, E^+)$, and the second change the *increasing change* w.r.t. (G, H, r) .

We say that a set S consisting of vertices and edges is *outside* with respect to an embedding if all $s \in S$ are outside with respect to the embedding. For a graph $G = (V, E)$ with a weight function r , a set of directed edges E' , and a set S consisting of one or two vertices, an embedding φ' is *vertex-constrained optimal* for (G, r, E', S) if S is outside with respect to φ' and if the following is true for all embeddings φ'' of G : Either S is not outside with respect to φ'' or $\mathcal{P}'(G) \leq \mathcal{P}''(G)$, where \mathcal{P}' and \mathcal{P}'' are the extended peelings of (G, φ', r, E') and (G, φ'', r, E') , respectively. If the same condition holds for $S = \{u^*, v^*, \{u^*, v^*\}\}$ with $u^*, v^* \in V$, the embedding φ' is called *edge-constrained optimal*.

We conclude this section with an algorithm to construct constrained optimal embeddings. Given a graph $G = (V, E)$ with a weight function $r : V \cup E \rightarrow \mathbb{N}$ and a set S consisting of one or two vertices in V and, in the latter case, possibly of an edge connecting these two vertices, one can find a vertex- or edge-constrained optimal embedding for (G, r, E^+, S) with $E^+ \in \mathcal{L}(S)$ as follows:

For a good peninsula H of G —possibly $H = G$ —that contains all vertices and edges of S , first determine recursively for each H -attached component C and for each $E' \in \mathcal{L}(\mathcal{R}_H(C))$ a vertex-constrained optimal embedding for $(C, r, E', \mathcal{R}_H(C))$.

Then, iterate over all embeddings of $H(G)$ such that all vertices and edges of S are outside with respect to the embedding. For each such embedding φ , compute $\mathcal{P}_\varphi(H(G))$ with respect to the extended peeling \mathcal{P}_φ of $(H(G), \varphi, r', E^+)$, where r' is the function obtained from both changes, the extending change w.r.t. (G, H, φ, r, E^+) and the increasing change w.r.t. (G, H, r) . Finally, take the embedding φ with $\mathcal{P}_\varphi(H(G))$ being as small as possible, and combine φ and the embeddings found recursively to an edge-constrained optimal embedding for (G, r, E^+, S) .

4.5 Biconnected Graphs

First of all, for a biconnected planar graph G and an edge e of G , let us define a *rooted SPQR tree* (T, M) of (G, e) as an SPQR tree of G with $T = (W, F)$ rooted at the unique node $r \in W$ with $e \in M(r)$. Moreover, we define the *subtree module* of a node w in T as the graph $M(T_w)$, and the *parent edge* e_p of w as follows: If w is the root of a rooted SPQR tree of (G, e) , take $e_p = e$; otherwise, take $e_p = \mathcal{R}_C(H)$ with $C = M(w)$, $H = M(w')$, and w' being the parent of w . The *reduced subtree module* of a node w is the subtree module of w without the parent edge of w .

Before we can apply the algorithm of the last section, we have to decompose our graph into good peninsulas. Given a biconnected planar graph G , this can be done in linear time by constructing an SPQR tree of G [11, 45], which we subsequently turn into a rooted SPQR tree (T, M) of (G, e) for an edge e of G . Take $T = (W, F)$. Observe that, for each node w of T , each $M(w)$ is the induced graph of a peninsula H with respect to G , and each reduced subtree module of a child w' of w is, for some virtual edge e , the union of all H -attached components having e as virtual edge. More exactly, if w' is no multiple-edge component, the reduced subtree module of w' is exactly one H -attached component and, otherwise, each of the reduced subtree modules of the children of w' is one H -attached component. Given additionally a weight function r , an edge e_{out} of G , and $E^+ \in \mathcal{L}(e_{\text{out}})$, our intermediate goal is the computation of an edge-constrained optimal embedding of $(G, r, E^+, e_{\text{out}} \cup \{e_{\text{out}}\})$ since we can later use this computation as an auxiliary procedure to find an optimal or an approximative embedding. Therefore, we use the following algorithm.

We initially compute a rooted SPQR tree (T, M) of (G, e_{out}) [11, 45]. This tree is then traversed bottom-up. Let w be the current node visited by the traversal, and let $e_p = \{u, v\}$ be the parent edge of w . Define $H = (V_H, E_H) = M(w)$ and C (C') as the (reduced) subtree module of w . In addition, set $H' = H$ if w is the root of T , and $H' = (V_H, E_H \setminus \{e_p\})$ otherwise. For each $E^+ \in \mathcal{L}(\{u, v\})$, we determine at w a constrained optimal embedding φ for (C', r, E^+, S) , where $S = e_p$, if w is not the root of T , and $S = e_{\text{out}} \cup \{e_{\text{out}}\}$ otherwise. By the algorithm described in the last section, such an embedding φ can be computed by considering the extended peeling of $(H', \varphi|_{H'}, r', E^+)$ for all embeddings $\varphi|_{H'}$ of H' with S being outside and with r' being the extended weight function of $(C', H', \varphi|_{H'}, r, E^+)$. The reason for this is that recursive calls have already found, for each child w' of w , the vertex-constrained optimal embeddings for the reduced subtree module of w' .

If H is a cycle component, there are only two possible rooted embeddings of H with u and v outside. The same is true if H is a triconnected component since a third rooted embedding would imply that φ has three faces being incident to u and v , i.e., there is a u - v -path separated from the rest of G by the two vertices in $\{u, v\}$. Thus, in both cases, there is only one rooted embedding of H' with u and v outside. Therefore, $\varphi|_{H'}$ and also an extension of the embeddings of the reduced subtree modules of the children of w to an embedding $\varphi|_{C'}$ can be found in time linear in the size of H' .

If H is a multiple-edge component, S has to be outside. Let u and v be the vertices of H . Depending on E^+ (and at the root also on S), we can choose $d \in \{0, 1, 2\}$ children of w such that their reduced subtree modules are (with at least one edge) outside with respect to the embedding obtained from $\varphi|_{C'}$ by adding

E^+ clockwise. Observe that the extended peeling numbers of a subtree module of a child of w differ only by at most one for different orders of the embeddings of the subtree modules between u and v . Let C_1, \dots, C_ℓ be the reduced subtree modules of the children of w . For a simpler notation, let us define in this paragraph the *inside peeling number* of C_i ($i \in \{1, \dots, \ell\}$) as the total peeling number in a constrained optimal embedding for $(C_i, r, \{(u, v), (v, u)\}, \{u, v\})$ and the *outside peeling number* of C_i as the smaller number of the total peeling number in the vertex-constrained optimal embeddings for $(C_i, r, \{(u, v)\}, \{u, v\})$ and for $(C_i, r, \{(v, u)\}, \{u, v\})$. Since the total peeling number of an embedding of C' equals the maximum over the total peeling number of all embeddings of C_i ($1 \leq i \leq \ell$), our goal is to reduce the largest numbers. Therefore, we choose for C' an embedding such that d reduced subtree modules of the children of w with largest inside peeling number have outside vertices with respect to $\varphi|_{C'}$ —intuitively, the total peeling number of these d embeddings decrease to the outside peeling number in that way. Finding d reduced subtree modules with largest inside peeling number, i.e., finding $\varphi|_{C'}$ can be done in a time linear in the number of virtual edges if we ignore the time for the recursive calls.

Lemma 4.5.1. *Given a biconnected graph G with a weight function r and an edge e_{out} of G , an edge-constrained optimal embedding for $(G, r, \emptyset, e_{\text{out}} \cup \{e_{\text{out}}\})$ can be found in linear time.*

By a similar approach, we also can find a vertex-constrained optimal embedding for $(G, r, \emptyset, e_{\text{out}})$. If e_{out} is an edge of G , compute an edge-constrained optimal embedding for $(G, r, \emptyset, e_{\text{out}} \cup \{e_{\text{out}}\})$. Otherwise, let G^* be the graph obtained from G by adding edge e_{out} . Compute then the SPQR (T, M) tree for (G^*, e_{out}) and traverse T bottom-up. The only difference to the algorithm for an edge-constrained optimal embedding is at the root H of T . Remove e_{out} from H and compute a vertex-constrained optimal embedding with the two vertices in e_{out} being outside. Informally, the peeling of H should consider H as invisible.

Lemma 4.5.2. *Given a biconnected graph G with a weight function r and two vertices e_{out} of G , a vertex-constrained optimal embedding for $(G, r, \emptyset, e_{\text{out}})$ can be found in linear time.*

For an optimal embedding of a given graph $G = (V, E)$, iterate over all edges $e \in E$. In each iteration, compute an edge-constrained optimal embedding of G with e outside. If we use initially the weight function $r \equiv 0$, then one of the embeddings found is optimal.

Moreover, for an approximative embedding of G , compute only for one arbitrary edge $e \in E$ an edge-constrained optimal embedding of G . The quality of the approximative embedding φ with edge e outside can be estimated with arguments similar to those used to prove Corollary 4.3.2. Let φ^{OPT} be an optimal embedding of G , and choose φ' as the rooted embedding such that φ^{OPT} and φ' have the same combinatorial embedding and such that e is outside with respect to φ' . The number of peeling steps to remove (G, φ') is at most twice the number of peeling steps to remove $(G, \varphi^{\text{OPT}})$. Since φ is an edge-constrained optimal embedding with $r \equiv 0$ and e outside, the peeling index of (G, φ) is bounded by the peeling index of (G, φ') .

Corollary 4.5.3. *Given a biconnected planar graph G , one can find a 2-approximative embedding of G in linear and an optimal embedding of G in quadratic time.*

See Figure 4.5.1 for an example of an extended peeling of the split components of a rooted SPQR tree.

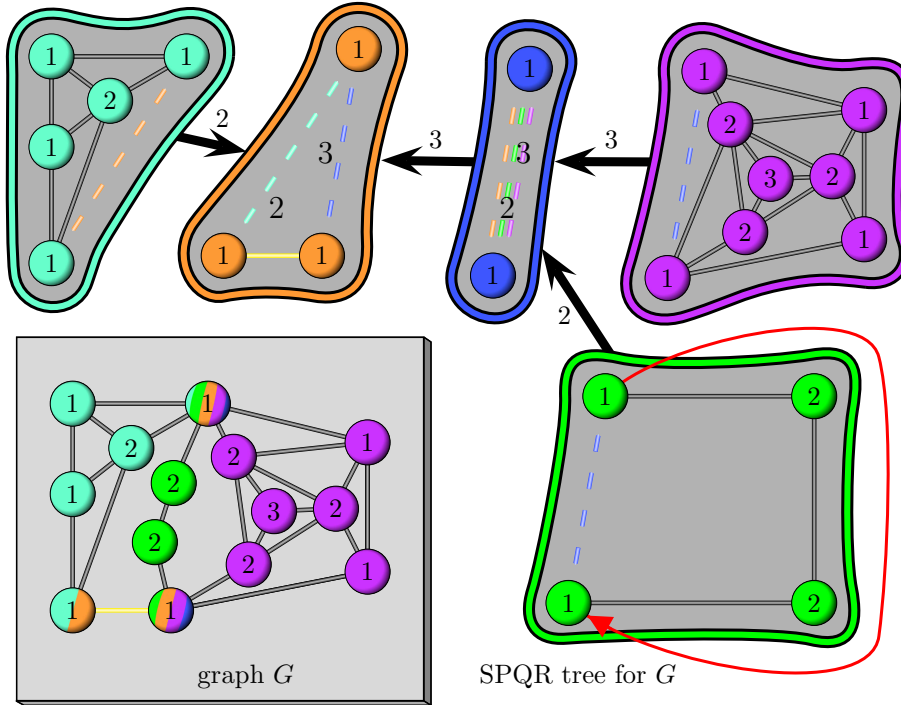


Figure 4.5.1: Optimal embedding with the yellow edge outside with respect to the graph shown in Fig. 3.2.3. The numbers of the vertices and of the virtual edges are extended peeling numbers. By extending the split components by the red directed edge, the peeling numbers of every vertex is the same in the split components and in G .

4.6 General Graphs

Given a planar graph $G = (V, E)$ with weight function r , we can determine the extended peeling number for each connected component separately. Thus, let us assume in the following that all graphs considered are connected.

Let $e \in E$ be an edge contained in a biconnected component B . Let us assume for the moment that, for each B -attached component H , for each cutpoint u of G part of H , and for each $E^+ \in (\emptyset, \{(u, u)\})$, we know the extended peeling number of a vertex-constrained optimal embedding for $(H, r, E^+, \mathcal{R}_H(B))$. Using the algorithm described at the end of Section 4.4 we can then determine the increased weight function r' with respect to (G, B, r) and use the algorithm to find an edge-constrained optimal embedding for $(B, r', \emptyset, e \cup \{e\})$ to obtain an edge-constrained optimal embedding for $(G, 0, \emptyset, e \cup \{e\})$. By iterating over all edges of G an optimal embedding can be determined. Thus, it remains to show how all necessary vertex-constrained optimal embeddings can be found.

First of all, construct a block-cutpoint tree $T = (W, F)$ for G in linear time (Lemma 3.2.7) and choose some biconnected component r as root of T . Let us define the *subtree (supertree) component* of a biconnected component B of G as the subgraph of G induced by the vertices in the biconnected components being a (no) descendant of B in T .

To find all vertex-constrained optimal embeddings of a graph G , let us traverse a block-cutpoint tree $T = (W, F)$ of G with a biconnected component B^* as root first bottom-up and then top-down. During the bottom-up (top-down) traversal we compute, for each biconnected component $B \neq B^*$ with a parent v , a vertex-constrained optimal embedding of the subtree (supertree) component of B with v outside as follows. For each B , define B' as B during the bottom-up traversal and as the *grandparent*—i.e., the parent of the parent—of B during the top-down traversal. Note that, for the subtree (supertree) component C of B , we already know the extended peeling numbers of each B' -attached component in C .

Thus, after the two traversals, we have the increased weight function r' of (C, B, r) for each component B . Moreover, a vertex-constrained optimal embedding for $(B, r', \emptyset, \{v\})$ can be found by iterating over each edge e of B incident to v and computing an edge-constrained optimal embedding for $(B, r', \emptyset, e \cup \{e\})$. In the end, we obtain for each cutpoint v a vertex-constrained optimal embedding of all v -attached components in G in quadratic time.

Theorem 4.6.1. *Given a planar graph G , one can find an optimal embedding and the outerplanarity index of G in quadratic time.*

For an approximate embedding of a graph G with weight function r , the idea is so far to fix an arbitrary edge e and to search for an edge-constrained optimal embedding with e outside. Unfortunately, we cannot use the same approach as for an optimal embedding since determining a vertex-constrained optimal embedding of a single big biconnected component may take too much time. Therefore we take a rooted block-cutpoint tree T , compute only bottom-up the two embeddings φ' and φ'' of B as defined below for each biconnected component B with parent v and subtree component $C = (V_C, E_C)$ in T , and determine the smaller extended peeling number of the two extended peelings $\mathcal{P}_{\varphi'}$ of $(B, \varphi', r', \emptyset)$ and $\mathcal{P}_{\varphi''}$ of $(B, \varphi'', r', \emptyset)$, where r' is the increased weight function of (C, B, r) computed from the extended peeling numbers obtained from the recursive calls. Moreover, take the embedding $\varphi^\nabla \in \{\varphi', \varphi''\}$ such that $\mathcal{P}_{\varphi^\nabla}(B)$ is as small as possible, extend φ^∇ in constant time to an embedding φ of C by using the results of the recursive calls and return φ .

For some edge e incident to v , take φ' as an edge-constrained optimal embedding of B with e outside. If B has not a *grandchild*—i.e., a child of a child—define $\varphi'' = \varphi'$ and skip to the next paragraph. Otherwise, let D be a grandchild of B such that the algorithm has recursively obtained a largest extended peeling number on D among all grandchildren of B . In addition, let v^* be the parent of D . If a vertex-constrained optimal embedding of B with v and v^* outside exists, define φ'' as such an embedding. Otherwise define $\varphi'' = \varphi'$ for a simpler description in the following.

By Lemma 4.5.1 and 4.5.2, we can find φ' and φ'' in a time linear in the number of vertices of B . We now show by induction over the height of a vertex B in T that this bottom-up traversal finds a *vertex-constrained 2-approximative embedding* φ of C with v outside, i.e., $\mathcal{P}(C)$ obtained from the extended peeling

\mathcal{P} of $(C, \varphi, r, \emptyset)$ is at most twice the extended peeling number of C obtained from $(C, \varphi^*, r, \emptyset)$ for every other embedding φ^* of C with v outside. Let φ^B , φ^C , and φ^D be vertex-constrained optimal embeddings for $(B, r, \emptyset, \{v\})$, for $(C, r, \emptyset, \{v\})$, and for $(D, r, \emptyset, \{v\})$, respectively, and let $p_B^{\text{OPT}} = \mathcal{P}_B(B)$, $p_C^{\text{OPT}} = \mathcal{P}_C(C)$ and $p_D^{\text{OPT}} = \mathcal{P}_D(D)$, where \mathcal{P}_B , \mathcal{P}_C , and \mathcal{P}_D is the extended peeling of $(B, \varphi^B, r, \emptyset)$, $(C, \varphi^C, r, \emptyset)$, and $(D, \varphi^D, r, \emptyset)$, respectively. In addition, take p_B , p_C , and p_D as the extended peeling numbers obtained from our algorithm for B , C , and D , respectively. Although we enforce one arbitrary edge incident to v to be outside, $p_B - p_B^{\text{OPT}} \leq 1$ since after a first peeling step the outer face contains each face incident to v . If a vertex of B has the largest extended peeling number among all vertices in C , i.e., $p_C = p_B$ —this case is also our induction basis, where no grandchild of B exists—the embedding obtained for C is vertex-constrained 2-approximative since $p_C = p_B \leq p_B^{\text{OPT}} + 1 \leq 2p_B^{\text{OPT}} \leq 2p_C^{\text{OPT}}$. Otherwise, a vertex in some subtree component H —possibly $H = D$ —of some grandchild of B has the extended peeling number p_C . Take v' as the inner vertex of the B - H -path and p_H as the extended peeling number obtained from our algorithm for input H . Let φ^H be a constrained optimal embedding for $(H, r, \emptyset, \{v\})$, and let $p_H^{\text{OPT}} = \mathcal{P}_H(H)$, where \mathcal{P}_H is the extended peeling of $(H, \varphi^H, r, \emptyset)$. By construction of our algorithm $p_D \geq p_H$ holds. Moreover, by induction $p_H \leq 2p_H^{\text{OPT}}$ and $p_D \leq 2p_D^{\text{OPT}}$. Define $q = p_C^{\text{OPT}} - p_H^{\text{OPT}}$. Intuitively, we need q peeling steps in φ^C until the first vertex of H can be removed; thus, our algorithm needs at most $q + 1$ peeling steps until a first vertex of H can be removed. Let us consider 3 cases:

$$\mathbf{q} > \mathbf{0} : p_C \leq p_H + (q + 1) \leq 2p_H^{\text{OPT}} + (q + 1) \leq 2p_H^{\text{OPT}} + 2q = 2p_C^{\text{OPT}}.$$

$$\mathbf{q} = \mathbf{0} \wedge \mathbf{p}_H < \mathbf{2p}_H^{\text{OPT}} : p_C \leq p_H + (q + 1) = p_H + 1 \leq 2p_H^{\text{OPT}} = 2p_C^{\text{OPT}}.$$

$$\mathbf{q} = \mathbf{0} \wedge \mathbf{p}_H \geq \mathbf{2p}_H^{\text{OPT}} : p_D^{\text{OPT}} \geq p_D/2 \geq p_H/2 \geq p_H^{\text{OPT}} = p_C^{\text{OPT}}, \text{ so that } v^*,$$

which is the parent of D , is outside with respect to some optimal embedding of C with v outside.

In the first and second case φ' and in the last case φ'' leads to a vertex-constrained 2-approximative embedding φ . Similarly as in the proof of Corollary 4.5.3, we can conclude that a vertex-constrained 2-approximative embedding is also a non vertex-constrained 4-approximative embedding.

Theorem 4.6.2. *For a planar graph G , a 4-approximative embedding of G and a 4-approximation of the outerplanarity index of G can be found in linear time.*

Chapter 5

Generalization of Trees

5.1 A Sketch of Monadic Second-Order Logic

Many properties can be expressed in formulas of a suitable logic by *predicates*, i.e., functions whose image set is $\{\text{true}, \text{false}\}$. For this purpose, we define a *labeled graph* as graph $G = (V, E)$ with $i \in \mathbb{N}$ functions $X_1^L, \dots, X_i^L : V \rightarrow \{\text{true}, \text{false}\}$ to denote that, for all $1 \leq j \leq i$, all $v \in V$ with $X_j^L(v) = \text{true}$ are labeled with j . Additionally, we call X_1^L, \dots, X_i^L the *labels* of G and $X_j = \{v \mid X_j(v) = \text{true}\}$ the *set for the label X_j^L* . We also write G^{X_1, \dots, X_j} to denote explicitly that X_1, \dots, X_j are the sets for the labels of G .

To simplify matters, we consider in the following only logics over (labeled) graphs; this means the universe consists of the vertex set of a graph G .

Definition 5.1.1 (first-order logic, free and bound variable). For each formula φ in first-order logic as defined below, let us call each variable in φ a *free variable* unless with some rules below, we explicitly call it bounded. The following expressions are formulas in *first-order logic* over a (labeled) graph $G = (V, E)$.

- Expressions $E(x, y)$ and $=(x, y)$ for variables x and y are formulas in first-order logic.
- If X^L is a label of G , $X^L(x)$ is a formula in first-order logic.
- If φ is a formula in first-order logic with free variables x_1, \dots, x_i , for each $1 \leq j \leq i$, $\forall x_j : \varphi$ and $\exists x_j : \varphi$ are formulas in first-order logic. After introducing such a quantifier \forall or \exists for x_j , all occurrences of x_j that are free in φ are now called *bounded* (by the new introduced quantifier). The remaining variables $x_1, \dots, x_{j-1}, x_{j+1}, \dots, x_i$ are still free in φ .
- If φ_1 and φ_2 are formulas in first-order logic, then $\neg\varphi_1$ and $\varphi_1 \otimes \varphi_2$ with \otimes being a *logical connective* such as, e.g., $\wedge, \vee, \rightarrow$, and \leftrightarrow are also formulas in first-order logic.

Note that a vertex of the universe is the only value represented by a variable—usually denoted by lowercase letters such as x and y —in a formula of first-order logic. For an easier notation, if X^L is a label and X the set for X^L ,

we usually write $x \in X$ instead of $X^L(x)$. If a predicate P is equal to a formula in logic, the free variables x_1, \dots, x_i ($i \in \mathbb{N}_0$) of the formula are exactly the arguments of the predicate. A predicate is usually denoted by one or several capital letters followed by its arguments, e.g., $P(x_1, \dots, x_i)$. Moreover, different labels imply different predicates. The number of arguments of a predicate is its *arity*. A predicate of arity zero is, e.g., $P \equiv \forall x : x \in X' \rightarrow x \in X$. P is true if and only if $X' \subseteq X$. A predicate $Q(x)$ of arity one can be, e.g., $Q(x) \equiv \neg(x \in X)$, which is true if and only if $x \notin X$. From now on, we use \subseteq and \notin as an abbreviated notation expressing that a vertex set is a subset or equal to another vertex set and that a vertex is not part of a vertex set, respectively.

Observe that a predicate P in first-order logic with exactly one free variable x —also called a *monadic predicate*—can always be expressed as $x \in X$ since we only have to choose X as a subset of the universe such that it contains all elements for which $P(x)$ is true. We next define an extension of first-order logic that allows quantifications over monadic predicates, in other words, quantifications over sets. Therefore, we introduce new variables—usually denoted by uppercase letters such as X and Y —to represent subsets of the universe. In order to distinguish between variables for single elements and for sets, let us call the first kind *individual variables* and the latter kind *set variables*.

Definition 5.1.2 (monadic second-order logic (MSOL)). Formulas in *monadic second-order logic* (MSOL) are defined recursively as described in the next paragraph. First of all, let us call a variable in a formula of MSOL *free* if it is not bounded explicitly by a quantifier with the rules below or the rules for first-order logic.

Each formula in first-order logic is also a formula in MSOL. If φ is a formula in MSOL with a monadic predicate $P(x) \equiv x \in X$, then—in contrast to first-order logic— X can be a set variable. Moreover, if X is a free (individual- or set) variable in a formula φ in MSOL, then $\forall X : \varphi$ and $\exists X : \varphi$ are also formulas in MSOL. In this case, all occurrences of X being free in φ are called *bounded* (by the new introduced quantifier). Additionally, if φ_1 and φ_2 are formulas in MSOL, then $\neg\varphi_1$ and $\varphi_1 \otimes \varphi_2$ with \otimes being a logical connective are also formulas in MSOL.

For a (labeled) graph $G = (V, E)$, for a predicate $P(X_1, \dots, X_g, x_1, \dots, x_h)$, for vertex sets $V_1, \dots, V_g \subseteq V$, and for vertices $v_1, \dots, v_h \in V$, we say G *models* $P(V_1, \dots, V_g, v_1, \dots, v_h)$ and write $G \models P(V_1, \dots, V_g, v_1, \dots, v_h)$ to denote that $P(X_1, \dots, X_g, x_1, \dots, x_h)$ is true if

- the universe for the quantification is V ,
- for all $1 \leq i \leq g, 1 \leq j \leq h$, X_i and x_j represents V_i and v_j , respectively,
- for variables y, y' of P and for a set variable or the set for a label Y of P ,
 - $=(y, y')$ is true if and only if y and y' represent the same vertex,
 - $E(y, y')$ is true if and only if y and y' represent vertices u and v , respectively, with $\{u, v\} \in E$, and
 - $y \in Y$ is true if and only if y represents a vertex that belongs to the set represented by Y .

Next we see some examples how we can use logic over graphs to express graph problems. By the formalism of first-order logic we next show the property that a vertex set X of a graph $G = (V, E)$ is independent:

$$IS \equiv \forall x, y \in V : x \notin X \vee y \notin X \vee \neg E(x, y).$$

In other words, $G^X \models IS$ if and only if X is an independent set in G . In MSOL, we can take X as a free set variable. Then $G \models IS(X)$ if and only if X represents an independent set in G .

Definition 5.1.3 ((ℓ) -vertex-disjoint path problem). Given a graph $G = (V, E)$ and, for some $\ell \in \mathbb{N}$, ℓ pairs $(s_1, t_1), \dots, (s_\ell, t_\ell)$ of vertices in V with $|\{s_1, t_1, \dots, s_\ell, t_\ell\}| = 2\ell$, in the *vertex-disjoint path problem* (VDPP) the goal is to find—if possible— ℓ pairwise vertex-disjoint paths P_1, \dots, P_ℓ such that, for all $i \in \{1 \dots \ell\}$, s_i and t_i are the endpoints of P_i .

In the *ℓ -vertex-disjoint path problem* (ℓ -VDPP)—in contrast to the VDPP— ℓ is given as a fixed constant and is not part of the input.

As another example, we consider the *decision version* of the ℓ -vertex-disjoint path problem, i.e., we next search for a predicate that gets true if and only if the ℓ -vertex-disjoint path problem has a solution. First, we introduce an auxiliary predicate. For a graph $G = (V, E)$, vertices $s, t \in V$, and a set $X \subseteq V$, the next predicate in MSOL is modeled by a graph G if and only if s and t are part of the same connected component in $G[X]$.

$$\begin{aligned} \text{JOINS}(X, s, t) &\equiv s \in X \wedge t \in X \wedge \forall X' \subseteq X : \\ &(s \in X' \wedge t \notin X') \rightarrow (\exists x, y \in V : E(x, y) \wedge x \in X' \wedge y \in X \setminus X') \end{aligned}$$

The idea behind $\text{JOINS}(X, s, t)$ is that, for every partition X' and X'' of X with $s \in X'$ and $t \in X''$, we can find an edge in G connecting a vertex in X' with a vertex in $X \setminus X'$. Moreover, for a graph $G = (V, E)$ and $\ell \in \mathbb{N}$ pairs (s_i, t_i) of vertices in V ($1 \leq i \leq \ell$), we can test by the following predicate if there exist ℓ pairwise vertex-disjoint paths P_i with endpoints s_i and t_i . Note that it is enough to show that there exist k pairwise vertex-disjoint sets X_i with $G[X_i]$ containing a path from s_i to t_i .

$$\begin{aligned} \text{PATH}_\ell(s_1, t_1, \dots, s_\ell, t_\ell) &\equiv \exists X_1, \dots, X_\ell : \\ &\bigwedge_{i=1}^{\ell} \left(\text{JOINS}(X_i, s_i, t_i) \wedge \left(\bigwedge_{j=1}^{\ell} (\forall v \in V : v \notin X_i \vee v \notin X_j) \right) \right) \end{aligned}$$

Thatcher and Wright [88] showed that one can associate with each fixed predicate $P(X_1, \dots, X_g, x_1, \dots, x_h)$ ($g, h \in \mathbb{N} \cup \{0\}$) in MSOL a so-called *deterministic bottom-up tree automaton* A_P such that, for each binary tree $T^+ = (V, E)$ with a constant number of labels, each choice of vertex sets $V_1, \dots, V_g \subseteq V$ and each choice of vertices $v_1, \dots, v_h \in V$, $T^+ \models P(V_1, \dots, V_g, v_1, \dots, v_h)$ if and only if after replacing X_i by V_i and x_j by v_j , for all $1 \leq i \leq g, 1 \leq j \leq h$, A_P accepts T^+ . Moreover, one can show that a simulation of A_P can test whether A_P accepts in a time linear in the size of T^+ .

Theorem 5.1.4. *For each fixed predicate in MSOL, there is a linear-time algorithm that tests if a binary (labeled) tree models the predicate.*

Courcelle [25] has shown that a generalization from trees to graphs with bounded treewidth is possible by an approach that can be sketched as follows: Given a fixed predicate $P(X_1, \dots, X_g, x_1, \dots, x_h)$ ($g, h \in \mathbb{N}_0$) in MSOL as well as a graph G and a binary tree decomposition (T, B) for G with (T, B) having a width bounded by a constant, one can construct a predicate P^+ by modifying $P(X_1, \dots, X_g, x_1, \dots, x_h)$ and a labeled tree T^+ from T by adding some labels to T —mainly for encoding the vertex sets of the bags of (T, B) and $X_1, \dots, X_g, x_1, \dots, x_h$ —such that $G \models P(X_1, \dots, X_g, x_1, \dots, x_h)$ if and only if $T^+ \models P^+$. Combining these ideas with the last theorem we obtain Courcelle’s fundamental theorem:

Theorem 5.1.5. *For a graph class \mathcal{G} with bounded treewidth, a property formulated in MSOL can be checked on each graph $G \in \mathcal{G}$ in linear time.*

Recall that in the first chapter we saw how to find a maximum independent set on a graph G of bounded treewidth in linear time. In particular, we can determine in linear time whether an independent set of size at least $\ell \in \mathbb{N}$ exists in G . The same result can be obtained by the last theorem using the predicate $\text{IS}(X) \wedge \text{SIZE}_{\geq \ell}(X)$, where

$$\text{SIZE}_{\geq \ell}(X) \equiv \exists x_1 \dots, x_\ell \in X : \bigwedge_{1 \leq i < j \leq \ell} \neg(x_i = x_j).$$

Additionally, for each graph class of bounded treewidth, we can solve the decision version of the ℓ -vertex-disjoint path problem in linear time. Given an algorithm for the decision version of the ℓ -vertex-disjoint path problem with a running time $T(n, m) = \Omega(n + m)$, one can iterate over every edge e and test if there are ℓ -vertex-disjoint paths in the graph $(V, E \setminus \{e\})$. If this is the case, remove the edge from the given graph and continue the iteration on the graph obtained. At the end exactly the edges of ℓ vertex-disjoint paths remain. By ℓ DFS one can finally construct the ℓ paths. Consequently, we can construct ℓ -vertex-disjoint paths in $O(n + mT(n, m))$ time.

However, there are also problems for which no predicates in MSOL are known or do exist. As we see in Section 5.3, certain so-called convex coloring problems are NP-hard even on trees, i.e., using the common believed assumption $P \neq NP$ and Theorem 5.1.5 we can conclude that no predicate in MSOL exists for these convex coloring problems. (Even if $P = NP$, no predicate in MSOL is known currently for these problems.)

As we see in the following sections of this chapter, for problems where no predicate in MSOL is known, we can also generalize ideas on trees to graphs of bounded treewidth.

5.2 Three Convex Coloring Variants

Before we can define convex colorings, we need some further definitions. A *colored graph* (G, C) is a tuple consisting of a graph G and a *coloring* C of G , i.e., a function assigning each vertex v to a color that is either 0 or a so-called *real color*. A vertex colored with 0 is also called *uncolored*. A coloring is an

(a, b) -coloring if the color set used for coloring the vertices contains at most a real colors and if each real color is used to color at most b vertices. Two equal-colored vertices u and v in a colored graph (G, C) are C -connected if there is a path from u to v whose vertices are all colored by C with the color of u and v . A coloring C is called *convex* if all pairs of vertices colored with the same real color are C -connected. For a colored graph (G, C_1) , another arbitrary coloring C_2 of G is also called a *recoloring* of (G, C_1) . We then say that C_1 is the *initial coloring* of G and that C_2 *recolors* or *uncolors* a vertex v of G if $C_2(v) \neq C_1(v)$ and $C_2(v) = 0$, respectively. The *cost* of a recoloring C_2 of a colored graph (G, C_1) with $G = (V, E)$ is $\sum_{v \in V: 0 \neq C_1(v) \neq C_2(v)} w(v)$, where $w(v)$ denotes the weight of v with $w(v) = 1$ in the case of an unweighted graph. This means that we have to pay for recoloring or uncoloring a real-colored vertex, but not for recoloring an uncolored vertex.

In the *minimum convex recoloring problem* (MCRP) we are given a colored graph possibly with a positive weight-function w , and we search for a convex recoloring with minimal cost. The MCRP describes a basic problem in graph theory with different applications in practice: a first systematic study of the MCRP on trees is from Moran and Snir [68] and was motivated by applications in biology. Further applications are so-called multicast communications in optical wavelength division multiplexing networks; see, e.g., [22] for a short discussion of these applications. Here we focus on the MCRP as a special kind of routing problem. Suppose we are given a telecommunication or transportation network described by a graph whose vertices represent routers. Moreover, assume that each router can establish a connection between itself and an arbitrary set of adjacent routers. Then routers of the same initial color could represent clients that want to be connected by the other routers to communicate with each other or to exchange data or commodities. More precisely, connecting clients of the same color means finding a connected subgraph of the network containing all the clients, where the subgraphs for clients of different colors should be disjoint. If we cannot establish a connection between all the clients, we want to give up connecting as few clients to the other clients of the same color as possible in the unweighted case ($w(v) = 1$ for all vertices v) or to give up a set of clients with minimal total weight in the weighted case. Hence, the problem reduces to the MCRP, where the initially uncolored vertices that are recolored by a color c represent exactly the routers used to connect clients of color c . Fig. 5.2.1 shows an example instance and Fig. 5.2.2 an optimal solution for it. The case in which routers can connect a constant number of disjoint sets of adjacent routers can be handled by duplicating the vertices representing a router.

We also study a relaxed version of the problem above that we call the *minimum restricted convex recoloring problem* (MRRP). In this problem we ask for a convex recoloring C' that does not recolor any real-colored vertex with a different real color. In practice clients often cannot be used for routing connections for other clients so that a clear distinction between clients and routers should be made. This corresponds to the MRRP, where a client that cannot be connected to the other clients of the same color may only be uncolored. See Fig. 5.2.3 for an example instance and Fig. 5.2.4 for an optimal solution.

In addition, we consider a variant of the MCRP, where we search for a convex recoloring, but assign a cost to each color c . We have to pay the cost for color c if

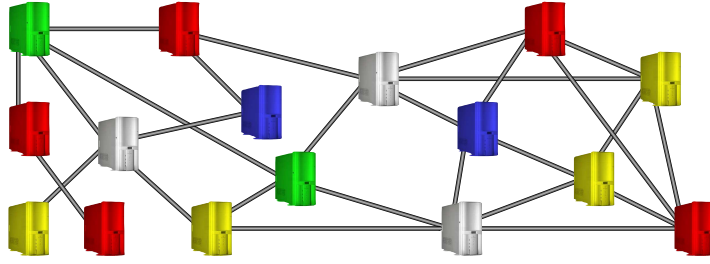


Figure 5.2.1: An instance of a colored graph, where each vertex is a router.

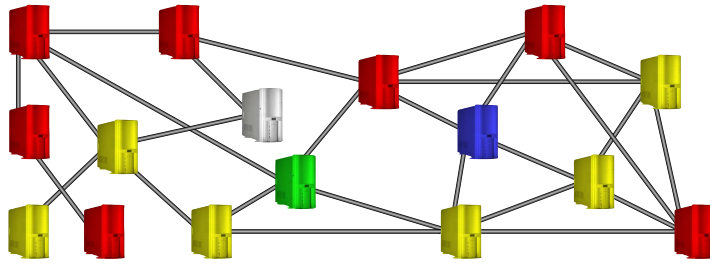


Figure 5.2.2: An optimal convex recoloring of Fig. 5.2.1.

at least one vertex of color c is recolored. Let us call this recoloring problem the *minimum block recoloring problem* (MBRP). In an unweighted version we assign cost 1 to each color. The MBRP is useful if it is not enough in an application to connect only a proper subset of clients (of the same color) that want to be connected. See also Fig. 5.2.5.

The MCRP, the MRRP, and the MBRP can also be considered as generalizations of the vertex-disjoint paths problem. Indeed any algorithm solving one of the three recoloring problems on $(\infty, 2)$ -colorings to optimality can solve the VDPP, too. Given an algorithm for the MBRP one can also solve the problem of connecting a subset of a given set of weighted vertex pairs $(s_1, t_1), \dots, (s_l, t_l)$ by disjoint paths such that the sum of the weights of the connected pairs is maximized.

Given an instance of the VDPP, it is NP-hard to determine the maximum number of pairs that can be connected by pairwise disjoint paths as shown by Knuth [59] and Lynch [63]. Moreover, Andrews and Zhang [3] have shown that this problem can not be approximated on N -vertex graphs in polynomial time within a ratio of size $\leq \log^{1/3-\epsilon} N$ unless NP is contained in the so-called complexity class $ZPTIME(n^{\text{poly} \log(n)})$. The NP-hardness of the unweighted MCRP, MRRP, and MBRP follows directly from Knuth's and Lynch's result. However, non-approximability results for the recoloring problems do not follow from the result of Andrews et al. since the recoloring problems are *minimization problems*, i.e., we want to minimize the number of colors that can not be connected. Moran and Snir [68] showed that the MCRP on (∞, ∞) -colorings remains NP-hard on trees, and the same is true for the MRRP, as follows implicitly from Moran's and Snir's result [68] concerning leaf-colored trees. Moreover, Snir [85] presented a

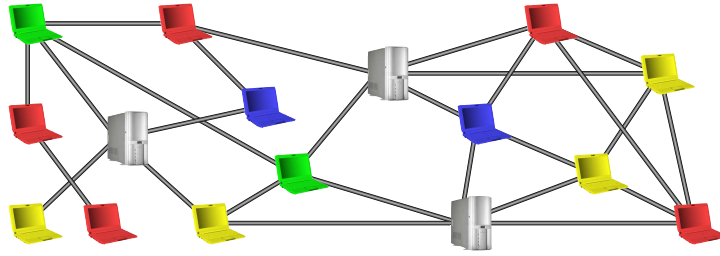


Figure 5.2.3: The colored graph of Fig. 5.2.1 with routers and laptops (clients).

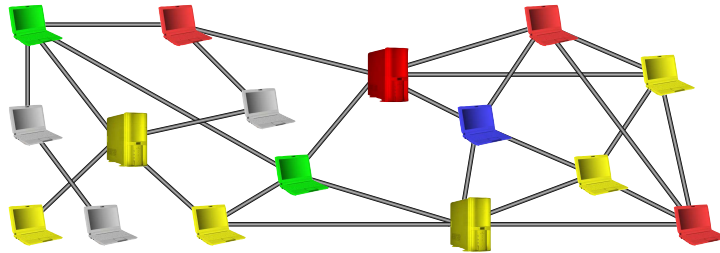


Figure 5.2.4: An optimal restricted convex recoloring of figure Fig. 5.2.3.

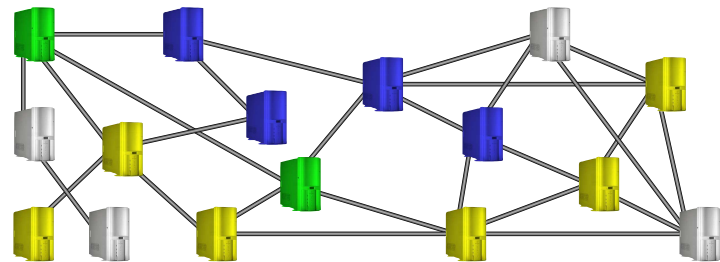


Figure 5.2.5: An optimal block recoloring of Fig. 5.2.1.

polynomial-time 2-approximation algorithm for the weighted MCRP on paths and a polynomial-time 3-approximation algorithm for the weighted MCRP on trees also published in [69]. Bar-Yehuda, Feldman, and Rawitz [9] could improve the approximation ratio on trees to $2 + \epsilon$. Their algorithm is generalized in Section 5.7.

In contrast to the work of Moran, Snir as well as of Bar-Yehuda et al., we consider initial (a, b) -colorings with a and b being different from ∞ . In addition, we also consider graphs of bounded treewidth instead of only trees.

Surprisingly, the three variants of the recoloring problem all have different complexities on graphs of bounded treewidth, as we prove in the following four sections. Table 5.2.6 summarizes the results on graphs of bounded treewidth shown in this chapter. We will see that the MCRP is NP-hard even on unweighted trees initially colored with $(\infty, 2)$ -colorings, whereas the MRRP can be solved in polynomial time for the more general $(\infty, 3)$ -colorings as input colorings even on weighted graphs of bounded treewidth. We also observe the NP-hardness of the MRRP on trees colored with $(\infty, 4)$ -colorings. Moreover,

we study a polynomial-time algorithm for the MBRP on weighted graphs of bounded treewidth for general colorings, i.e., (∞, ∞) -colorings.

Beside the results on graphs of bounded treewidth we will see that, for some $c > 0$, the unweighted versions of the recoloring problems on an n -vertex graph cannot be approximated in polynomial time within an approximation ratio of $c \ln \ln n$ unless $P = NP$ even if the initial colorings are restricted to $(\infty, 2)$ -colorings. As a consequence of this result, $P \neq NP$ implies that there is no good polynomial-time approximation possible for the following problem: Given pairs of vertices, find the minimal ℓ such that all except ℓ pairs can be connected by disjoint paths.

	$a \in O(\log n)$	$a = \infty$	
		$b = 2, 3$	$b \geq 4$
MCRP	$\in P$	NP-hard	NP-hard
MRRP	$\in P$	$\in P$	NP-hard
MBRP	$\in P$	$\in P$	$\in P$

Table 5.2.6: Complexity results for graphs with treewidth $O(1)$ on (a, b) -colorings.

5.3 Hardness Results for Convex Coloring

In this section, we first show an inapproximability result for all three convex recoloring variants on general graphs by a reduction from the following problem.

Definition 5.3.1 (set cover). Given a tuple (U, \mathcal{F}) , where U is a finite set called *universe* and \mathcal{F} is a collection of sets, a *set cover* for (U, \mathcal{F}) is a collection $\mathcal{F}^* \subseteq \mathcal{F}$ such that $\bigcup_{Z \in \mathcal{F}^*} Z = U$. $|\mathcal{F}^*|$ is then called the *size* of the set-cover and a *minimum set cover* is a set cover of minimum size.

Lemma 5.3.2. *Given an unweighted n -vertex graph with an $(\infty, 2)$ -coloring, no polynomial-time algorithm for the MCRP, the MRRP or the MBRP has an approximation ratio $c \ln \ln n$ (for some constant $c > 0$) unless $P = NP$.*

Proof. Alon, Moshkovitz, and Safra [2] showed that, for an appropriate chosen constant $c > 0$, there is no polynomial-time approximation algorithm of ratio $c \ln |U|$ for the minimum set-cover problem with universe U unless $P = NP$. In the following we prove the lemma by a reduction from the minimum set-cover problem. Since one can easily test in polynomial time if a set-cover instance has a solution, we assume in the following that we are given a set-cover instance (U, \mathcal{F}) that has a solution. W.l.o.g. we additionally assume that $|U| > 5$, that each element of U is contained in a set z in \mathcal{F} , and that \mathcal{F} contains only sets with cardinality $\leq (|U| - 2)$, i.e., in particular $|\mathcal{F}| \leq 2^{|U|} - |U|$.

We now create a colored graph (G, C) as follows. For each set z in \mathcal{F} , we define a ordering $\sigma_1, \sigma_2, \dots, \sigma_r$ of the elements of z and introduce in G a path consisting of new vertices $z', z_{\sigma_1}, z_{\sigma_2}, \dots, z_{\sigma_r}, z''$ in this order. For each $u \in U$, we add vertices u' and u'' to G . Moreover, for each $z \in \mathcal{F}$ and each $u \in z$, we add edges $\{z_u, u'\}$ and $\{z_u, u''\}$ (see Fig. 5.3.1). For each set $z \in \mathcal{F}$ and each $u \in U$, we define new colors c_z and c_u different from the other colors and color

the vertices z' and z'' with c_z and the vertices u' and u'' with color c_u . All other vertices remain uncolored.

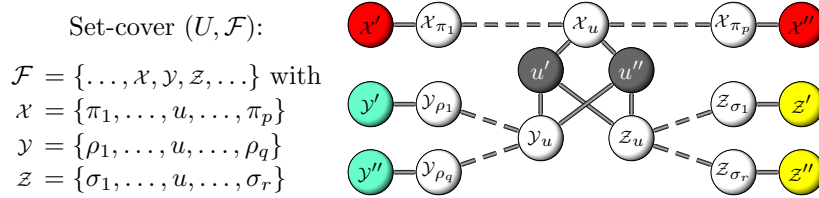


Figure 5.3.1: Reduction from minimum set cover to a recoloring problem.

We next show that each set cover for (U, \mathcal{F}) of size k can be used to construct a convex recoloring of (G, C) with cost k for each of the three coloring problems in polynomial time and vice versa. Since the constructed graph G has $n \leq 2|\mathcal{F}| + (|U| - 2)|\mathcal{F}| + 2|U| = |U| \cdot (|\mathcal{F}| + 2) \leq |U|2^{|\mathcal{F}|} \leq e^{|\mathcal{F}|}$ vertices and because of the non-approximability result for the set-cover problem, for some constant $c > 0$, no polynomial-time algorithm can solve the three recoloring problems with an approximation ratio $c \ln |U| \geq c \ln \ln n$ unless $P = NP$.

Given a set cover $\mathcal{F}^* \subseteq \mathcal{F}$ of size k , we can find a recoloring C' of G with cost k as follows: For each set $z \in \mathcal{F} \setminus \mathcal{F}^*$, we recolor the inner vertices of the paths from z' to z'' with color c_z . For each set $z = \{\sigma_1, \sigma_2, \dots, \sigma_r\} \in \mathcal{F}^*$, we uncolor z'' and color z_{σ_i} with c_{σ_i} for all $1 \leq i \leq r$. Since \mathcal{F}^* is a set cover, for each $u \in U$, there exists a set $z \in \mathcal{F}^*$ with $u \in z$. Therefore, the recoloring guarantees that, for each $u \in U$, the vertices u' and u'' are C' -connected because of the recoloring of z_u with color c_u for some $z \in \mathcal{F}^*$. For each set $z \in \mathcal{F} \setminus \mathcal{F}^*$, the vertices z' and z'' are also C' -connected. Since each remaining color c_z with $z \in \mathcal{F}^*$ is used by C' to color only one vertex, C' is a convex recoloring. The cost for the recoloring C' with respect to the MCRP, MRRP or MBRP is $|\mathcal{F}^*| = k$ since the only real-colored vertices with respect to C that are recolored by C' are the vertices in $\{z'' \mid z \in \mathcal{F}^*\}$.

Let us now assume that there is a convex recoloring C' for the MCRP, MRRP or MBRP with cost k in G . We show that we can find a set cover of size at most k : Let \mathcal{F}_1^* be the collection of all sets $z \in \mathcal{F}$ for which either at least one of z' or z'' is recolored by C' or there are at least two vertices of $\{u' \mid u \in z\} \cup \{u'' \mid u \in z\}$ being colored with c_z by C' . The recoloring cost for all vertices whose recoloring is responsible for inserting a set $z \in \mathcal{F}$ into \mathcal{F}_1^* is of size at least $|\mathcal{F}_1^*|$ even in the case of the MBRP. Let us remove from G the paths introduced for all $z \in \mathcal{F}_1^*$, all vertices u' and u'' with $u \in U$ being contained in a set of \mathcal{F}_1^* , and all edges being incident to at least one removed vertex. Let $U^* \subseteq U$ be the set of elements u for which u' and u'' remain in G . For each set $z \in \mathcal{F}$ with $z \cap U^* \neq \emptyset$, we know that the z' - z'' -path is colored with c_z . Consequently, for no $u \in U^*$, the vertices u' and u'' can be C' -connected. Therefore and since C' is a convex recoloring, for each $u \in U^*$, one of the vertices u' and u'' must be uncolored. This means that we have the additional recoloring cost $|U^*|$. We can easily find a collection $\mathcal{F}_2^* \subseteq \mathcal{F}$ consisting of at most $|U^*|$ sets whose union contains all elements of U^* . Hence $\mathcal{F}_1^* \cup \mathcal{F}_2^*$ is a set cover of (\mathcal{F}, U) of size at most k . \square

Now we restrict the considered graphs to trees, and let the initial colorings using only very few colors. Nevertheless, two of the three convex recoloring problems remain NP-hard.

Theorem 5.3.3. *The MCRP on initial $(\infty, 2)$ -colorings and the MRRP on initial $(\infty, 4)$ -colorings are NP-hard on unweighted trees.*

Proof. For both problems, we use a reduction from 3-SAT. In *3-Satisfiability* (3-SAT) we are given a Boolean formula in 3-CNF—i.e., a formula in conjunctive normal form with at most 3 variables per clause—and we have to find an assignment of the variables such that at least one literal is true in every clause of the formula. Let F be an instance of 3-SAT, i.e., F is a Boolean formula in 3-CNF. Since satisfiability does not change by replacing a clause consisting of only one literal ℓ by the four clauses $\ell \vee z_1 \vee z_2, \ell \vee \bar{z}_1 \vee z_2, \ell \vee z_1 \vee \bar{z}_2$, and $\ell \vee \bar{z}_1 \vee \bar{z}_2$ and by replacing a clause $\ell_1 \vee \ell_2$ by the two clauses $\ell_1 \vee \ell_2 \vee z_3$ and $\ell_1 \vee \ell_2 \vee \bar{z}_3$ with z_1, z_2 , and z_3 being new variables, we assume that each clause in F has exactly three literals. Let n and m be the number of literals in F and the number of clauses of F , respectively. In addition, let r be the minimal number such that each literal in F appears at most r times in F . The following construction of an instance I_{MCRP} for the MCRP is illustrated by Fig. 5.3.2. We construct G by introducing for each variable x a so-called *gadget* G_x consisting of

- an uncolored vertex v_x ,
- leaves $v_x^{L,i}, v_x^{R,i}$ colored with a color c_x^i for each $i \in \{0, 1\}$,
- an edge $\{v_x^{L,i}, v_x\}$ for each $i \in \{0, 1\}$, and
- two internally vertex-disjoint paths of length $r + 1$, one from v_x to $v_x^{R,0}$, and the other from v_x to $v_x^{R,1}$.

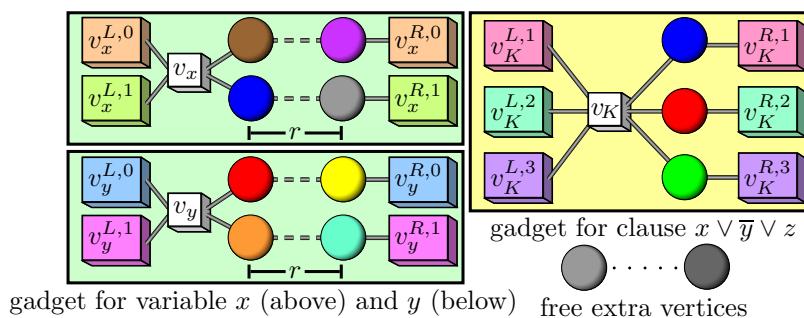


Figure 5.3.2: The figure shows parts of the graph obtained by the reduction from a formula in 3-CNF with a clause $x \vee \bar{y} \vee z$ to MCRP.

Let us call the internal vertices of the v_x - $v_x^{R,1}$ -path the *positive* and those of the v_x - $v_x^{R,0}$ -path the *negative vertices* in the gadget G_x . For each clause K , we introduce a similar *gadget* G_K consisting of

- an uncolored vertex v_K ,

- leaves $v_K^{L,j}, v_K^{R,j}$ colored with a color c_K^j for each $j \in \{1, 2, 3\}$,
- an edge $\{v_K, v_K^{L,j}\}$ for each $j \in \{1, 2, 3\}$, and
- three internally vertex-disjoint paths that have length 2, start in v_K , and end in $v_K^{R,1}, v_K^{R,2}$, and $v_K^{R,3}$, respectively.

In addition, we also introduce $2nr$ extra vertices without any incident edges called the *free vertices* of G . From this forest we obtain a tree T if we simply connect all gadgets and all free vertices by the following three steps. First, add a path of length three consisting of v^1, v^2 , and v^3 into G that all are colored with the same new color. Second, for each variable x , connect v_x to v^1 . Third, for each clause K and each free vertex v , connect v_K and v both to v^3 .

In addition, C colors further vertices of T . For each literal $\ell = x$ or $\ell = \bar{x}$ in a clause K , color one positive (in the case $\ell = x$) or one negative (in the case $\ell = \bar{x}$) vertex of G_x as well as one of the non-leaves adjacent to v_K with a new color $c_{K,\ell}$. If after these colorings there is at least one uncolored positive or negative vertex, we take for each such vertex u a new color c_u and assign it to u as well as to exactly one uncolored free vertex.

Let us call a pair of equal-colored vertices a *literal-clause pair* if it contains a positive or negative vertex. The construction of an instance I_{MRRP} for the MRRP can be obtained from above by the following modification for each vertex v being part of a literal-clause pair. Add two new vertices v^* and v^{**} to T , connect each of them with an edge to v , and move the color c of v to v^* and v^{**} , i.e., color both new vertices with c and uncolor v . The vertices v^* and v^{**} are called an *agent pair* of v . The result is illustrated by Fig. 5.3.3.

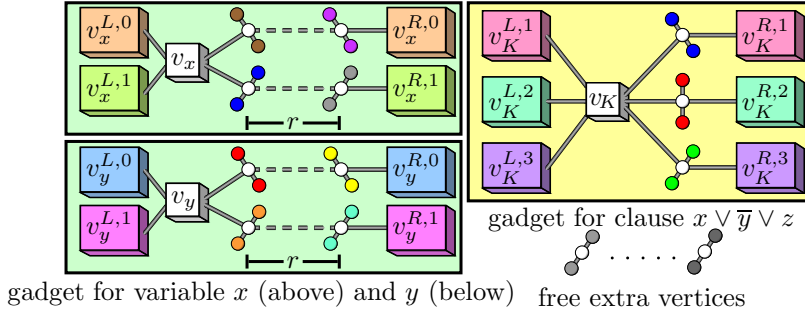


Figure 5.3.3: The figure shows parts of the graph obtained by the reduction from a formula in 3-CNF with a clause $x \vee \bar{y} \vee z$ to MRRP.

It remains to show that the following three statements are equivalent.

1. F is satisfiable, i.e., $F \in 3\text{-SAT}$.
2. I_{MCRP} has a convex recoloring C' with cost $\leq n + 2m + 2nr$.
3. I_{MRRP} has a restricted convex recoloring C'' with cost $\leq n + 2m + 4nr$.

1 \Rightarrow 2: Let B be a satisfying assignment of F . We can construct a convex recoloring C' as follows. For each variable x , if x is true with respect to B ,

we recolor all negative vertices of the gadget x as well as v_x with color c_x^0 , and otherwise we recolor the positive vertices of the gadget x and v_x with color c_x^1 . Moreover, for each clause K , we choose one literal ℓ being true with respect to B . We then search for the vertex u with color $c_{K,\ell}$ adjacent to a leaf $v_K^{R,j}$ for some $j \in \{1, 2, 3\}$ and recolor v_K and u by the color c_K^j . Next, for each literal-clause pair with both vertices still having the same color, we uncolor one of the two. Finally, we uncolor for all $i \in \{0, 1\}$, $j \in \{1, 2, 3\}$, for all variables x and all clauses K the leaves $v_x^{L,i}$ and $v_K^{L,j}$ that do not have the same color as the rules above have assigned to v_x and v_K . Note that for each variable x and each clause K , one pair of initially equal-colored leaves in G_x and G_K are C' -connected. All other colors of C' are used for exactly one vertex. Altogether, we can observe that C' is a convex recoloring that is obtained by un- or recoloring $2m$ leaves of the gadgets for the clauses, n leaves of the gadgets for the variables, and one vertex for each of the $2nr$ literal-clause vertex-pairs. Therefore, C' has cost $n + 2m + 2nr$.

2 \Rightarrow 3: We modify a solution C' for the I_{MCRP} with cost at most $n + 2m + 2nr$ to obtain a solution C'' for the I_{MRRP} with cost at most $n + 2m + 4nr$ by considering each vertex v being part of a literal-clause pair. If v is neither uncolored nor recolored by C' , then C'' colors the agent pair v^* and v^{**} of v as well as v itself with the color $C'(v)$. The agent pair v^* and v^{**} of an uncolored or recolored vertex v are uncolored by C'' .

Note that C' has to recolor one of the four leaves of the gadget G_x for each variable x and two of the six leaves of the gadget G_K for each clause K . The reason for this is that the vertices v_x and v_K can have only one color. Thus, apart from these $(n + 2m)$ leaves, C' can only recolor at most $2nr$ further vertices. Consequently, we have at most $2nr$ vertices whose agent pairs are uncolored, and C'' has cost at most $n + 2m + 4nr$. Finally note that C'' is convex since C' is convex and since we color a vertex v^* and v^{**} with a color c only if their neighbor v is colored with c , too.

3 \Rightarrow 1: Let us consider a convex restricted recoloring C'' of I_{MRRP} with cost at most $n + 2m + 4nr$. Similar to the arguments used in last reduction, C'' needs to uncolor at least $n + 2m$ leaves. Hence, C'' can only uncolor at most $4nr$ non-leaves. Assume that C'' uncolors the three vertices v^1, v^2 , and v^3 (possibly in order to C'' -connect the two agent pairs of a literal-clause pair). Consequently, for each of at least $2nr - 1$ literal-clause pairs, two of the four vertices of its two agent pairs must be uncolored, i.e., at least $4nr - 2$ vertices are to be uncolored by cost at most $4nr - 3$. Since this is not possible, at least one vertex of $\{v^1, v^2, v^3\}$ is not uncolored and, for each literal-clause pair $\{u_1, u_2\}$, either the agent-pair of u_1 or of u_2 has to be uncolored.

In summary, C'' uncolors exactly one agent pair of each literal-clause pair, a leaf of each gadget G_x and 2 leaves of each gadget G_K . Consequently, for each clause K , two leaves in the gadget for K have to be C'' -connected, i.e., there is one non-leaf u adjacent to v_K on the path from $v_K^{L,j}$ to $v_K^{R,j}$ for some $j \in \{1, 2, 3\}$ that is colored with c_K^j by C'' . If the agent pair of u was initially colored by $c_{K,\ell}$, the idea is now to assign to the literal ℓ the value 1 (true) making clause K and finally the whole formula F true. However, this only works if no agent pair initially colored with $c_{K',\bar{\ell}}$ for some clause K' is uncolored. Assume for a contradiction that our construction sets a variable $x = 1$ for some clause K and

$\bar{x} = 1$ for some other clause K' . Since for each literal-clause pair, we can uncolor exactly one of the two agent pairs, G_x contains a positive and a negative vertex, neither of whose agent pair is uncolored by C'' . This is a contradiction to the fact that only one leaf of each G_x is allowed to be uncolored. \square

5.4 MCRP on Trees

The NP-hardness results in Theorem 5.3.3 for the minimum convex recoloring problem (MCRP) on trees and the strongly believed assumption $P \neq NP$ implies that there is no polynomial time algorithm for the MCRP on trees. By the ideas of Section 1.2, we might be tempted to search for a fixed-parameter algorithm with the treewidth as complexity parameter. Unfortunately, even if we use the treewidth as a fixed parameter and restrict the instances such that this parameter k is small, e.g., $k = 1$, we still have to solve the problem on all trees by Lemma 1.3.12; thus, the problem remains NP-hard. Bar-Yehuda, Feldman, and Rawitz [9] observed that the difficult instances on trees are those with many colors, i.e., their idea was to use the number of colors 'somehow' as a complexity parameter and is described in the following. Since in the next section we solve the same problem on graphs of bounded treewidth, the algorithm on trees is only sketched.

Recall that the algorithm for the maximum independent set problem of Section 1.2 traverses a rooted tree bottom-up and, for each vertex v of a tree T , computes the two sizes $|S_v^-|$ and $|S_v^+|$ of biggest independent sets S_v^- and S_v^+ in T_v with $v \in S_v^+ \setminus S_v^-$. As an alternative description, we can sketch this algorithm as follows: For each vertex v , we partition the independent sets in T_v into two collections; one consists of all independent sets containing v and the other consists of all remaining independent sets. However, for an efficient implementation, we compute for each collection S only the size of a biggest independent set in S . Intuitively speaking, this strategy works since we need not to know all independent sets in T_v for determining a biggest independent set in $T_{v'}$ with v' being the parent of v . More exactly, if I_1 and I_2 are independent sets in S_v^+ (or if $I_1, I_2 \in S_v^-$), then for a vertex set I' consisting of no vertices of T_v , $I_1 \cup I'$ is an independent set in T if and only if this is the case for $I_2 \cup I'$.

Before we can describe an algorithm for the MCRP, some further definitions and observations are necessary. For the rest of this section, let us assume that we are given a rooted tree $T = (V, E)$ colored by an initial coloring C . For each $V' \subseteq V$ and for each subtree T^* of T with vertex set V^* , we define $C(V') = \{C(v) \mid v \in V'\} \setminus \{0\}$ and $C(T^*) = C(V^*)$. For a vertex v of T and a color c , let $\text{comp}_c(C, v)$ be the set of all connected components of $T[V - v]$ containing at least one vertex v' with $C(v') = c$. Moreover, let $\text{sep}(C, v)$ be the set of real colors that C uses to color vertices in more than one connected component of $T[V - v]$. An *extension of a (re-)coloring* C_1 for a graph H_1 is a new (re-)coloring C'_1 for a graph $H_2 \supseteq H_1$ with $C'_1(v) = C_1(v)$ for all vertices v of H_1 .

Definition 5.4.1. A recoloring of a colored tree (T, C) is called *effective* if it colors each vertex v of T with a color in $\text{sep}(C, v) \cup \{C(v), 0\}$ and each connected component T' of $T[V \setminus \{v\}]$ only with colors of $C(T') \cup \{0\}$.

Observe that, if there is a convex recoloring C' of (T, C) that colors a vertex v with a color $c \notin \text{sep}(C, v) \cup \{C(v), 0\}$, there is only one connected component T^* in $T[V \setminus \{v\}]$ with vertices colored with c by C , i.e., we can modify C' by uncoloring all vertices of color c that are not part of T^* . Moreover, if there is a convex recoloring C'' of (T, C) that colors a vertex part of a connected component T' in $T[V \setminus \{v\}]$ with a color $c' \notin C(T')$, we can uncolor all vertices that belong to T' and that are colored with c' by C' . The coloring obtained in both cases is still convex, and both modifications do not increase the cost.

Observation 5.4.2. *There is an optimal convex recoloring that is also effective.*

For an efficient algorithm for the MCRP, we also have to partition the convex colorings of T_v for each vertex v into as few collections as possible. Before we can find such a partition, an interesting question is which of the collections \mathcal{C} of colorings of T_v can be characterized somehow easily and consist only of colorings with the following property: For each pair of colorings C' and C'' of \mathcal{C} , C' and C'' can be extended in the same way to a convex recoloring of the whole tree T . Two colorings C_1 and C_2 for a graph H_1 can be *extended in the same way* if, for each extension C'_1 of C_1 and C'_2 of C_2 , at least one of the three following conditions holds: $C'_1(u) \neq C'_2(u)$ for a vertex u not in H_1 or C'_1 and C'_2 are both convex or C'_1 and C'_2 are both not convex. One can easily observe that a possible choice of \mathcal{C} is to take all colorings that color v with the same color, say c , and that use the same set Z of colors for the vertices in T_v obtained through the intersection with the colors in $\text{sep}(C, v) \setminus \{c, 0\}$.

Our algorithm for the MCRP considers the vertices of $T = (V, E)$ in a bottom-up traversal and, for each vertex $v \in V$, the algorithm partitions the convex colorings of T_v such that two convex colorings belong to the same set of the partition if and only if they have the same characteristic. A *characteristic* for a vertex v is a tuple $\mathcal{Q} = (c, Z)$, where $c \in \text{sep}(C, v) \cup \{C(v), 0\}$ and where $Z \subseteq \text{sep}(C, v) \setminus \{c\}$. For obtaining a convex recoloring, the colors in Z are not allowed to be used for the vertices of T not belonging to T_v . Therefore, we call Z in the following the *forbidden colors* of \mathcal{Q} . Moreover, we say that a characteristic (c, Z) *represents* the collection \mathcal{C} of all convex colorings C' of T_v with the following properties: C' is an effective recoloring that colors v with c and that uses exactly the colors $(C(T_v) \setminus \text{sep}(C, v)) \cup \{c, 0\} \cup Z$. For a characteristic (c, Z) , we define the *cost* for (c, Z) as the minimum cost of a convex coloring part of the collection represented by (c, Z) . Alternatively, the reader can consider a characteristic (c, Z) as one effective convex recoloring C' of (T_v, C) such that C' has minimal cost under all effective convex recolorings that color v with c and that use exactly the colors in $(C(T_v) \setminus \text{sep}(C, v)) \cup \{c, 0\} \cup Z$. The cost of a characteristic is then defined as the cost of the recoloring identified with it.

Note that the cost of an optimal solution for the MCRP is exactly the minimum cost of a characteristic at the root of T . For an efficient implementation of the bottom-up traversal—similar to MIS—we compute only the cost for each characteristic and not the collections represented by the characteristics. Similar to the MIS, a subsequent top-down traversal gives us an optimal convex recoloring.

Thus, it remains to show how to compute, for each vertex v of T , the cost for each characteristic (c, Z) at v stored in a variable $\text{cost}_v(c, Z)$ at the end of the algorithm. If v is a leaf of T , the fact that we search only for effective convex

recolorings implies that the only characteristics of v are $(C(v), \emptyset)$ as well as $(0, \emptyset)$, and we store $cost_v(C(v), \emptyset) = 0$ and $cost_v(0, \emptyset) = w(v)$. If, for some $\ell \in \mathbb{N}$, v has children v_1, \dots, v_ℓ , the algorithm initially sets $cost_v(c', Z') = \infty$ for all possible characteristics (c', Z') of v , iterates over all colors $c \in sep(C, v) \cup \{C(v), 0\}$ as possible colors for v , and executes the steps described in the next paragraph.

For a simpler notation, let V^* be the set of vertices not in T_v , and call a set \mathcal{S} of characteristics consisting of a characteristic for each v_1, \dots, v_ℓ *compatible* if, for each pair of characteristics $\mathcal{Q}', \mathcal{Q}'' \in \mathcal{S}$, the forbidden colors of \mathcal{Q}' and of \mathcal{Q}'' are disjoint. Since finding a convex recoloring implies that each color $c' \in sep(C, v) \setminus \{c\}$ can be used only for one child v_i of v to color T_{v_i} , we are only allowed to combine compatible characteristics of v_1, \dots, v_ℓ to a characteristic for v . Therefore, the algorithm considers all possibilities to assign each color of $sep(C, v) \setminus \{c\}$ to one connected component of $T[V - \{v\}]$; more exactly, searching only for effective convex recolorings implies that each color $c' \in sep(C, v) \setminus \{c\}$ can be assigned to one component of $comp_{c'}(C, v)$. Let C'_{V^*} be the colors assigned to $T[V^*]$, i.e., C'_{V^*} is designated to be used in $T[V^*]$. After assigning all colors, the algorithm can compute by some simple, but ugly computations the cost x of a cheapest recoloring of (T_v, C) extending colorings for the subtrees $T_{v_1}, \dots, T_{v_\ell}$ with the assigned colors using the values stored in $cost_{v_1}, \dots, cost_{v_\ell}$. If $cost_v(c, sep(C, v) \setminus (\{c\} \cup C'_{V^*}))$ is larger than x , we update it to x .

Bar-Yehuda et. al. observed that an efficient implementation of the algorithm above runs in time

$$O(n^2 + \sum_{v \in V} (|sep(C, v)| \cdot ((deg(v) + 1) + \prod_{c' \in sep(C, v)} |comp_{c'}(C, v)|))).$$

If $s = \max_{v \in V} sep(C, v)$ is additionally defined as complexity parameter, a less detailed description of the running time is $O(n^2 + s \sum_{v \in V} (n + n^s)) = O(n^2 + sn^{s+1})$. Comparing this running time with the running time of the algorithm of the next section in the case $k = 1$ —see Theorem 5.5.8—we can observe that on trees the latter algorithm is better for graphs with many vertices. The reason for this improvement is that the algorithm above has a much better running time on binary trees than on arbitrary trees and that an arbitrary tree T can always be 'packed' into a binary tree decomposition for T . Therefore, it can be also a good idea to use tree decompositions even for trees.

5.5 MCRP on Graphs of Bounded Treewidth

Extending the result of the last section we obtain in this section an algorithm for the MCRP on weighted colored graphs (G, C) . However, Obs. 5.4.2 can not be easily generalized to tree decompositions. Without using the restriction to effective colorings, we must consider the possibility that a vertex in the given tree is colored by all possible colors of the initial coloring. If we follow this approach in a straight forward way, we have to store at each node w of a tree decomposition (T, B) for an n -vertex graph G at least $\Omega(|C(G)|^{|B(w)|})$ characteristics, where possibly $|C(G)| = \Theta(n)$. In other words, if G has treewidth k , we would obtain as running time $\Omega(n^{k+1})$. Additional ideas presented on the following three pages still guarantee a running time quadratic in n .

For having a smaller number of case distinctions, we start with the definition of a special kind of a tree decomposition.

Definition 5.5.1 (nice tree decomposition). A tree decomposition (T, B) for an n -vertex graph is called *nice* if

- T is a rooted and binary tree with $O(n)$ nodes
- $B(w) = B(w_1) = B(w_2)$ holds for each node w of T with two children w_1 and w_2 , and
- for all nodes w of T with only one child w_1 , either $|B(w) \setminus B(w_1)| = 1$ and $B(w) \supset B(w_1)$ or $|B(w_1) \setminus B(w)| = 1$ and $B(w_1) \supset B(w)$ holds.

Given an n -vertex graph G and a binary tree decomposition (T, B) with $O(n)$ nodes and width k , we easily obtain a nice tree decomposition by modifying T in $O(k^2n)$ time [61]. In detail, for each edge $\{w_1, w_2\}$ of T with $B(w_1)$ and $B(w_2)$ differing by $x > 1$ vertices, replace $\{w_1, w_2\}$ by a new w_1 - w_2 -path of length x and assign bags to the nodes of the new path such that stepwise $B(w_1)$ transforms to $B(w_2)$. Moreover, as long as T contains a node w having only one child w' with $B(w') = B(w)$, connect the children of w' to w and remove w' from T . Finally, for each node w of T having two children, check if its bag is identical to the bag of each child w' of w . In case of a failure, replace edge $\{w, w'\}$ of T by a w - w' -path P of length 2 and take as content of the bag for the inner vertex of P the content of $B(w)$. Note that all modifications can be done by a DFS on T in $O(k^2n)$ time and that the tree decomposition obtained is nice.

For the remainder of this section, we assume that we are given an n -vertex graph $G = (V, E)$ and a nice tree decomposition (T, B) for G of width $k - 1$ ($n, k \in \mathbb{N}$) such that T has $O(n)$ nodes. The description of an algorithm for the convex coloring problems needs some further notations and definitions. By w_l and w_r we denote the left and the right child, respectively, of w in a tree T . In addition, if w has only one child, we define it to be a left child. We also introduce a new set consisting of k *gray colors*—in this chapter always denoted by Y —and we allow for each recoloring additionally to use the gray colors. A gray colored vertex v of G intuitively means that v is uncolored and is later colored with a real color. We therefore define the *cost* for recoloring a gray colored vertex to be 0 and do not consider the gray colors as real colors. A *convex coloring* from now on should denote a coloring C where all pairs of vertices of the same gray or real color are C -connected. For each node w in T , each subset S of vertices of G , each subgraph H of G , and each coloring C of a subgraph G' of G with G' containing all vertices of S and of H , we let

- $B(\overline{T_w})$ be the set vertices in a bag of a node being no descendant of w .
- $C(S), C(H)$ be the set of real colors used by C to color the vertices of S and of H , respectively.
- $\text{SEP}(C, w)$ be the set of real colors used to color vertices in more than one of the graphs $G[B(T_{w_l}) \setminus B(w)]$, $G[B(T_{w_r}) \setminus B(w)]$, and $G[B(\overline{T_w})]$.

On trees, we could easily restrict the recolorings we must consider to find an optimal solution by Obs. 5.4.2. Next, a weaker restriction for graphs of bounded treewidth is defined.

Definition 5.5.2 (legal coloring). A *legal recoloring* of a colored graph (G, C) is a recoloring C' of (G, C) such that, for each color c assigned by C' , there is a vertex u of G with $c = C(u) = C'(u)$.

Observe that, if there is a convex recoloring C'' of a colored graph (H, C) of cost k , there is also a legal convex recoloring C' of (H, C) with cost k . C' can be obtained from C'' without increasing the cost by uncoloring all vertices colored with a color c for which no vertex u with $C''(u) = C(u) = c$ exists. Hence for solving the MCRP, we only need to search for legal recolorings.

For the rest of this section we assume that our given graph G is a weighted graph being colored by an initial coloring C without any gray colors. Similar to the algorithm on trees, the algorithm considers the nodes of T in a bottom-up traversal, computes for each node w a set of characteristics, and stepwise extends the colorings represented by a characteristic. However, *extending a (re-)coloring* with gray colors needs a special rule: A vertex colored with a gray color c_1 may be recolored with a real color c_2 by an extension if all vertices of color c_1 are recolored with c_2 by the extension. For the MCRP on graphs of bounded treewidth, a *characteristic* for a node w is defined as a tuple $(P, (P_S)_{S \in P}, (c_S)_{S \in P}, Z)$, where

- P is a partition of $B(w)$, i.e., a collection of nonempty pairwise disjoint sets S_1, \dots, S_j with $j \in \mathbb{N}$ and $\bigcup_{1 \leq i \leq j} S_i = B(w)$. The sets S_1, \dots, S_j are called *macro sets*.
- P_S is a partition of the macro set S , where the subsets of S contained in P_S are called *micro sets*.
- $c_S \in C(B(w)) \cup \text{SEP}(C, w) \cup Y \cup \{0, -1\}$, where -1 is an extra value different from the real and gray colors.
- $Z \subseteq \text{SEP}(C, w)$. The colors in Z are called the *forbidden colors*.

In the following, for a characteristic \mathcal{Q} and a macro set S of \mathcal{Q} , we denote the values P, P_S, c_S and Z above by $P^\mathcal{Q}, P_S^\mathcal{Q}, c_S^\mathcal{Q}$ and $Z^\mathcal{Q}$.

We next describe a first intuitive approach of solving the MCRP extending the ideas from trees to graphs of bounded treewidth by introducing macro and micro sets, but not using gray colors or the extra value -1 . A characteristic \mathcal{Q} for a node w represents a collection \mathcal{C} of colorings of $B(T_w)$ such that, for each coloring $C' \in \mathcal{C}$, the following holds: A macro set S of \mathcal{Q} denotes a maximal subset of vertices in $B(w)$ that are colored by C' with the same unique color equal to the value $c_S^\mathcal{Q}$ stored with the macro set—maximal means that there is no further vertex in $B(w) \setminus S$ colored with $c_S^\mathcal{Q}$. A micro set is a maximal subset of a macro set that is C' -connected in $G[B(T_w)]$. Concerning the set Z of forbidden colors, for a coloring represented by the characteristic, we want to have $Z = \text{SEP}(C, w) \cap (C'(B(T_w)) \setminus C'(B(w)))$. The reason that these colors are called forbidden is that the real colors in $C'(B(T_w)) \setminus C'(B(w))$ may not be used any more to color a vertex in $V \setminus B(T_w)$. The additional restriction to the set $\text{SEP}(C, w)$ is possible since we only want to consider legal recolorings.

The definition of a characteristic above using tree decompositions needs more information compared to a characteristic on trees. The reason is that macro

and micro sets are superfluous on trees: Note that there is at most one common vertex in the bag of a node and the bag of its parent if we take the tree decompositions for trees constructed in the proof of Lemma 1.3.12. Since the division into macro and micro sets is essential only for the common vertices in the bags of a node and its parent, already the algorithm using a tree decomposition for trees does not really need macro and micro sets. Thus, it should be no surprise any more that the algorithm on trees does not need macro and micro sets. Ignoring additionally the gray colors and the extra value -1 we obtain the definition of a characteristic on trees from the last section—now formulated in the context of tree decompositions.

The main idea of the algorithm is the following: Given all characteristics for the children of a node w and, for each collection \mathcal{C} of colorings described by one of these characteristics, the minimal cost among all costs of recolorings in \mathcal{C} , the algorithm uses a bottom-up traversal to compute the same information also for w and its ancestors. Since we only want to compute convex recolorings, we have to remove at the root of T all characteristics having a real colored macro set that consists of at least two micro sets. The cost of an optimal convex recoloring is the minimal cost among all costs computed for the remaining characteristics. An additional top-down traversal of T can also determine a recoloring having optimal cost. Unfortunately, the number of characteristics to be considered by the approach above would be too high for an efficient algorithm. The problem is that for graphs of bounded treewidth, in contrary to what is the case for trees, a path connecting two vertices in $V \setminus B(T_w)$ may use vertices in $B(T_w)$, and a path connecting two vertices in $B(T_w)$ may use vertices in $V \setminus B(T_w)$. In other words, we must take into account all legal recolorings contrary to the restriction of the last section, where we take into account only effective recolorings. In order to avoid a too large increase of the running time, as a further extension to the algorithm on trees, we use gray colors and the extra value -1 intuitively as follows.

If a real color c is used by C only to color vertices in $V \setminus B(T_w)$, a recoloring C' of G may possibly also want to recolor a set S of vertices in $B(T_w)$ with c in order to C' -connect some vertices with color c . The cost for recoloring vertices of $B(T_w)$ with c is independent of the exact value of c and can be computed as the cost of uncoloring all vertices of S (since $C(v) \neq c$ by our choice of c for all v in $B(T_w)$) and the cost of recoloring it (without any costs) with color c . Therefore, when considering recolorings of the graph $B(T_w)$, we do not allow to color it with a real color $c \notin C(B(T_w))$. Instead of c we use a gray color since coloring a vertex gray has the same cost as uncoloring the vertex but allows us to distinguish the vertex from vertices in $B(w)$ colored with another gray color or being uncolored. Note that our definition of extending a recoloring allows us with zero cost to recolor gray vertices in a later step with a real color, whereas recoloring real-colored vertices is forbidden when extending a recoloring.

If a recoloring C' of $B(T_w)$ colors a macro set S with a real color c that is only used by C to color vertices of $B(T_w)$, then for extending the recoloring C' to a recoloring C'' , we do not need to know the exact color of S . The reason for this is that, for any vertex v in $V \setminus B(T_w)$, the cost for setting $C''(v) = c$ can be computed again independently of the color of S : We have to pay the weight of v as cost if v is real-colored by C and cost zero otherwise. Thus, we use the extra value -1 to denote that a macro set S is real-colored with a color $c \in C(B(T_w)) \setminus C(B(\overline{T_w}))$, and in this case, we set $c_S^Q = -1$ instead of $c_S^Q = c$.

Following the ideas described above we let the algorithm consider only a restricted class of characteristics. For a node w of T , we denote by $C|_{B(T_w)}$ the coloring C restricted to $B(T_w)$. We define a characteristic \mathcal{Q} to be a *good characteristic* if there exists a legal recoloring C' of $(B(T_w), C|_{B(T_w)})$ with the properties (P1) - (P7). C' is then called to be *consistent* with \mathcal{Q} .

- (P1) $C'(B(T_w)) \subseteq C(B(T_w)) \cup Y \cup \{0\}$.
- (P2) For each macro set S of \mathcal{Q} , C' colors all vertices of S with one color c . Moreover, if c is a real color in $C(B(T_w)) \setminus C(B(\overline{T_w}))$, then $c_S^{\mathcal{Q}} = -1$, and $c_S^{\mathcal{Q}} = c$ otherwise.
- (P3) C' colors vertices of different macro sets with different colors.
- (P4) A micro set is a maximal subset of $B(w)$ that is C' -connected in $B(T_w)$.
- (P5) C' is a convex recoloring for the graph obtained from $B(T_w)$ by adding, for each macro set S , edges of an arbitrary simple path visiting exactly one vertex of each micro set of S .
- (P6) Every gray colored vertex in $B(T_w)$ is C' -connected to a vertex in $B(w)$.
- (P7) $Z^{\mathcal{Q}} = \text{SEP}(C, w) \cap (C'(B(T_w)) \setminus C'(B(w)))$.

Note that each convex legal recoloring C' of the initial colored graph (G, C) is consistent with a good characteristic \mathcal{Q} for the root r of T . More explicitly, we obtain \mathcal{Q} by dividing $B(r)$ into macro sets each consisting of all vertices of one color with respect to C' , by defining the partition of each macro set to consist of only one micro set, by setting $Z^{\mathcal{Q}} = \emptyset$ and, by defining, for each macro set S , $c_S^{\mathcal{Q}} = -1$ if $C'(S)$ is a real color or $c_S^{\mathcal{Q}} = 0$ otherwise.

Our algorithm computes in a bottom-up traversal for each node w of T all good characteristics of w from the good characteristics of the children of w . However, not all pairs of good characteristics of the children can be combined to good characteristics \mathcal{Q} of w . Therefore we call a characteristic \mathcal{Q}_l of w_l and a characteristic \mathcal{Q}_r of w_r *compatible* if they satisfy the following three conditions:

- Two vertices $v_1, v_2 \in B(w_l) = B(w_r)$ belong to the same macro set in \mathcal{Q}_l if and only if this is true for \mathcal{Q}_r .
- For each macro set S of \mathcal{Q}_l and hence also of \mathcal{Q}_r , either $c_S^{\mathcal{Q}_l} = c_S^{\mathcal{Q}_r} \neq -1$ or $\{c_S^{\mathcal{Q}_l}, c_S^{\mathcal{Q}_r}\}$ contains exactly one gray color and not the value 0.
- The sets of forbidden colors of \mathcal{Q}_l and of \mathcal{Q}_r are disjoint.

Note that in a tree T' with a vertex v , T'_{v_l} and T'_{v_r} have no common vertices; thus, the first and the second condition are superfluous in trees. The definition above is hence a generalization of the definition of 'compatible' in the last section.

The following algorithm computes, for each node w of T , a set \mathcal{M}_w of characteristics for which we later show that it is exactly the set of good characteristics of w . First of all, in a preprocessing phase compute by a bottom-up and a top-down traversal of T , for each node w of T , the set $\text{SEP}(C, w)$ as well as the subset of real colors in $C(B(T_w)) \setminus C(B(\overline{T_w}))$ in the following denoted by $\text{LOW}(C, w)$. Second, start a bottom-up traversal of T . For each leaf w of T , \mathcal{M}_w is obtained by taking into account all possible divisions of the vertices of

$B(w)$ into macro sets and all possible colorings of each macro set S with colors of $C(S) \cup Y \cup \{0\}$ such that, for each pair S' and S'' of macro sets, S' and S'' is colored differently. More precisely, for each choice, a characteristic \mathcal{Q} is obtained and added to \mathcal{M}_w by defining, for each macro set S colored with a color c , the micro sets of S to be the connected components of the subgraph of G induced by the vertices of S and by setting $c_S^{\mathcal{Q}} = -1$ if c is a real color in $\text{LOW}(C, w)$ or $c_S^{\mathcal{Q}} = c$ otherwise. The set $Z^{\mathcal{Q}}$ of forbidden colors is set to \emptyset .

At a non-leaf w all already computed characteristics of the children are considered. In detail, for each characteristic \mathcal{Q}_l of \mathcal{M}_{w_l} and—if w has two children—for each compatible good characteristic \mathcal{Q}_r of \mathcal{M}_{w_r} , we add to \mathcal{M}_w the set of characteristics that could be obtained as output of the following non-deterministic algorithm:

- For the vertices in $B(w) \cap B(w_l)$, take the same division into macro sets for \mathcal{Q} as for \mathcal{Q}_l . If w has only one child and if there is also a vertex $v \in B(w) \setminus B(w_l)$, choose one of the $\leq k - 1$ possibilities of assigning v to one of the macro sets of $B(w) \cap B(w_l)$ or choose $\{v\}$ to be its own new macro set.
- For dividing the vertices of $B(w)$ into micro sets, construct the graph H consisting of the vertices in $B(w)$ and having an edge between two vertices if and only if both vertices belong to the same macro set and either this edge exists in G or both vertices belong to the same micro set in \mathcal{Q}_l or \mathcal{Q}_r . Define the vertices of each connected component in H to be a micro set of \mathcal{Q} .
- For each macro set S obtained by the construction above, distinguish between three cases. Let $C^+(\mathcal{Q}_l)$ be the union of $Z^{\mathcal{Q}_l}$ and the values $c_{S_1}^{\mathcal{Q}_l}$ over all macro sets S_1 of \mathcal{Q}_l .
 - $S \subseteq S'$ for a macro set S' of \mathcal{Q}_l : If w_r does not exist or if $c_{S'}^{\mathcal{Q}_l} = c_{S'}^{\mathcal{Q}_r}$, set $c_S^{\mathcal{Q}} = c_{S'}^{\mathcal{Q}_l}$. Otherwise set $c_S^{\mathcal{Q}}$ to the non-gray value in $\{c_{S'}^{\mathcal{Q}_l}, c_{S'}^{\mathcal{Q}_r}\}$.
 - $|S| > 1$ and $\exists v \in S \cap (B(w) \setminus B(w_l))$: If $C(v) = 0$ or $c_{S \setminus \{v\}}^{\mathcal{Q}_l}$ is no gray color, set $c_S^{\mathcal{Q}} = c_{S \setminus \{v\}}^{\mathcal{Q}_l}$ else choose $c_S^{\mathcal{Q}} \in \{c_{S \setminus \{v\}}^{\mathcal{Q}_l}\} \cup (\{C(v)\} \setminus C^+(\mathcal{Q}_l))$.
 - $S = \{v\}$ with $v \in B(w) \setminus B(w_l)$: Choose for $c_S^{\mathcal{Q}}$ a value in the set $(Y \cup \{0, C(v)\}) \setminus C^+(\mathcal{Q}_l)$.

After defining $c_S^{\mathcal{Q}}$ as described above, if $c_S^{\mathcal{Q}}$ is a real color in $\text{LOW}(c, w)$, redefine $c_S^{\mathcal{Q}} = -1$.

- Reject the computation if there is a singleton micro set S' part of a real- or gray-colored macro set S in \mathcal{Q}_l with $S' \cap B(w) = \emptyset$ and either $S \setminus S' \neq \emptyset$ or $c_{S'}^{\mathcal{Q}_l}$ is a gray color.
- If there is a singleton macro set $S = B(w_l) \setminus B(w)$ of \mathcal{Q}_l and if $c_S^{\mathcal{Q}_l}$ is a real color, set $Z' = \{c_S^{\mathcal{Q}_l}\}$. In all other cases, set $Z' = \emptyset$. Finally, set $Z^{\mathcal{Q}} = \text{SEP}(C, w) \cap (Z' \cup Z^{\mathcal{Q}_l} \cup Z^{\mathcal{Q}_r})$ with $Z^{\mathcal{Q}_r} = \emptyset$ if w has only one child.

For a non-leaf w of T , it is not hard to see that our algorithms constructs for a good characteristic \mathcal{Q}_l of w_l and, if w_r exists, also for a compatible good

characteristic \mathcal{Q}_r of w_r , only good characteristics \mathcal{Q} of w for which there exist colorings C' , C'_1 and, if w_r exists, also C'_r consistent with \mathcal{Q} , \mathcal{Q}_l , and \mathcal{Q}_r , respectively, such that the following conditions hold:

- (C1) C' is a color extension of C'_1 and, if w_r exists, also of C'_r .
- (C2) For each $x \in \{l, r\}$ with w_x being a child of w , two vertices in $B(w) \cap B(w_x)$ belong to the same macro set in \mathcal{Q}_x if and only if this is true for \mathcal{Q} .
- (C3) If w has two children and C' colors a macro set S of \mathcal{Q} with a real color c , then either C'_l or C'_r also colors S with c .
- (C4) If w has only one child, if S is a macro set not containing a vertex $v \in B(w) \setminus B(w_1)$, and if S is colored with a real color c , then C'_1 also colors S with c .
- (C5) If w has only one child and there is a vertex $v \in B(w) \setminus B(w_1)$, C' colors the macro set S of \mathcal{Q} containing v with a color in $\{0, C(v)\} \cup Y \cup C'_1(S \cap B(w_1))$.

We now show that even adhering to the restrictions above one can still construct all good characteristics for a node w .

Lemma 5.5.3. *Let C' be a recoloring consistent with a good characteristic \mathcal{Q} of a node w of T . Then there exists a recoloring C'_1 consistent with a good characteristic \mathcal{Q}_l of w_l and, if w_r exists, also a coloring C'_r consistent with a good characteristic of \mathcal{Q}_r of w_r compatible to \mathcal{Q}_l such that (C1) - (C5) hold.*

Proof. Let w_x with $x \in \{l, r\}$ be a child of w . We first define a good characteristic \mathcal{Q}_x for w_x and a recoloring C'_x consistent with it. The recoloring C'_x is obtained from restricting C' to the vertices in $B(T_{w_x})$, and in the case of $B(w) \setminus B(w_1)$ consisting of a vertex being colored with a real color $c \notin C(B(T_{w_x}))$, by additionally recoloring all vertices of color c with one gray color not in $C'(B(w) \cap B(w_1))$. Note that there are enough gray colors to color all vertices with different gray colors since $|Y| = k$. Let the macro sets of \mathcal{Q}_x ($x \in \{l, r\}$) be the maximal subsets of the vertices in $B(w_x)$ colored with the same color by C'_x and the micro set of \mathcal{Q}_x be maximal subsets of vertices of $B(w_x)$ that are C'_x -connected in $B(T_{w_x})$. We then define the forbidden colors $Z^{\mathcal{Q}_x}$ as in (P7) and set, for each macro set S of \mathcal{Q}_x , $c_S^{\mathcal{Q}_x}$ to -1 if the only color c in $C'_x(S)$ is contained in $\text{LOW}(C, w_x)$ and to c , otherwise. By these definitions \mathcal{Q}_x is a good characteristic and C'_x a recoloring consistent with it, i.e., properties (P1) - (P7) hold for \mathcal{Q}_x and C'_x . Additionally, if w has two children, \mathcal{Q}_l and \mathcal{Q}_r are compatible.

It is easy to see that (C1) and (C2) hold. For the properties (C3) and (C4) let us assume that C'_x ($x \in \{l, r\}$) colors a vertex v gray that is colored with a real color c by C' . Then by our construction of C'_x , we have $c \notin C(B(T_{w_x}))$. Recall that C' is a legal coloring. If w has only one child, there must be a vertex $v \in B(w) \setminus B(w_1)$ with $c = C'(v) = C(v)$. Thus, (C4) holds. If w has two children, we must have $c \in C(B(T_{w_y}))$ for $w_y \neq w_x$ being the other child of w . Therefore, the characteristic for w_y colors the macro set containing v with c and (C3) holds. For showing (C5), let us assume that there is a vertex $v \in B(w) \setminus B(w_1)$. Let S be the macro set of \mathcal{Q} containing v and let us assume that S is colored by C' with a real color $c \neq C(v)$. Since C' is a legal coloring,

there must be a vertex v' in $B(T_w)$ with $v' \neq v$ and $C'(v') = C(v') = c$. By our construction C'_1 colors v' also with c , and c is the color in $C'_1(S - \{v\}) = C'_1(S \cap B(w_1))$. \square

The next two lemmata show that our algorithm correctly computes the set of all good characteristics for each node of T .

Lemma 5.5.4. *Let \mathcal{Q}' and \mathcal{Q}_l be good characteristics of a node w and its child w_l , respectively, and, if w_r exists, \mathcal{Q}_r be a compatible good characteristic of w_r . Let C'_1 and, if w_r exists, C'_r be recolorings consistent with \mathcal{Q}_l and \mathcal{Q}_r , respectively. Assume that there is a coloring C' being consistent to \mathcal{Q}' such that (C1) - (C5) holds for C' , C'_1 , and C'_r (if existing). If the algorithm constructs a characteristic \mathcal{Q} from the characteristics \mathcal{Q}_l and \mathcal{Q}_r (if existing), then the value computed for $Z^{\mathcal{Q}}$ by the algorithm for the MCRP is exactly the set of forbidden colors of \mathcal{Q}' .*

Proof. Recall that the algorithm computes the value $\text{SEP}(C, w) \cap (Z' \cup Z^{\mathcal{Q}_l} \cup Z^{\mathcal{Q}_r})$ for $Z^{\mathcal{Q}}$, where $Z^{\mathcal{Q}_r} = \emptyset$ if w has only one child and Z' is defined as in the algorithm for the MCRP. In contrast, by the definition of a consistent recoloring we must have $Z^{\mathcal{Q}'} = \text{SEP}(C, w) \cap (C'(B(T_w)) \setminus C'(B(w)))$. It remains to show $Z^{\mathcal{Q}'} = \text{SEP}(C, w) \cap (Z' \cup Z^{\mathcal{Q}_l} \cup Z^{\mathcal{Q}_r})$.

We first show that $Z^{\mathcal{Q}'} \subseteq \text{SEP}(C, w) \cap (Z' \cup Z^{\mathcal{Q}_l} \cup Z^{\mathcal{Q}_r})$. Hence, let $c \in Z^{\mathcal{Q}'}$. Then $c \in \text{SEP}(C, w)$, and a vertex $v' \in V \setminus B(T_w)$ exists with $C'(v') = c$. If C' colors a vertex in $B(w_1)$ with c , we have $c \in Z'$ since $c \notin C'(B(w))$, i.e., c is the real color of the singleton macro set $S = B(w_1) \setminus B(w)$. Otherwise, there exists a vertex v in $B(T_w) - (B(w) \cup B(w_1))$ with $C'(v) = C(v) = c$ since C' is a legal recoloring and since $c \in Z^{\mathcal{Q}'}$. Thus, $c \in \text{SEP}(C, w_x)$ and hence $c \in Z^{\mathcal{Q}^x}$ for a child w_x ($x \in \{l, r\}$) of w .

We next show $\text{SEP}(C, w) \cap (Z' \cup Z^{\mathcal{Q}_l} \cup Z^{\mathcal{Q}_r}) \subseteq Z^{\mathcal{Q}'}$. Since C' is a convex coloring, we know that $Z^{\mathcal{Q}_l} \cup Z^{\mathcal{Q}_r}$ does not contain a color of $C'(B(w))$. Consequently, $Z^{\mathcal{Q}_l} \subseteq C'(B(T_{w_l})) \setminus C'(B(w))$, $Z^{\mathcal{Q}_r} \subseteq C'(B(T_{w_r})) \setminus C'(B(w))$ and also $Z' \cup Z^{\mathcal{Q}_l} \cup Z^{\mathcal{Q}_r} \subseteq C'(B(T_w)) \setminus C'(B(w))$ holds. The last equation implies $\text{SEP}(C, w) \cap (Z' \cup Z^{\mathcal{Q}_l} \cup Z^{\mathcal{Q}_r}) \subseteq Z^{\mathcal{Q}'}$. \square

Lemma 5.5.5. *For each node w , \mathcal{M}_w is the set of good characteristics of w .*

Proof. We first prove that the set \mathcal{M}_w is indeed a set of characteristics. The only difficult point is to show that the color c assigned to a macro set is in $C(B(w)) \cup \text{SEP}(C, w) \cup Y \cup \{0, -1\}$. Hence, assume that this is not the case. Then either $c \in C(B(T_w)) \setminus C(B(w))$ or $c \in \text{LOW}(C, w)$. However, both cases can not occur since it is easy to see that in the first case our algorithm assigns to c only real colors in $C(B(T_w))$, and in the latter case, our algorithm chooses $c = -1$. It remains to show that all characteristics in \mathcal{M}_w are good and that no good characteristic of w is missing in \mathcal{M}_w .

For a leaf w of T , the statement of the lemma is clear since we consider all possible divisions into macro sets and, for each macro set S , all possible colorings with colors in $C(S) \cup \{0\} \cup Y$ different from the colorings chosen for the other macro sets. The further restrictions can easily be shown to be necessary and sufficient to guarantee that \mathcal{M}_w is the set of all good characteristics of w .

For a non-leaf w of T , we next want to consider which possibilities exist to construct a good characteristic \mathcal{Q} of w from a good characteristic \mathcal{Q}_l of w_l and, if w_r exists, a compatible good characteristic \mathcal{Q}_r of w_r such that there are colorings

C' , C'_1 and, if w_r exists, C'_r consistent with \mathcal{Q} , \mathcal{Q}_1 and \mathcal{Q}_r , respectively, such that the conditions (C1) - (C5) hold. By Lemma 5.5.3 it suffices to construct only such characteristics.

The divisions into macro sets considered by the algorithm are the only possibilities without violating (C2). Moreover, because of property (P4) the vertices of a macro set must be divided into micro sets as it is done by the algorithm.

We next consider the possible colorings of a macro set S . If S contains a vertex $v \in B(w) \setminus B(w_1)$, v can introduce a new color. However, beside the choices made by the algorithm there are no further possibilities to color S without violating (C1) or (C5). If S contains no vertex $v \in B(w) \setminus B(w_1)$, we have $S \subseteq S'$ for a macro set S' of \mathcal{Q}_1 . In this case we have to pay attention to (C1), (C3) and (C4). If w has two children (i.e., $S' = S$) and if $c_{S'}^{\mathcal{Q}_1} \neq c_{S'}^{\mathcal{Q}_r}$, exactly one of the two values is a gray value and the other is equal to a value $c \neq 0$ since \mathcal{Q}_1 and \mathcal{Q}_r are compatible. Thus, to guarantee (C1) we have to define $c_S^{\mathcal{Q}} = c$. Otherwise we have to set $c_S^{\mathcal{Q}} = c_{S'}^{\mathcal{Q}_1}$ for not violating (C1), (C3) and (C4), where the conditions (C3) and (C4) forbid a recoloring in the case of $c_{S'}^{\mathcal{Q}_1}$ being a gray color. Note that the redefinitions $c_S^{\mathcal{Q}} = -1$ made by the algorithm are necessary because of property (P2).

We already know that $Z^{\mathcal{Q}}$ is defined correctly by Lemma 5.5.4. To sum up, there is no other possibility to construct a good characteristic \mathcal{Q} for w and a recoloring consistent with \mathcal{Q} without violating our properties (P1) - (P7) and (C1) - (C5). It is also easy to verify that, because of the compatibility of \mathcal{Q}_1 and \mathcal{Q}_r , our recursive construction only constructs characteristics satisfying our properties. In particular, for guaranteeing (P5) and (P6) we use the fact that our non-deterministic algorithm rejects some computations. \square

Lemma 5.5.6. *Our algorithm can be implemented with a running time of $O(n^2 + 4^s(2k + s + 2)^{6k+1}(k^2 + s)n)$, where $s = \max_{w \text{ node of } T} |\text{SEP}(C, w)|$.*

Proof. The algorithm can compute the sets $\text{SEP}(C, w)$ and $\text{LOW}(C, w)$ for all nodes w easily in $O(n^2)$ time in a preprocessing phase. We next analyze the running time for computing all good characteristics for a leaf w of T . In general, there are at most k^k possibilities to divide the vertices of $B(w)$ into macro sets. For each such partitioning, there are at most $(2k + 1)^k$ possibilities of assigning different colors of $C(B(w)) \cup Y \cup \{0\}$ to all macro sets—one to each macro set. For each such choice, we have to determine the connected components defining the micro sets. This can be done in $O(k^2)$ time for each choice. Hence the running time for a leaf is $O(k^k(2k + 1)^k k^2)$.

Note that a non-leaf w of T can have at most $k^{2k}(2k + s + 2)^k 2^s$ possible good characteristics: There are at most $k^k \cdot k^k = k^{2k}$ possibilities for a division into micro and macro sets, $(2k + s + 2)^k$ possibilities for choosing values of $C(B(w)) \cup \text{SEP}(C, w) \cup Y \cup \{0, -1\}$ for the macro sets and 2^s possibilities to choose a set of forbidden colors. For a leaf, there are not more possible good characteristics.

Let us analyze the time needed for constructing a single good characteristic \mathcal{Q} of a node w from fixed good characteristics \mathcal{Q}_1 and possibly \mathcal{Q}_r for the children of w . The sets $Z^{\mathcal{Q}}$ and $C^+(\mathcal{Q}_1)$ can be constructed in $O(k + s)$ time from $Z^{\mathcal{Q}_1}$ and $Z^{\mathcal{Q}_r}$. All remaining computations (the division into micro sets, recoloring some vertices, etc.) can be done in $O(k^2)$ time. Note that, for our pair of compatible characteristics \mathcal{Q}_1 and \mathcal{Q}_r we have to compute at most $3k = (2(k - 1)) + (k + 2)$

new good characteristics for w : If $B(w)$ contains a vertex $v \in B(w) \setminus B(w_1)$ —and hence w has only one child—we have at most $k - 1$ possibilities to assign v to a macro set of \mathcal{Q}_1 and, as a further possibility, to make $\{v\}$ to a new macro set. Moreover, we have at most two possibilities to color a macro set that contains v and another vertex; and we have $k + 2$ possibilities to color a macro set $\{v\}$. If $B(w)$ contains no vertex of $v \in B(w) \setminus B(w_1)$ we have less possibilities. Therefore, for fixed characteristics for the children of w , we can determine all of the characteristics above in $O(k(k^2 + s))$ time. Whether two characteristics are compatible can be also tested in $O(k(k^2 + s))$ time. Since for a non-leaf with two children there are at most $(k^{2k}(2k + s + 2)^{k2s})^2$ different pairs of compatible characteristics for its children, the good characteristics of a non-leaf can be computed in $O((k^{2k}(2k + s + 2)^{k2s})^2 k(k^2 + s)) = O(4^s(2k + s + 2)^{6k+1}(k^2 + s))$ time. The lemma now follows from the fact that T has $O(n)$ nodes. \square

The next lemma is essential for computing a legal recoloring of minimal cost.

Lemma 5.5.7. *Our algorithm can be extended such that it computes for each good characteristic \mathcal{Q} the cost of a recoloring consistent with \mathcal{Q} that among all such recolorings has minimal cost. The asymptotic running time does not increase by this extension.*

Proof. For a good characteristic \mathcal{Q} of a node w , let $V^*(\mathcal{Q})$ be the set of vertices of $B(w)$ contained in a macro set S of \mathcal{Q} with $c_S^{\mathcal{Q}} \neq -1$. Let $\text{cost}_w(\mathcal{Q})$ be the cost of a coloring C' consistent with \mathcal{Q} that among all such colorings has minimal cost. Let $\text{cost}'_w(\mathcal{Q})$ be $\text{cost}_w(\mathcal{Q})$ minus the recoloring cost of C' for the vertices in $V^*(\mathcal{Q})$.

We first show how to compute $\text{cost}'_w(\mathcal{Q})$ for a good characteristic \mathcal{Q} of a node w . If w is a leaf, $\text{cost}'_w(\mathcal{Q})$ can be computed in $O(|B(w)|) = O(k)$ time. Assume for a moment that w has two children for which there is exactly one pair of a good characteristic \mathcal{Q}_1 for w_1 and a compatible good characteristic \mathcal{Q}_r for w_r such that (C1) - (C5) hold for colorings C' , C'_1 , and C'_r consistent with \mathcal{Q} , \mathcal{Q}_1 , and \mathcal{Q}_r , respectively. By definition of $\text{cost}'_{w_1}(\mathcal{Q}_1)$ and $\text{cost}'_{w_r}(\mathcal{Q}_r)$, the value $\text{cost}'_w(\mathcal{Q})$ can be computed from $\text{cost}'_{w_1}(\mathcal{Q}_1) + \text{cost}'_{w_r}(\mathcal{Q}_r)$ by adding the recoloring costs for the vertices in $(V^*(\mathcal{Q}_1) \cap V^*(\mathcal{Q}_r)) \setminus V^*(\mathcal{Q})$ —note that the latter cost is the same for each choice of a coloring C' consistent to \mathcal{Q} since \mathcal{Q}_1 and \mathcal{Q}_r being good compatible characteristics implies that, for each $v \in (V^*(\mathcal{Q}_1) \cap V^*(\mathcal{Q}_r)) \setminus V^*(\mathcal{Q})$, $C'(v)$ is equal to the non-gray value in $\{c_S^{\mathcal{Q}_1}, c_S^{\mathcal{Q}_r}\}$ for each coloring C' consistent to \mathcal{Q} and the macro set S of \mathcal{Q} containing v . Hence, the recoloring cost can be directly computed from \mathcal{Q}_1 and \mathcal{Q}_r , and is independent from the choice of C' . If a good characteristic \mathcal{Q} can be obtained—in the same way as described above—from several possible pairs of good characteristics \mathcal{Q}_1 and \mathcal{Q}_r , we determine the cost above for each such pair and define $\text{cost}'_w(\mathcal{Q})$ as the minimum of the costs. For each pair of good characteristics \mathcal{Q}_1 and \mathcal{Q}_r and for each good characteristic \mathcal{Q} obtained from \mathcal{Q}_1 and \mathcal{Q}_r the additional running time is $O(k)$. The value $\text{cost}'_w(\mathcal{Q})$ for a node w with one child can be computed in a similar way, i.e., we take the minimum of the cost for the vertices in $V^*(\mathcal{Q}_1) \setminus V^*(\mathcal{Q})$ plus $\text{cost}'_{w_1}(\mathcal{Q}_1)$ over all \mathcal{Q}_1 being consistent with \mathcal{Q} .

The $\text{cost}_w(\mathcal{Q})$ for each good characteristic \mathcal{Q} can be computed from $\text{cost}'_w(\mathcal{Q})$ in $O(k)$ time since in this time we can easily determine the recoloring cost of the vertices in $V^*(\mathcal{Q})$. Note that all modifications of our algorithm can be applied during the computation of a characteristic \mathcal{Q} for a fixed characteristic \mathcal{Q}_1 and, if

it exists, a fixed characteristic \mathcal{Q}_r in $O(k)$ extra time. Therefore the asymptotic running time of Lemma 5.5.6 does not increase. \square

After the removal of all characteristics having a real colored macro set that consists of at least two micro sets or having a gray colored macro set, we obtain the cost of an optimal legal convex recoloring as the minimal cost among all costs stored with the remaining characteristics constructed for the root of T . Finally by an additional top-down traversal the algorithm above can be easily extended such that—beside the minimal cost of a legal recoloring—it also determines the coloring itself within the same time bound.

Theorem 5.5.8. *Given a colored graph (G, C) and a tree decomposition (T, B) of width $k - 1$, the MCRP can be solved in $O(n^2 + 4^s(2k + s + 2)^{6k+1}(k^2 + s)n)$ time, where $s = \max_{w \text{ node of } T} |\text{SEP}(C, w)|$.*

5.6 MBRP and MRRP

It is easy to modify the algorithm from the last section such that it solves the MRRP within the same time bound: In each bottom-up step we only have to exclude recolorings that recolor a real-colored vertex with a gray or another real color.

Theorem 5.6.1. *Given a colored graph (G, C) and a tree decomposition (T, B) of width $k - 1$, the MRRP can be solved in $O(n^2 + 4^s(2k + s + 2)^{6k+1}(k^2 + s)n)$ time, where $s = \max_{w \text{ node of } T} |\text{SEP}(C, w)|$.*

Unfortunately, the algorithms above for the MCRP and the MRRP are exponential in s since there are 2^s different possible sets of forbidden colors. As in the last section, let us assume that we are given an n -vertex graph $G = (V, E)$ and a nice tree decomposition (T, B) for G of width $k - 1$ ($n, k \in \mathbb{N}$) such that T has $O(n)$ nodes. Recall that storing forbidden colors in a characteristic of a node w of T is necessary to distinguish between recolorings that color a subgraph of $B(T_w)$, but no vertex in $B(w)$ with a forbidden color c from recolorings that do not use color c . Note that we only have to search for a legal recoloring if we want to solve the MBRP or the MRRP—the reason is the same as for the MCRP. The good news concerning the MBRP on a general initial coloring and the MRRP with its initial coloring being an $(\infty, 3)$ -coloring is that we can omit to store the forbidden colors explicitly. We next describe the necessary modifications.

For the MBRP we use the same basic algorithm as for the MCRP. However, we compute as a solution for the MBRP w.l.o.g. only recolorings that, for each real color c , either recolor all or none of the vertices initially colored with c . Following this approach, a characteristic of a node w should only represent recolorings that, for each real color c , either recolor all or none of the vertices v in $B(T_w)$ for which $C(v) = c$ holds. If in the latter case $c \in \text{SEP}(C, w)$, we claim that a vertex of $B(w)$ is also colored with c since otherwise the recoloring cannot be extended to a legal convex recoloring not recoloring any vertex of c . This implies an additional rule for constructing characteristics:

Assume that—as in our algorithm of the last section for the MCRP—we want to construct a characteristic \mathcal{Q} of a non-leaf w from a characteristic \mathcal{Q}_x of a child w_x of w . Then we are only allowed to color a macro set S of \mathcal{Q} with c if

- (1) all vertices in $B(w)$ that are initially colored with c are contained in S and
(2) either \mathcal{Q}_x also contains a macro set S' with $c_{S'}^{\mathcal{Q}_x} = c$ or $c \notin C(B(T_{w_x}))$.

To test condition (2) efficiently, we construct in a preprocessing phase for each node w an array A_w with the following entries: For each color $c \in C(G)$, $A_w[c] = 1$ if $B(T_w)$ contains a vertex v with $C(v) = c$. Otherwise $A_w[c]$ is defined to be 0. If the array is computed by a bottom-up traversal of T , the preprocessing phase takes $O(n^2)$ time. After the preprocessing phase we can test, for each characteristic \mathcal{Q} of a node w constructed from characteristics \mathcal{Q}_l and possibly \mathcal{Q}_r of its children and each color c , in $O(k)$ time whether (1) and (2) hold. Hence, the asymptotic running time of the algorithm does not increase. Moreover, the additional rule enables us to find out, for each real color c , whether a vertex of $B(T_{w_x})$ ($x \in \{l, r\}$) is colored with c by considering $A_{w_x}[c]$ and by testing whether a macro set S of \mathcal{Q}_x is colored with c . Hence, there is no need to store the forbidden colors.

Theorem 5.6.2. *Given a graph with a tree decomposition (T, B) of bounded width the MBRP can be solved in polynomial time.*

More complicated modifications are necessary for the MRRP. Since in the MRRP a real-colored vertex may not be recolored with a real color, we can assume w.l.o.g. that, for each color c , there are either no or at least two vertices colored with c by C . The main idea of the algorithm is the following: To improve the running time at a node w of T , we only want to consider legal recolorings C' of $B(T_w)$ such that, for each color $c \in C(G)$, the following property $(C6)_c$ additionally holds. The correctness of this step is discussed later.

- $(C6)_c$ If u is a vertex in $B(T_w)$ with $C'(u) = C(u) = c$, then either there exists a vertex $v \neq u$ in $B(T_w)$ with $C(v) = C'(v) = c$ or $c \in C'(B(w))$ and there is another vertex v not in $B(T_w)$ with $C(v) = c$.

Let w be a node such that for a child w_x of w , $B(T_{w_x})$ contains exactly one vertex v initially colored with a color c , and let C' be a recoloring with property $(C6)_c$ such that it is consistent to a good characteristic \mathcal{Q} of w . Then, this property for example guarantees that we have $C'(v) = c$ if and only if $B(w_x)$ contains a vertex colored with c . Therefore, there is also no need to store c explicitly as a forbidden color in a characteristic of w_x any more. Since there is no need to store for a node w with $c \notin C(B(T_w))$ color c as forbidden color, we can conclude:

Observation 5.6.3. *Let w be a node of T such that, for each child w_x of w , $B(T_{w_x})$ contains at most one vertex of color c . Then, there is no need to store in a good characteristic \mathcal{Q} of w_x whether color c is a forbidden color since, by using property $(C6)_c$, this follows for each coloring C' consistent to \mathcal{Q} from the colors stored for the vertices in $B(w)$.*

We later show that also for all remaining nodes it is not necessary to store the color c as forbidden color in their characteristics. One problem of property $(C6)_c$ is that its application permits some legal recolorings. However, each convex recoloring C_{opt} of optimal cost either is a recoloring with property $(C6)_c$ at each node of T or it colors w.l.o.g. exactly one vertex u with c . In the latter case a coloring with the same cost as C_{opt} can be obtained from a recoloring with property $(C6)_c$ not coloring any vertex with c by undoing the uncoloring of

one vertex originally colored with c that among all such vertices has a maximal weight. Therefore, for computing the cost of an optimal convex recoloring, we only have to consider the cost of recolorings with property $(C6)_c$ and eventually to subtract the maximal weight over all vertices originally colored with c . Let us call such a subtraction a c -cost adaption. Our goal now is to describe an algorithm that runs the c -cost-adaption during the bottom-up traversal of T at a certain node w —called the c -decision node—having the following property:

For each characteristic \mathcal{Q} of w , either each recoloring C' of G , for which $(C6)_c$ holds and which extends a recoloring consistent with \mathcal{Q} , C' -connects at least two vertices initially colored with c (and we therefore must not run a c -cost-adaption) or all such recolorings uncolor all vertices initially colored with c (and therefore we have to run a c -cost-adaption).

If we know the c -decision node for each color c , the algorithm works as the algorithm for the MCRP with the following modifications: For each node w (also above the c -decision node), we only compute all good characteristics representing recolorings for which $(C6)_c$ holds for all c . If we reach the c -decision node w of a color c , for each characteristic \mathcal{Q} of w , we test whether all recolorings extending \mathcal{Q} do not use color c , and if so, we run a c -cost adaption for \mathcal{Q} .

For a characteristic \mathcal{Q} of a node w , let us define $C^*(\mathcal{Q})$ to be the set of all colors c for which the c -decision node is equal to w or a descendant of w , and for which we know that all recolorings extending \mathcal{Q} do not color any vertex with c . Then, the cost stored with \mathcal{Q} is the minimal cost among all costs of recolorings C' of $B(T_w)$ that can be obtained from a recoloring C'' consistent with \mathcal{Q} and satisfying $\forall c (C6)_c$ as follows: For each color $c \in C^*(\mathcal{Q})$, recolor the vertex of maximal weight among all vertices initially colored with c with its original color. Consequently, at the root of T , we obtain a good characteristic whose cost is exactly the optimal recoloring cost. Thus, it suffices to show, for each fixed color c ,

- (i) how we can find a c -decision node.
- (ii) how we guarantee that property $(C6)_c$ holds for each coloring consistent to a good characteristics of a node w of T .
- (iii) how we can decide, for a non-leaf w , whether a good characteristic is allowed to introduce c as a new color or not.

With (iii) we mean the following. Let us assume that we are given a good characteristic \mathcal{Q}_l of w_l and, if w_r exists, a good characteristic \mathcal{Q}_r of w_r as well as colorings C_l and C_r consistent to them. Then, we show that we can decide without making use of the set of forbidden colors whether colorings C' with the properties (C1) - (C5) and $(C6)_c$ (C'_l and C'_r chosen appropriately) can color a macro set with the color c or not.

Let $U = \{u_1, u_2\}$ or $U = \{u_1, u_2, u_3\}$ be the set of vertices initially colored with c by C and let $W = \{w_1, w_2\}$ or $W = \{w_1, w_2, w_3\}$ be a set of nodes in T with $u_i \in B(w_i)$ ($i \in \{1, 2\}$ or $i \in \{1, 2, 3\}$). We distinguish three cases.

Case 1: For each node of T , its bag contains at most one vertex initially colored with c . We choose from all lowest common ancestors of a pair of nodes in W

the one that has the largest depth and denote it by w^c . W.l.o.g. w^c is the lowest common ancestor of w_1 and w_2 .

For guaranteeing property $(C6)_c$ for each proper descendant of w^c , we apply the following *extra rule* when constructing a good characteristic of such a descendant w' .

- If a good characteristic Q of w' is constructed from a good characteristic Q' of a child of w' and if Q' has a macro set S' with $c_{S'}^{Q'} = c$, check if Q has a macro set S with $c_S^Q = c$, too. If not, reject this computation of Q .

Since $(C6)_c$ holds now for all proper descendants of w^c , Obs. 5.6.3 implies that we can solve (iii) for all nodes in T_{w^c} . Let us now consider the modifications when constructing a good characteristic Q of w^c by combining two compatible good characteristics Q_l of w_l^c and Q_r of w_r^c . By Obs. 5.6.3 we can find out which of the vertices u_1 and u_2 are uncolored by a recoloring extending recolorings consistent with Q_l and Q_r by testing which of Q_l and Q_r contains a macro set colored with c .

If both are uncolored, u_3 —if it exists—must be also uncolored by all recolorings with property $(C6)_c$ that extend a recoloring consistent with Q . Therefore, we will run a c -cost adaption.

Next, we consider the case that exactly only one of u_1 and u_2 is colored with c , say u_1 . If $|W| = 3$, the good characteristic Q should contain a macro set S with at least one vertex on a path from u_1 to u_3 colored with c (i.e., $c_S^Q = c$) since $(C6)_c$ requires a connection from u_1 to u_3 . Thus, in this case we reject this computation of Q if Q does not contain a macro set S with $c_S^Q = c$. We also reject if $|W| = 2$. In the case, where $|W| = 3$ and u_1 as well as u_2 are colored, we explicitly take two collections of recolorings C' of G extending recolorings consistent with Q_l and Q_r into account; both collections leading to different characteristics of w^c : The first collection contains all convex colorings that color u_1 , u_2 , and u_3 with c . In this case as usual Q should contain a colored macro set S with $c_S^Q = c$. The second collection contains all colorings that uncolor u_3 . In this case—in opposite to our usual recoloring rules—we define $c_S^Q = -1$ for the macro set S in Q colored with c to denote the fact that the recoloring should not be extended by further vertices initially colored with c . Intuitively, if u_1 and u_2 are colored with c and if we decide that u_3 should not be C' -connected to these vertices, this corresponds to uncoloring u_3 . Then $G[V \setminus B(T_w)]$ would not contain any vertex colored with c and in this case also the old algorithm would use the value -1 . In the case, where $|W| = 2$ and u_1 as well as u_2 are colored, we also set $c_S^Q = -1$ for the macro set S in Q colored with c .

Since we distinguish between the value c and -1 , we can take w^c as c -decision node. It is also easy to see that our modifications at node w^c guarantee that property $(C6)_c$ holds also for w^c .

Moreover, if u_3 exists, let \hat{w} be the lowest common ancestor of w^c and w_3 . For guaranteeing that $(C6)_c$ holds for \hat{w} and its descendants, we apply our extra rule also for the proper descendants of \hat{w} not being contained in T_{w^c} . However, there is one problem if we apply the extra rule for these nodes. Let us consider a good characteristic Q of a node w being a proper ancestor of w^c and an descendant of \hat{w} constructed during the the bottom-up traversal of T from a good characteristic Q' of w^c . Then, if Q' contains a macro set S' with $c_{S'}^{Q'} = c$,

\mathcal{Q} must also contain a macro set S with $c_{S'}^{\mathcal{Q}'} = c$. Note that this may not be necessary to guarantee property $(C6)_c$ if the colorings consistent to \mathcal{Q}' color u_1 and u_2 both with c . This seems to imply that we possibly could lose some good characteristics of \mathcal{Q}' . However, in such a case we also have constructed a good characteristic \mathcal{Q}'' of w^c equal to \mathcal{Q}' except $c_{S'}^{\mathcal{Q}''} = -1$. Good characteristics constructed from \mathcal{Q}'' do not have a macro set colored with c . Indeed this is one further reason for using the value -1 , which here guarantees that we do not lose any good characteristics.

Let \hat{w}_x be the child of \hat{w} such that $T_{\hat{w}_x}$ contains w^c . Since, for all nodes w in the subtree $T_{\hat{w}_x}$ not in T_{w^c} and for a macro set S of a good characteristic \mathcal{Q} of w , $c_S^{\mathcal{Q}} = c$ now means that at least one of the two vertices has to be C' -connected to u_3 for all colorings C' extending a recoloring consistent to \mathcal{Q} —even if u_1 and u_2 are already C' -connected—it is now easy to solve (iii) for all nodes in the subtree rooted in \hat{w} . We also can guarantee property $(C6)_c$ for \hat{w} , if during the construction of a good characteristic \mathcal{Q} of \hat{w} from good characteristics \mathcal{Q}_l and \mathcal{Q}_r of the two children of \hat{w} , we reject this non-deterministic computation of \mathcal{Q} if exactly one of the two good characteristics \mathcal{Q}_l and \mathcal{Q}_r contains a macro set S with $c_S = c$.

Let T^* be the subtree of T rooted in w^c if $|W| = 2$ and rooted in \hat{w} if $|W| = 3$. Then it is easy to see that $(C6)_c$ holds for the nodes outside T^* if it holds for all nodes of T^* . Hence (ii) holds. Finally (iii) holds because we only need to consider the vertices contained in the sets $B(w)$ for the nodes w in T^* for solving (iii).

Case 2: $|U| = 3$, and there is no node of T whose bag contains all vertices in U , but a node whose bag contains two of them. W.l.o.g. we assume that there is a node whose bag contains u_1 and u_2 , and we let \check{w} be the node of smallest depth with this property. We distinguish to subcases

(2a) There is another node whose bag contains another pair of two vertices in $\{u_1, u_2, u_3\}$, say u_2 and u_3 .

(2b) Condition (2a) does not hold.

Case (2b) can be handled very similar as case 1 if we let $w^c = \check{w}$ be the c -decision node. We therefore only consider the case (2a). We now choose \tilde{w} to be the vertex of smallest depth containing both vertices in $\{u_2, u_3\}$ and take \hat{w} as the lowest common ancestor of \check{w} and \tilde{w} .

Note that, for each node w' on the unique \check{w} - \hat{w} -path in T or on the unique \tilde{w} - \hat{w} -path in T , we have $u_2 \in B(w')$. Similar to case 1, since we want to consider only colorings for which property $(C6)_c$ holds, the extra rule is applied to all nodes $w' \notin \{\check{w}, \hat{w}, \tilde{w}\}$ in the subtree of $T_{\hat{w}}$, and the following special rules are applied for the construction of good characteristics of the nodes in $\{\check{w}, \hat{w}, \tilde{w}\}$: Assume we want to construct a good characteristic \mathcal{Q} of $w \in \{\check{w}, \tilde{w}\} \setminus \{\hat{w}\}$ by combining two good characteristics \mathcal{Q}_l of w_l and \mathcal{Q}_r of w_r : Note that we know for each good characteristic of w how their consistent colorings color the vertices of U part of T_w —we simply have to consider the colors of the macro sets containing these vertices of U . If these recolorings color one (none) of the vertices u_1 and u_2 if $w = \check{w}$ or one (none) of u_2 and u_3 if $w = \tilde{w}$, \mathcal{Q} as usual should (not) contain a macro set S with $c_S^{\mathcal{Q}} = c$; if this is not the case, reject the computation of \mathcal{Q} . Otherwise—similar to case 1—we have to distinguish

between the case that all vertices of U should be C' -connected and the case that this is not true. In both cases \mathcal{Q} should contain a macro set colored with c , but in the first case, we set $c_S^{\mathcal{Q}} = c'$, and in the second case, we set $c_S^{\mathcal{Q}} = c''$ for some extra values introduced to distinguish the two cases and the case that only one of the vertices u_1 and u_2 is uncolored. The suspicious reader may have noticed that then beside property $(C6)_c$ also properties $(C6)_{c'}$ and $(C6)_{c''}$ should hold for all $w' \notin \{\hat{w}, \hat{w}, \tilde{w}\}$ in the subtree of $T_{\hat{w}}$. For improving the efficiency, note that it is not really necessary to introduce for each color c its own two extra colors c' and c'' . We just can introduce two global extra colors -2 and -3 since it is clear for the MRRP that the only possible real color for a macro set containing u_2 is c .

When combining good characteristics \mathcal{Q}_l and \mathcal{Q}_r of \hat{w}_l and \hat{w}_r , respectively, to a good characteristic \mathcal{Q} of \hat{w} , it is easy to decide by the modifications above which of the vertices u_1 and u_2 (or u_2 and u_3) are colored by recolorings consistent with \mathcal{Q}_l and \mathcal{Q}_r . Thus, it is easy to determine all good characteristics of \hat{w} with property $(C6)_c$ and to decide whether a c -cost adaption should be done or not. Therefore, we take \hat{w} as the c -decision node.

Case 3: There is a node w^c of T whose bag contains all vertices in U . Then we could choose w^c as the c -decision node. We could also guarantee (ii) and (iii) with arguments similar to that used in the cases 1 and 2. However, it is much easier to give up property $(C6)_c$ in this case since we can easily find out the colors of u_1, u_2, u_3 when constructing a good characteristic of w^c from the good characteristics of the children; i.e., we also do not need the forbidden colors in this case.

It is not hard to implement all changes within polynomial running time.

Theorem 5.6.4. *The MRRP restricted to initial $(\infty, 3)$ -colorings is solvable in polynomial time on graphs for which a tree decomposition (T, B) of bounded width is known.*

Note that the running times for the MCRP and the MRRP on arbitrary initial colorings are polynomial if s , defined as in both Theorems 5.5.8 and 5.6.1, is of size $O(\log n)$. This is the case if an (a, b) -coloring with $a = O(\log n)$ is given.

Theorem 5.6.5. *On graphs for which a tree decomposition (T, B) of bounded width is known, the MCRP and the MRRP are solvable in polynomial time if the initial coloring is an (a, b) -coloring with $a = O(\log n)$.*

5.7 Approximation of MCRP and MRRP

Since the MCRP is NP-hard even on trees, we can not hope for a polynomial-time algorithm that solves the problem to optimality—even if we consider graphs of bounded treewidth. Using the algorithm of Subsection 5.5 we now present $(2 + \epsilon)$ -approximation algorithms for the MCRP and the MRRP for graphs of bounded treewidth with arbitrary (∞, ∞) -colorings. The following algorithm is inspired by the algorithm of Bar-Yehuda et al. [9]. We extend the algorithm from trees to graphs of bounded treewidth and present a slightly different description for proving the correctness of the algorithm.

Given a graph $G = (V, E)$ with a coloring C and a nice tree decomposition (T, B) of width $k - 1$ for G , the results can be obtained by iteratively modifying the coloring C and the weights of the vertices such that finally $|\text{SEP}(C, w)| < s$ for all nodes w of T and a fixed $s \in \mathbb{N}$ with $s > k$. Let w be a node of T such that there is a set $R' \subseteq \text{SEP}(C, w)$ containing exactly s colors and let V' be a set consisting of two vertices of color c for all $c \in R'$ such that, for each pair of vertices $u, v \in V'$ of the same color, the vertices u and v are in different connected components in $G[V \setminus B(w)]$. Moreover, let α be the minimal weight of a vertex in V' . The number of vertices in $\text{SEP}(C, w)$ is decremented by reducing the weight of all vertices in V' by α and subsequently uncoloring the vertices of zero-weight.

On the one hand, this weight reduction decreases the cost of an optimal convex (re-)coloring C' of G by at least $(s - k)\alpha$ since the vertices in $B(w)$ allow to connect at most k of the s colors in R' , i.e., at least $s - k$ vertices in V' must be uncolored. On the other hand, if we have a solution for the MCRP (or the MRRP) with the reduced weight function, we can simply take this solution as a solution for the MCRP (or the MRRP) with the original weights and our cost increases by at most $2s\alpha$. Thus, in each iteration our cost decreases by at most a factor of $2s/(s - k)$ more than the decrease of the cost of an optimal solution. If at the end no further weight reduction is possible, we can use the exact algorithms from the previous section, i.e., we can solve the instance obtained by this weight reduction as good as an optimal algorithm. Altogether, we have only a recoloring cost that is a factor of $2s/(s - k)$ larger than the cost of an optimal solution. Note that this technique above for determining the approximation ratio is the so-called *local ratio technique*. Choosing s large enough, we obtain the following.

Corollary 5.7.1. *On graphs for which a tree decomposition (T, B) of bounded width is known, a $(2 + \epsilon)$ -approximation algorithm exists for the MCRP and the MRRP with quadratic running time.*

Chapter 6

VDPP on Chordal Graphs

6.1 The ℓ -Vertex-Disjoint Paths Problem

As we have already seen from Section 5.1, we can solve the ℓ -vertex-disjoint path problem (ℓ -VDPP) in polynomial time on graphs of bounded treewidth. However, on many graph classes the ℓ -VDPP appears to be a hard problem since efficient algorithms are unknown or do not exist. Indeed, Fortune, Hopcroft, and Wyllie [31] have shown that the problem is NP-hard on directed graphs, even if ℓ is restricted to 2. As shown by Lynch [63] and by Knuth (see the paper of Karp [59]) the same is true on undirected graphs for the VDPP.

It is a common approach in combinatorial optimization to construct fixed-parameter algorithms for NP-hard problems. In this chapter, we see a linear time algorithm for the ℓ -VDPP, which then can be considered also as a fixed parameter algorithm for the VDPP.

For every fixed ℓ , Robertson and Seymour, in their series of papers, developed a polynomial algorithm for the ℓ -VDPP on undirected graphs. Perković and Reed [74] presented an algorithm with an improved running time. Unfortunately, the constants hidden in the O -notation of the running time of the algorithms above are extremely large and make these algorithms unfeasible in practice. Algorithms with better practical running times are known for some classes of graphs such as so-called directed acyclic graphs [31]. However, for many classes of graphs, e.g., for general, for planar, or for chordal undirected graphs, algorithms more efficient than the algorithm of Perković and Reed are known only for the special case $\ell = 2$. The first polynomial-time algorithms for the case $\ell = 2$ on general undirected graphs are given, e.g., in [71, 81, 84, 90]. Algorithms with better asymptotic running times can be found in [60] and [89]. The first linear-time algorithm for the case $\ell = 2$ on planar undirected graphs is the algorithm of Perl and Shiloach [75]. A simpler algorithm was later given by Woeginger [94] and more practical algorithms can be found in [46] and [89]. Perl and Shiloach [75] also presented the first polynomial-time algorithm for the 2-VDPP on undirected chordal graphs and on undirected planar graphs, again with a linear running time. A simpler algorithm for chordal graphs can be found in [62].

In the next section, an algorithm is presented for solving the ℓ -VDPP on a chordal graph with n vertices and m edges in a running time of $O(n^{2\ell+2} + m)$.

As shown in Section 6.3, this algorithm can be modified to connect given pairs of vertices by pairwise disjoint paths such that the number of edges used by the paths is minimized among all such solutions.

In Section 6.4, a combination of the tree decomposition based algorithm of Section 6.2 with a sparsification technique reduces the running time for solving the ℓ -VDPP on chordal graphs to $O(m + (2\ell)^{4\ell+2}n)$. This means that we obtain a linear fixed parameter algorithm for the VDPP. Moreover, the additional constants hidden in the O -notation are of moderate size. The algorithm is easy to implement and, for small values of ℓ , it is practical. It is no surprise that the running time increases exponentially in ℓ since the vertex-disjoint path problem (with ℓ being non-fixed) is NP-hard even for chordal graphs. Details can be found in Section 6.5.

6.2 A Simple Approach for the ℓ -VDPP

Similar to the algorithm for the maximum independent set problem on chordal graphs in Section 1.2, we can use a weak clique tree for solving the ℓ -VDPP. Let $G = (V, E)$ be a chordal graph with treewidth $k \in \mathbb{N}$. Moreover, let (T, B) be a weak clique tree of size $O(|V| + |E|)$ for G of width k with T being a binary rooted tree having $O(|V|)$ nodes—such a clique tree can be found in linear time (Corollary 2.3.10). We call the vertices to be connected by vertex-disjoint paths the *terminals* of G . In order to obtain a simpler description of our algorithm, we describe our problem as a recoloring problem. For an instance of the ℓ -VDPP, we always define an *initial coloring* that, for each pair of terminals to be connected, colors both terminals with the same color different from the colors of the other terminals. Let C be our initial coloring in the following. Recall that a recoloring of (G, C) is called a restricted convex recoloring of zero cost only if it does not recolor initial colored vertices—i.e., the terminals—and if, for each pair t_1 and t_2 of terminals sharing a color c , there is a path from t_1 to t_2 in the subgraph of G induced by the vertices of color c . We call a recoloring *benign* with respect to a weak clique tree if it is a restricted convex recoloring of zero cost and if each bag of the clique tree contains at most two vertices of the same color. For a node w of T , recall that $G[B(T_w)]$ is the subgraph of G induced by all vertices contained in at least one bag of a descendant of w . In addition, for a subgraph H of G , we define $C|_H$ as the coloring C restricted to H .

For a node w of T with a parent w' , let us call the set of vertices in $B(w) \cap B(w')$ the *transition set* of w denoted by $A(w)$. The reason for introducing in the following two kinds of characteristics is that our algorithm needs to know the colors of all vertices in a bag for finally obtaining a complete coloring of G as well as our algorithm extends colorings of $G[B(T_w)]$ to colorings of $G[B(T_{w'})]$ and for determining the colors for the new vertices of this extension (namely the vertices in $B(T_{w'}) \setminus B(T_w)$), it only needs to know the colors of the vertices in $A(w)$. Thus, we define a *full* and a *reduced characteristic* for a node w of T as a coloring of the nodes in $B(w)$ and $A(w)$, respectively, such that for each color c at most two vertices are colored with c . Let \mathcal{Q} be a full or a reduced characteristic of a node w . Then \mathcal{Q} is *valid* if and only if

1. there exists a benign recoloring C' of $(G[B(T_w)], C|_{G[B(T_w)]})$ extending \mathcal{Q} .

2. for each color c the following is true: If there is exactly one terminal in $G[B(T_w)]$ of color c , a vertex in $A(w)$ is colored with c by \mathcal{Q} .

A recoloring C' with properties 1 and 2 is called *suitable* to \mathcal{Q} . We also call two characteristics *compatible* if one characteristic is an extension of the other.

There is a connection between the ℓ -VDPP and benign recolorings: If the ℓ -VDPP has a solution, take a solution with minimal total length. Then a recoloring that colors the vertices of each path of the solution with the initial color of its endpoints is a benign recoloring because the following is true: if one of the vertex-disjoint paths visits three vertices v_1, v_2 , and v_3 of one bag in this order, we obtain a shorter path by replacing the subpath from v_1 to v_3 by the edge $\{v_1, v_3\}$. For the converse assume that a benign recoloring is given such that it is suitable to a valid full characteristic of the root of T . Then the vertex-disjoint paths connecting the terminals of the same color can be easily obtained by a depth-first search on each subgraph induced by the vertices of one color. Hence, the idea is to solve the ℓ -VDPP by computing a benign recoloring suitable to a valid full characteristic of the root of T —details are described next.

For all nodes of T , we next want to determine bottom-up all valid full and all valid reduced characteristics. If we restrict a recoloring of $(G[B(T_w)], C|_{G[B(T_w)]})$ suitable to a valid full characteristic of a node w to the graph $G[B(T_{w'})]$ for a descendant w' of w , this recoloring is suitable to a valid full as well as to a valid reduced characteristic of w' . In the reverse direction, a full characteristic \mathcal{Q} of a node w is valid if and only if

- the terminals in $B(w)$ are colored by \mathcal{Q} with their original color,
- for each child w' of w , the reduced characteristic of w' compatible to \mathcal{Q} is valid,
- each color is used by \mathcal{Q} to color at most two vertices, and
- for each color c , the following is true: if there is exactly one terminal in $G[B(T_w)]$ of color c , a vertex in $A(w)$ is colored with c by \mathcal{Q} .

The four conditions above imply that there exists a benign recoloring of $(G[B(T_w)], C|_{G[B(T_w)]})$ extending \mathcal{Q} ; in particular, using the last condition we can conclude by induction that, for each pair of terminals t_1 and t_2 sharing a color c , there is a path from t_1 to t_2 in the subgraph of G induced by the vertices of color c . By iterating over all valid full characteristics of a node w in T we can easily compute a lookup-table storing 1 for each valid reduced characteristic of w and 0 for each non-valid reduced characteristic of w . Hence, by a subsequently top-down traversal of T we can find a benign recoloring of (G, C) if such a recoloring exists.

We next analyze the running time of our algorithm. Since T is binary, for each node of T , we can test whether a certain full characteristic is valid in at most $O(k + \ell)$ time by testing the four properties listed above. For testing the last condition, note that, in $O(\ell)$ time, we can update the set of colors c for which there is exactly one terminal of color c in $G[B(T_w)]$ if the corresponding sets for the children of w are given. Concerning the number of full characteristics for a node w of T with $q = |B(w)|$, we can bound this number by $q^{2^{\ell-1}}(q + \ell) = O((k + 1)^{2^{\ell-1}}(k + \ell))$: Let c be a fixed color. Then the number of full characteristics using c can be bounded by q^{2^ℓ} since we can select for color c

either one or two vertices out of $B(w)$ (at most q^2 possibilities) and, for each color $c' \neq c$, zero to two vertices out of at most $q - 1$ vertices (again at most q^2 possibilities). The number of full characteristics without color c can be bounded by $q^{2\ell-2}\ell$ since there are at most ℓ possibilities to choose a color for a fixed vertex of $B(w)$ and since subsequently, for each color $c' \neq c$, zero to two vertices out of at most $q - 1$ vertices are selectable. Finally note that the time needed to initialize the lookup-table for the reduced characteristics of a node w is bounded linear in $O(|A(w)|) = O(k)$ times the number of full characteristics.

Lemma 6.2.1. *The ℓ -VDPP can be solved in $O((k+1)^{2\ell-1}(k+\ell)^2|V|+|E|) = O(|V|^{2\ell+2} + |E|)$ time on a chordal graph $G = (V, E)$ with treewidth k .*

6.3 Shortest ℓ -Vertex-Disjoint Paths

The algorithm of the last section can be modified to find paths solving an instance I of the ℓ -VDPP on G such that the total length of the paths—i.e., the number of edges visited by the paths—is minimized among all solutions for I . Let G be a chordal graph, and let C be an initial coloring of G defining the terminals of the ℓ -VDPP. We only have to store one additional value with each characteristic called the *weight* of the characteristic. Let us first define the weight of a benign recoloring of a colored subgraph $(H, C|_H)$ of (G, C) as the number of edges in H whose endpoints are both colored with the same color. Then the weight $W(\mathcal{Q})$ of a characteristic \mathcal{Q} of a node w is the minimal possible weight of a benign recoloring of $(G(w), C|_{G(w)})$ extending \mathcal{Q} .

Let w be a node of T and \mathcal{Q} be a full characteristic of w . If w has $j \in \{1, 2\}$ children w_1, \dots, w_j for which the weights $W(\mathcal{Q}_1), \dots, W(\mathcal{Q}_j)$ stored with the reduced characteristics $\mathcal{Q}_1, \dots, \mathcal{Q}_j$ compatible to \mathcal{Q} are given, then one can show by induction that the correct value of $W(\mathcal{Q})$ can be computed as follows: We initialize $W(\mathcal{Q})$ with $W(\mathcal{Q}_1) + \dots + W(\mathcal{Q}_j)$. Afterwards, for each color c used by \mathcal{Q} to color two vertices $v_1, v_2 \in B(w)$, we add one minus the number of children of w with their bags also containing both, v_1 and v_2 .

Note that the extra running time needed for computing the additional values for one full characteristic can be bounded by $O(\min\{\ell, |B(w)|\})$ and therefore cannot increase the asymptotic running time of the algorithm of the last section—remember that there are at most two vertices in $B(w)$ colored with one color. By stepping through all full characteristics, it is easy to compute $W(R)$ for all reduced characteristics R of w in a time linear in the number of all full characteristics for w . Finally at the root of T , we can read off the minimal possible weight of a benign recoloring of (G, C) which is equal to the length of ℓ pairs of vertex-disjoint paths solving our instance of the ℓ -VDPP and having minimal total length. The paths themselves can be easily computed by an additional top-down traversal of T .

Theorem 6.3.1. *For an instance of the ℓ -VDPP on a chordal graph $G = (V, E)$, one can find a set of ℓ paths in $O(|V|^{2\ell+2} + |E|)$ time such that these paths use a minimal number of edges among all sets of paths solving the instance.*

6.4 A Speedup for the ℓ -VDPP

In this section we present a speed-up of the algorithm in Section 6.2. Once again, for our input graph $G = (V, E)$, we first construct in $O(|V| + |E|)$ time a weak clique tree (T, B) of size $O(|V| + |E|)$ with T being a binary tree with $O(|V|)$ nodes. We assume that there is no edge $\{t_1, t_2\}$ in G for a pair $\{t_1, t_2\}$ of terminals that are to be connected in G . Otherwise our problem would be reduced to the $(\ell - 1)$ -VDPP on $G[V - \{t_1, t_2\}]$. For each pair $\{t_1, t_2\}$ of terminals that are to be connected by a path, let us choose $\Gamma(t_1)$ and $\Gamma(t_2)$ as the unique pair of nodes in T with their bags containing t_1 and t_2 , respectively, such that the distance between the nodes is minimal. Let C be the initial coloring of the given ℓ -VDPP instance. We choose for an arbitrary terminal t the node $\Gamma(t)$ as the root of T . Let f be a fixed bijection from V to $\{1, \dots, |V|\}$ assigning the highest numbers to the terminals of G . For nodes w_1 and w_2 of T , for a function \tilde{B} defining bags for the nodes of T , and for a vertex v of G , we define the $(w_1, w_2)_{\tilde{B}}$ -count of v as a tuple $(p, f(v))$, where p is the number of nodes w' on the path from w_1 to w_2 in T whose bags $\tilde{B}(w')$ contain v . We say that a vertex v_1 with $(w_1, w_2)_{\tilde{B}}$ -count (p_1, q_1) has a larger $(w_1, w_2)_{\tilde{B}}$ -count than a vertex v_2 with $(w_1, w_2)_{\tilde{B}}$ -count (p_2, q_2) if and only if either $p_1 = p_2$ and $q_1 > q_2$ or $p_1 > p_2$ holds. For a node $w \in T$, we let $I(w) = \{t \mid t \text{ is terminal with } \Gamma(t) \text{ contained in } T_w\}$.

In order to improve the efficiency of the algorithm presented in Section 6.2, we replace (T, B) by a new tuple (T^*, B^*) , where T^* will be a subtree of T and where B^* will be a function that maps each node w of T^* to a subset of $B(w)$ of size $\leq 4\ell^2$. In order to describe (T^*, B^*) more precisely, we need some further definitions. A bag $B(w)$ of a node w is called *small* if $|B(w)| \leq 2\ell$ and *big* otherwise. For each node w of T and for each terminal $t \in I(w)$, we define $D(w, t)$ as the set of the $\min\{2\ell, |B(w)|\}$ vertices of $B(w)$ with the largest $(w, \Gamma(t))_B$ -count. We also let $D(w)$ be the union of $D(w, t)$ over all $t \in I(w)$ and of the set of all terminals in $B(w) \setminus I(w)$.

We now obtain T^* from T by deleting all nodes w with $I(w) = \emptyset$. We choose the same root for T^* as for T and, for each node w , we insert the vertices of $D(w)$ into $B^*(w)$. Moreover, for each child w' of w , we insert an arbitrary subset of $D(w) \cap B(w')$ of size $\min\{2\ell, |D(w) \cap B(w')|\}$ into $B^*(w')$. Let $t \in I(w')$. Keep in mind that, if $|B(w) \cap B(w')| \geq 2\ell$, then $D(w, t)$ —and consequently also $B^*(w)$ —contains 2ℓ vertices of $B(w')$ since these vertices have the largest $(w, \Gamma(t))_B$ -count. Thus, if $|B(w) \cap B(w')| \geq 2\ell$, the rules for node w add 2ℓ vertices of $B^*(w)$ to $B^*(w')$, i.e., $|B^*(w) \cap B^*(w')| \geq 2\ell$. Note that by our definition $B(w) = B^*(w)$ holds for each small bag $B(w)$. We also can conclude:

Lemma 6.4.1. *Let v be a vertex of G , w_1 be a node of T^* with $v \in B^*(w_1)$, and w_2 be the node of lowest depth with $v \in B(w_2)$. Then $v \in B^*(w)$ holds for each node w on the path from w_1 to w_2 in T^* .*

Proof. Since $B(w)$ and $B^*(w)$ share the same terminals, Lemma 6.4.1 holds if v is a terminal. Otherwise, we merely need to show that, for each node w on the path from w_1 to w_2 in T with $v \in B^*(w)$, the following holds: if w has a parent w' with $v \in B(w')$, we also have $v \in B^*(w')$. Since $B^*(w') = B(w')$ if $B(w')$ is small, we only need to consider the case, where $B(w')$ is big. Let us consider the case, where $B(w)$ is small. Because of $v \in B^*(w)$, there is a terminal $t \in I(w)$ for which $v \in D(w, t)$ or $v \in D(w', t)$ holds. Since $|B(w)| \leq 2\ell$, $v \in D(w, t) \cap B(w')$

also implies $v \in D(w', t)$. Consequently, $v \in B^*(w')$. Let us finally consider the case where both, $B(w)$ and $B(w')$, are big. If the insertion rule for w' inserts v into $B^*(w)$, we have $v \in B^*(w')$. Otherwise the only reason for v being contained in $B^*(w)$ is that $v \in D(w, t)$ for a terminal $t \in I(w)$. Then v must also be one of the vertices with the 2ℓ largest $(w', \Gamma(t))_B$ -count and therefore is also contained in $B^*(w')$. \square

Corollary 6.4.2. *For each vertex v of G , the subtree of T^* induced by the nodes w with $v \in B^*(w)$ is connected.*

Let G^* be the graph obtained from G by removing all vertices v and all edges $\{v_1, v_2\}$ from G for which there is no longer a node w with $v \in B^*(w)$ and $\{v_1, v_2\} \subseteq B^*(w)$, respectively.

Lemma 6.4.3. *(T^*, B^*) is a weak clique tree for G^* of width at most $4\ell^2 - 1$.*

Proof. By our construction and Corollary 6.4.2 all properties of a weak clique tree hold for (T^*, B^*) . Concerning the treewidth, for the root r of T , we have $|B^*(r)| \leq |D(r)| \leq 4\ell^2$ since $|I(r)| = 2\ell$. By our choice of r there is a terminal t_1 with $\Gamma(t_1) = r$. Thus, we have $|D(w)| \leq 4\ell^2 - 2\ell$ for all nodes $w \neq r$ in T since the subtree of T rooted in w does not contain $\Gamma(t_1)$. Consequently, $|B^*(w)| \leq 4\ell^2$. \square

Lemma 6.4.4. *An instance of the ℓ -VDPP has a solution on G if and only if this is true for the instance of the ℓ -VDPP on G^* with the same terminals.*

Proof. Clearly, an instance of the ℓ -VDPP is solvable on G if this true for G^* . For the converse we merely need to show that a solution of the ℓ -VDPP on G allows us to construct a coloring of G^* with respect to (T^*, B^*) . First of all, for a restricted convex recoloring C' of (G, C) and for a pair of terminals t_1 and t_2 colored with c by C' , let us call a pair of incident nodes w_1 and w_2 on the unique path from $\Gamma(t_1)$ to $\Gamma(t_2)$ a *color break* with respect to c (and C') if no vertex in $B^*(w_1) \cap B^*(w_2)$ is colored with c . Let \mathcal{C} be the set of all restricted convex recolorings of (G, C) that color at most two vertices of each bag in (T^*, B^*) with the same color. The solvability of the ℓ -VDPP implies $\mathcal{C} \neq \emptyset$ since there exists—as shown in Section 6.2—at least one benign recoloring with respect to (T, B) and since each benign recoloring is contained in \mathcal{C} . It remains to show that there is a $C' \in \mathcal{C}$ without any color breaks since C' then is a benign recoloring of G^* with respect to (T^*, B^*) .

Assume now that we can find no recoloring in \mathcal{C} without color breaks. Let us choose a fixed numbering with $1, \dots, \ell$ for the colors assigned to the terminals and a recoloring $C' \in \mathcal{C}$ such that the smallest number among the colors with a color break is as large as possible. Moreover, if c is the color with the smallest number for which there is a color break and if t_1 and t_2 are the terminals of color c , we choose the recoloring $C' \in \mathcal{C}$ with the above properties such that there is a maximal distance between $\Gamma(t_1)$ and the node w_{σ_0} of the first color break $(w_{\sigma_0}, w_{\sigma_0+1})$ on the path $P = (w_1, w_2, w_3, \dots)$ from $\Gamma(t_1)$ to $\Gamma(t_2)$ in T . Let v be a vertex of color c with $v \in B(w_{\sigma_0}) \cap B(w_{\sigma_0+1})$ and $v \notin B^*(w_{\sigma_0}) \cap B^*(w_{\sigma_0+1})$.

Assume $|B^*(w_{\sigma_0}) \cap B^*(w_{\sigma_0+1})| < 2\ell$. Then $|B(w_{\sigma_0}) \cap B(w_{\sigma_0+1})| < 2\ell$. Let $w \in \{w_{\sigma_0}, w_{\sigma_0+1}\}$ be the parent of the other node $w' \in \{w_{\sigma_0}, w_{\sigma_0+1}\}$ and $t \in \{t_1, t_2\} \cap I(w')$. We can conclude $v \in D(w, t)$ and consequently $v \in B^*(w)$.

Since $|B(w_{\sigma_0}) \cap B(w_{\sigma_0+1})| < 2\ell$, the rule for w adds all vertices of $D(w, t) \cap B(w')$ including v into $B^*(w')$, a contradiction to $v \notin B^*(w_{\sigma_0}) \cap B^*(w_{\sigma_0+1})$.

Hence $|B^*(w_{\sigma_0}) \cap B^*(w_{\sigma_0+1})| \geq 2\ell$. Since no vertex in $B^*(w_{\sigma_0}) \cap B^*(w_{\sigma_0+1})$ is colored with c and since C' is a recoloring which uses each color at most twice in a bag of (T^*, B^*) and thus also in $B^*(w_{\sigma_0}) \cap B^*(w_{\sigma_0+1})$, it follows that there must be an uncolored vertex in $B^*(w_{\sigma_0}) \cap B^*(w_{\sigma_0+1})$. Let us define u to be the uncolored vertex in $B^*(w_{\sigma_0}) \cap B^*(w_{\sigma_0+1})$ that has the largest $(w_{\sigma_0+1}, \Gamma(t_2))_B$ -count among all uncolored vertices. Keep in mind that, if w_{σ_0} is the parent of w_{σ_0+1} , $u \in D(w_{\sigma_0+1}, t_2)$ since there are at most $2\ell - 1$ colored vertices in $B^*(w_{\sigma_0}) \cap B^*(w_{\sigma_0+1})$. We next show that we can construct a recoloring $C^* \in \mathcal{C}$ without any new color breaks for the colors different from c for which—if it has a color break with respect to c —the first such color break occurs after the pair $\{w_{\sigma_0}, w_{\sigma_0+1}\}$ on P . This leads to a contradiction to our choice of C' and proves our lemma.

After setting initially $C^* = C'$ we modify C^* . First of all, we color u with c . We then define $w_{\sigma-1}$ and w_{σ_1} as the first and the last node on P , respectively, such that $u \in B^*(w_{\sigma-1}) \cap B^*(w_{\sigma_1})$. Let S be the set consisting of u and all vertices colored with c by C' contained in a bag of $\{B(w_{\sigma-1}), \dots, B(w_{\sigma_1})\}$.

Second, modify C^* as follows: For the set X of all nodes reachable from one of the nodes in $\{w_{\sigma-1}, \dots, w_{\sigma_1}\}$ without visiting $w_{\sigma-1-1}$ or w_{σ_1+1} , uncolor all vertices in $S \cap B^*(X) \setminus \{u, v', v''\}$, where v' is the vertex with the largest $(w_{\sigma-1}, \Gamma(t_1))_{B^*}$ -count in $S \cap B^*(w_{\sigma-1})$ and where v'' is the vertex with the largest $(w_{\sigma_1}, \Gamma(t_2))_B$ -count in $S \cap B(w_{\sigma_1})$. From the fact that $(w_{\sigma_0}, w_{\sigma_0+1})$ is the first color break on P and that C' is a restricted convex recoloring, we can conclude that $v' \neq u$. The situation described above is sketched in Fig. 6.4.1.

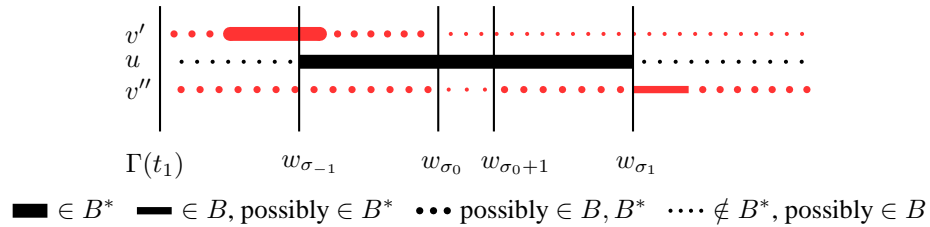


Figure 6.4.1: The figure shows the ranges of the variables in the case where no extra modifications are applied. Black horizontal lines represent vertices uncolored by C' .

Third, test whether $v'' \in B(w_{\sigma_0+1}) \setminus D(w_{\sigma_0+1}, t_2)$ is true. In this case, no vertex is colored with c by C' in $D(w_{\sigma_0+1}, t_2)$ since we have chosen v'' as the c -colored vertex of $S \cap B(w_{\sigma_1})$ with the largest $(w_{\sigma_1}, \Gamma(t_2))_B$ -count and since $v'' \in B(w_{\sigma_0+1})$ then implies that v'' also has the largest $(w_{\sigma_0+1}, \Gamma(t_2))_B$ -count under all vertices in S . Thus, there exists a vertex in $D(w_{\sigma_0+1}, t_2)$ not colored by C' . Take \tilde{u} as such a vertex of largest $(w_{\sigma_0+1}, \Gamma(t_2))_B$ -count. Clearly, $\tilde{u} \neq v''$. Let w_{σ_2} be the last node on P with $\tilde{u} \in B(w_{\sigma_2})$. Consequently, from $D(w_{\sigma_0+1}, t_2) \cap \{v'', \tilde{u}\} = \{\tilde{u}\}$ we can conclude that w_{σ_2} is on the subpath of P from w_{σ_1} to $\Gamma(t_2)$ and that the $(w_{\sigma_0+1}, \Gamma(t_2))_B$ -count of v'' is smaller than that of \tilde{u} . Moreover, let v''' be the vertex in $B(w_{\sigma_2})$ with $C'(v''') = c$ that has the largest $(w_{\sigma_2}, \Gamma(t_2))_B$ -count among all such vertices. In particular,

$v''' \in B(w_{\sigma_2+1})$ or we have $w_{\sigma_2} = \Gamma(t_2)$ and $v''' = t_2$. See Fig. 6.4.2. If additionally to $v'' \in B(w_{\sigma_0+1}) \setminus D(w_{\sigma_0+1}, t_2)$ the condition $v'' \neq u \neq \tilde{u}$ holds, so-called *extra modifications* of C^* are required: For the set Y of all nodes reachable from one of the nodes in $\{w_{\sigma_1}, \dots, w_{\sigma_2}\}$ without visiting w_{σ_1-1} or w_{σ_2+1} , change C^* by uncoloring v'' and all other vertices in $B^*(Y) \setminus \{u, v''\}$ and by coloring \tilde{u} with c .

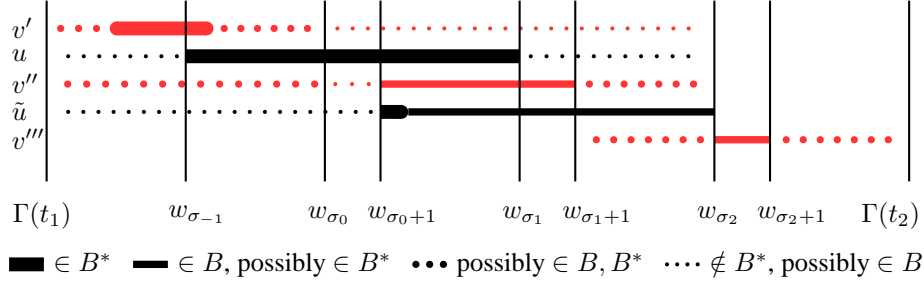


Figure 6.4.2: The figure shows the ranges of the variables in the case where $\Gamma(t_2) \neq w_{\sigma_2}$ and the extra modifications are applied. Black horizontal lines represent vertices uncolored by C' .

For analyzing the coloring C^* , first of all note that coloring v' with c guarantees that there is no new color break between $\Gamma(t_1)$ and w_{σ_0} on P .

Let us first consider the case, where no extra modifications are applied. Even if we uncolor $S \setminus \{u, v', v''\}$, by coloring u, v' and v'' with c we can guarantee that C^* is a restricted convex recoloring of (G, C) . Hence, if C^* does not belong to \mathcal{C} , $|\{u, v', v''\}| = 3$ and there is a node w whose bag $B^*(w)$ contains u, v', v'' . Due to our choice of v' we know that $v' \in B^*(w_{\sigma-1})$. Since $v' \notin B^*(w_{\sigma_0}) \cap B^*(w_{\sigma_0+1})$, from Corollary 6.4.2 follows that $w \neq w_{\sigma_0+1}$ and that w is reachable from $\Gamma(t_1)$ in T without using w_{σ_0+1} . Note that $v'' \in B^*(w)$ implies $v'' \in B(w)$. We consider two subcases:

- w_{σ_0+1} is the parent of w_{σ_0} . Since $v'' \in B(w_{\sigma_1})$, we also have $v'' \in B(w_{\sigma_0}) \cap B(w_{\sigma_0+1})$. Then, $v'' \in B^*(w_{\sigma_0}) \cap B^*(w_{\sigma_0+1})$ according to Lemma 6.4.1, and we obtain a contradiction since $(w_{\sigma_0}, w_{\sigma_0+1})$ is a color break.
- w_{σ_0} is the parent of w_{σ_0+1} . Since no extra modifications are applied and since $|\{u, v', v''\}| = 3$, i.e., $u \neq v''$ holds, we have $u = \tilde{u}$ or $v'' \notin B(w_{\sigma_0+1})$ or $v'' \in D(w_{\sigma_0+1}, t_2) \subseteq B^*(w_{\sigma_0+1})$. In the second case, v'' can not be part of $B(w)$ and $B(w_{\sigma_1})$. In the third case, again by Corollary 6.4.2 we can conclude that $v'' \in B^*(w_{\sigma_0}) \cap B^*(w_{\sigma_0+1})$. If only the first case holds, because of $u \in D(w_{\sigma_0+1}, t_2)$ and $v'' \notin D(w_{\sigma_0+1}, t_2)$, vertex u has a larger $(w_{\sigma_1}, \Gamma(t_2))_B$ -count than v'' . Thus, a contradiction occurs in each case.

Let us finally assume that the extra modifications are applied. Then $u \neq \tilde{u}$ and the definition of u implies $\tilde{u} \notin B^*(w_{\sigma_0})$. Like in the previous case without extra modifications, v' can only be contained in one of the bags $B^*(w)$ with w being reachable from $\Gamma(t_1)$ without using w_{σ_0+1} . Therefore, no bag of (T^*, B^*) contains v' and \tilde{u} . Recall that the $(w_{\sigma_0+1}, \Gamma(t_2))_B$ -count of \tilde{u} is larger than that of v'' . Because of this fact and because of $v'' \in B(w_{\sigma_0+1}) \cap B(w_{\sigma_1})$, the

$(w_{\sigma_1}, \Gamma(t_2))_B$ -count of \tilde{u} is larger than that of v'' . Assume for the moment that $v''' \in B^*(w_{\sigma_1})$. Then $v''' \in B(w_{\sigma_1})$ and the fact that $v''' \in B(w_{\sigma_2+1})$ or $v''' = t_2$ implies that v''' has a larger $(w_{\sigma_1}, \Gamma(t_2))_B$ -count than that of \tilde{u} and by transitivity then that of v'' . In particular, $v'' \neq v'''$. Since by our assumption also $v''' \in S$ holds, we obtain a contradiction to our choice of v'' . Consequently, no bag of (T^*, B^*) contains u and v''' , and C^* colors at most two vertices in each bag of (T^*, B^*) . Our choice of v''' guarantees that C^* is a restricted convex recoloring.

We have shown that $C^* \in \mathcal{C}$ and that the distance between $\Gamma(t_1)$ and the first node of a color break on P with respect to c and C^* —if indeed there is a color break—is larger than the corresponding distance for C' . This is a contradiction to our choice of C' . \square

We can therefore solve the ℓ -VDPP on a chordal graph G with n vertices and m edges as follows: In a preprocessing step, we determine for each node w of T the set $I(w)$ and subsequently all sets $D(w, t)$ and $D(w)$ with $t \in I(w)$. For an efficient computation of $D(w, t)$, initially compute in $O(n + m)$ time for each tuple (w, w') of adjacent nodes of T the set $\Delta(w, w') = B(w) \setminus B(w')$ —recall that the size of (T, B) is $O(n + m)$. Then, for each terminal t , let us consider the tree T_t obtained from T by making $\Gamma(t)$ the root of T_t . By a subsequent top-down traversal in T_t , for each vertex v of G , compute $d(v)$ as the smallest depth of a node w with $v \in B(w)$ in T_t . In detail, if we visit a node w with father w' , we set $d(v)$ to the depth of w exactly for all $v \in \Delta(w', w)$. Thus, for each w in T , $D(w, t)$ consist of the $\min\{2\ell, |B(w)|\}$ vertices v' in $B(w)$ with largest $(-d(v'), f(v'))$ tuple. It is not hard to see that using the Δ -values, we can compute the d -values for each fixed terminal t in $O(n)$ time since the sum of the cardinalities of the considered sets $\Delta(w, w')$ is $O(n)$. Consequently, for each fixed terminal t , we can compute $D(w, t)$ for all nodes w of T in $O(\ell n)$ time. Thus, the preprocessing step takes $O(m + \ell^2 n)$ time. Hence, we can replace G and (T, B) by G^* and (T^*, B^*) in the same running time. We then apply the algorithm of the Section 6.2 on G^* . From Lemma 6.2.1 we can conclude the following:

Theorem 6.4.5. *The ℓ -VDPP on chordal graphs with n vertices and m edges can be solved in $O(m + (4\ell^2)^{2\ell+1}n) = O(m + (2\ell)^{4\ell+2}n)$ time.*

6.5 Hardness of the VDPP

As shown, on a chordal graph $G = (V, E)$, the ℓ -vertex-disjoint path problem for fixed ℓ is solvable in $O(|V|)$ time. However, the constant hidden in the O -notation grows exponentially with ℓ . Unless $P \neq NP$, there is no hope to obtain a running time polynomial in ℓ as we can conclude from the next theorem.

Theorem 6.5.1. *The vertex-disjoint paths problem on chordal graphs is NP-hard.*

Proof. We can prove the theorem by a reduction from a restricted case of the so-called 1-in-3 SAT. In 1-in-3 SAT we are given a Boolean formula in conjunctive normal form with 3 variables per clause and we have to find an assignment of the variables such that exactly one of the three literals is true in every clause of the formula. A Boolean formula in conjunctive normal form

is monotone if every literal is positive and it is cubic if every variable occurs exactly three times. It is shown in [67] that 1-in-3 SAT is NP-complete even on monotone and cubic Boolean formulas. We now reduce an instance of 1-in-3 SAT consisting of a monotone and cubic Boolean formula F to an instance of the VDPP on a chordal graph G . Fig. 6.5.1 shall represent a clique tree of G . Each subgraph induced by the vertices of a bag should be a clique whose edges are colored gray—however, not all existing edges are shown in the figure. Black lines represent paths of length 0. Therefore, the endpoints of black lines represent the same vertex even if they appear in different shapes.

In detail, we construct G as follows: For each variable x and each clause C in F , we introduce a *variable gadget* and a *clause gadget*, respectively, as shown on the left side of Fig. 6.5.1. A variable gadget has six terminals $a_1, a_2, a_3, b_1, b_2, b_3$ and a clause gadget six terminals $y_1, y_2, y_3, z_1, z_2, z_3$. Each gadget is connected to one large clique Γ —see the rightmost bag in Fig. 6.5.1. Γ contains $6q$ vertices where q is the number of clauses of F . We next divide the terminals into pairs such that the resulting instance of the VDPP has a solution if and only if F has a satisfying assignment.

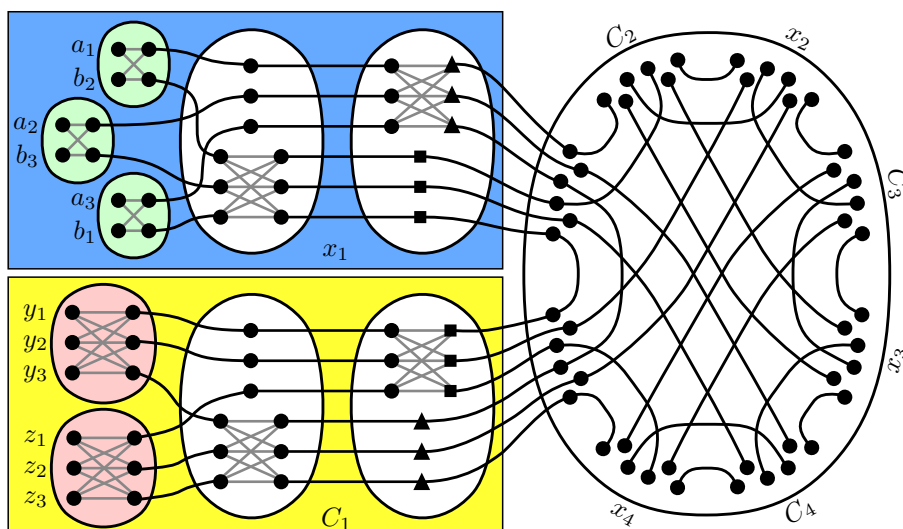


Figure 6.5.1: Sketch of a reduction to the disjoint paths problem from a Boolean formula $(x_1 \vee x_2 \vee x_4) \wedge (x_1 \vee x_2 \vee x_3) \wedge (x_2 \vee x_3 \vee x_4) \wedge (x_1 \vee x_3 \vee x_4)$.

If a clause C contains a variable x as the i -th variable and if it is the j -th occurrence of variable x in F that is part of C , the pairs (a_j, y_i) and (b_j, z_i) are part of our instance of the VDPP, where the four terminals a_j, b_j, y_i , and z_i belong to the gadgets for x and C . Moreover, we identify one triangular and one square vertex in the gadget of x with one triangular and one square vertex, respectively, in the gadget of C different from the triangular and square vertices chosen for other variables or clauses. For a simpler notation, the terminals a_1, a_2 and a_3 shown in Fig. 6.5.1 are called *A-terminals* and the remaining terminals *B-, Y- and Z-terminals*, respectively.

Let us consider a satisfying assignment of F . For each variable x , we construct six paths from the terminals to the triangular and square vertices in the gadget of x such that, if x is set to true, the paths starting in the A -terminals are only routed through the triangular vertices; otherwise they are only routed through the square vertices. Since each clause C has exactly one true variable, for each clause gadget, exactly one path from an A -terminal to the gadget of C passes at a triangular vertex, whereas the other two paths from an A -terminal to the gadget of C pass at a square vertex. Thus, we can forward the three paths to the Y -terminals of C . Similarly, we can forward the paths from the B -terminals to the Z -terminals. Hence, we have found a solution to our instance of the VDPP.

Let us now consider a solution for our VDPP. It remains to show that F can be satisfied. In our construction, the number of vertices in the large clique Γ is equal to the number of pairs that have to be connected in our instance. Moreover, each path has to use at least one and therefore exactly one vertex of Γ . Note that for each clause C and each variable x in C , Γ contains exactly two vertices common with the gadgets of C and x , respectively: one triangular and one square vertex. Thus, these two vertices must be the two vertices visited by the two paths connecting two pairs of terminals in the gadgets of C and x . As a consequence, for any fixed variable gadget the paths starting from the A -terminals must pass either exclusively through the triangular vertices or exclusively through the square vertices of this gadget—to see this, it might help to look at the order of the terminals in the variable gadget. We define a variable x of F to be true if the paths ending in the A -terminals of the gadget of x pass through triangular vertices. Since the A -terminals are connected to the Y -terminals, exactly one path ending in an A -terminal uses a triangular vertex of each clause gadget, i.e., exactly one variable of each clause is set to true. \square

Chapter 7

Generalization of Chordal Graphs

7.1 Motivation

As we have seen in the previous chapters, complexity parameters such as the treewidth can help to solve many NP-hard problems of theoretical or practical importance on a subclass of instances for which the parameter is very small. Although chordal graphs are with their clique tree already generalizations of trees, we next generalize chordal graphs by introducing new complexity parameters.

Definition 7.1.1 (intersection graphs). Let Q be a type of objects for which we can define an intersection—e.g., disk or square. An *intersection graph* of Q is the graph G defined by a set S of objects of type Q such that each object of S corresponds to a vertex in G , and there is an edge between the corresponding vertices of G if and only if two objects of S have a non-empty intersection. An intersection graph of Q is also called a Q *graph*—see for an example Fig. 7.1.1.

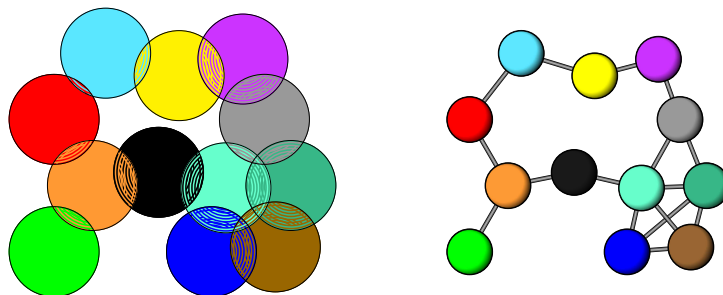


Figure 7.1.1: On the left side we see several disks in the plane. The right side shows the corresponding disk graph G . Note that G has several other embeddings.

For a better understanding of the generalization of chordal graphs, note that chordal graphs are exactly the intersection graphs of subtrees of a suitable tree

T if the intersection of two subtrees $T_1 = (V_1, E_1)$ and $T_2 = (V_2, E_2)$ is defined as the possibly empty tree $(V_1 \cap V_2, E_1 \cap E_2)$ [40].

Proof. If G is a chordal graph and (T, B) a clique tree for G , take for each vertex v of G exactly the subtree of T induced by the nodes whose bags contain v . G is exactly the intersection graph obtained from these subtrees since all pairs of vertices in every bag of (T, B) are connected by an edge.

For the converse, let G be an intersection graph of subtrees T_1, \dots, T_n ($n \in \mathbb{N}$) of a tree T and, for $1 \leq i \leq n$, let v_i be the vertex of G corresponding to T_i . Assume for contradiction that there is a chordless cycle $C = (v_1, v_2, \dots, v_x)$ of length $x \geq 4$ in G . Take $w_{1,2}$ and $w_{2,3}$, respectively, as a node of T_2 also contained in T_1 and T_3 . Since $\{v_1, v_3\}$ is no edge in G , $w_{1,2} \neq w_{2,3}$. Let P be a path in T_2 (and thus also in T) connecting $w_{1,2}$ and $w_{2,3}$. Now we can observe that, on the one hand, the tree $T^* = \cup_{i=1}^x T_i$ has at least a common vertex with both T_1 and T_3 . On the other hand, T^* has no common vertex with T_2 since otherwise a tree T_i with $4 \leq i \leq x$ has also a common vertex with T_2 , i.e., C is not chordless. Combining the last two observations, we obtain in T^* (and thus also in T) a $w_{1,2}$ - $w_{2,3}$ -path P' whose internal vertices are pairwise disjoint from the internal vertices in P . Consequently, we obtain a contradiction since the vertices of P and P' induce a cycle in T . \square

For measuring the similarity of intersection graphs of several kinds of objects to chordal graphs, we study in this chapter three complexity parameters (see also [57]). One of them is new, whereas the others also appear in [1, 51, 95], but were not analyzed in detail in these papers. A detailed definition of the new complexity parameters follows in the next section. Similar to treewidth and trees, for chordal graphs these parameters are all 1, and there is a sequence of graphs such that the complexity with respect to each of these parameters goes to infinity. There are many polynomial- or even linear-time algorithms on chordal graphs known, e.g., for maximum clique, for minimum clique partition [39], for maximum weighted independent set [33], and for minimum vertex coloring.

Definition 7.1.2 (maximum (weighted) clique (MC, MWC)). A *maximum clique* (MC) of a graph G is a clique of G with as many vertices as possible. For a graph $G = (V, E)$ with a weight function $w : V \rightarrow \mathbb{R}$, a *maximum weighted clique* (MWC) is a clique in G such that the sum of the weights of its vertices is as large as possible.

Definition 7.1.3 (minimum clique partition (MCP)). A *clique partition* of a graph $G = (V, E)$ is a partition of V into sets V_1, \dots, V_x such that each V_i ($1 \leq i \leq x$) induces a clique in G . A *minimum clique partition* (MCP) is a clique partition consisting of as few sets as possible.

Definition 7.1.4 (maximum weighted independent set (MWIS)). If $G = (V, E)$ is a graph with a weight function $w : V \rightarrow \mathbb{R}$, a *maximum weighted independent set* (MWIS) is an independent set S of G such that, for each independent set $S' \neq S$:

$$\sum_{v \in S'} w(v) \leq \sum_{v \in S} w(v).$$

Definition 7.1.5 (minimum vertex coloring (MVC)). A *minimum vertex coloring* (MVC) for a graph G is a coloring using as few colors as possible such that adjacent vertices are colored in G with different colors.

Contrary to the treewidth measuring the running time, each of the three parameters measure the quality of the results. In more detail, they determine the approximation ratio of several algorithms described in this chapter, e.g., for the problems above on big graph classes containing many intersection graph classes such as t -interval graphs, circular-arc graphs, (unit-)disk graphs, and intersection graphs of regular polygons or of arbitrary polygons of so-called bounded fatness. For a definition of all these objects see the end of this section.

It is not surprising that, for small graph classes such as unit-disk graphs, one can achieve better results than by the new algorithms designed for bigger classes of graphs. Nevertheless, also on small graph classes such as disk graphs new results are obtained for some of the problems mentioned above as well as for the following two problems.

Definition 7.1.6 (minimum (independent) dominating set (MDS, MIDS)). Given a graph $G = (V, E)$ a *dominating set* in G is a set $V' \subseteq V$ such that, for all $v \in V$, either $v \in V'$ or a neighbor of v is in V' . If V' is additionally an independent set, V' is an *independent dominating set*. A *minimum (independent) dominating set* (MDS, MIDS) is a set such that no other (independent) dominating set has a smaller cardinality.

Table 7.1.2 summarizes the best previously known and new approximation results for the intersection graphs of disks, regular polygons, fat objects, t -intervals, and t -fat-objects. By an *r -regular polygon* we mean a polygon with r corners placed on a cycle such that all pairs of consecutive corners of the polygon have the same distance. We assume that $r \in O(1)$. We define a set \mathcal{C} of geometric objects in \mathbb{R}^d to be a set of *fat objects* if the following holds: First of all, let us call the radius of a smallest d -dimensional ball containing all points of a geometric object S in \mathbb{R}^d the *size* of S . Moreover, let R be the size of the largest object in \mathcal{C} . Then, \mathcal{C} is called *fat* if there is a constant c such that, for each d -dimensional ball B of radius r with $0 < r \leq R$, there exist c points (possibly also outside B) such that every B -intersecting object $S \in \mathcal{C}$ of size at least r contains at least one of the c points. We also say that \mathcal{C} has *fatness* c . \mathcal{C} is called a *(c -)restricted set of fat objects* if in the condition above every B -intersecting object in \mathcal{C} (with arbitrary size) contains at least one of the c points. By a *unit* set of objects—in opposite to *arbitrary*—we mean that each object must be a copy of each other object, i.e., it has to be of the same size and shape. However, unit and arbitrary objects may be rotated and moved to any position. Thus, the right side of Fig. 7.1.1 shows a special kind of disk graph: a unit-disk graph. An intersection graph G of t -intervals is an intersection graph, where each vertex represents a *t -interval*, i.e., the union of t intervals taken from a set S of intervals. Similarly, by the intersection graph G of t -fat-objects we mean an intersection graph, where each vertex represents a *t -fat-object*, i.e., the union of t objects taken from a fat set S of objects. In both cases, G being a t -interval graph or the intersection graph of t -fat objects, S is called a *universe* of G .

	disk	r -reg. polygon	fat objects	t -interval	t -fat-objects
MIS	arbitrary: PTAS [21, 28] unit: PTAS [34] NP-h. [32]	arbitrary: PTAS [21, 28] unit: PTAS [34] NP-h. [32]	arbitrary: PTAS [21, 28] unit: NP-h. [32]	$2t$ -PA. [10] $t \geq 3$: \mathcal{APX} -h. [42, 73]	arbitrary: $O(t)$ -PA. [*] NP-h. [32]
MWIS	arbitrary: PTAS [28]	arbitrary: PTAS [28]	arbitrary: PTAS [28]	$2t$ -PA. [10]	arbitrary: $O(t)$ -PA. [*]
MDS	restricted: $O(1)$ -PA. [*] unit: PTAS [34] NP-h. [24]	restricted: $O(1)$ -PA. [*] unit: PTAS [34] NP-h. [24]	restricted: $O(1)$ -PA. [*] unit: NP-h. [24]	t^2 -PA. [19] $t \geq 2$: \mathcal{APX} -h. [42, 73]	restricted: $O(t)$ -PA. [*] $t \geq 2$: \mathcal{APX} -h. [42, 73]
MIDS	restricted: $O(1)$ -PA. [*] unit: PTAS [49] NP-h. [24]	restricted: $O(1)$ -PA. [*] unit: PTAS [49] NP-h. [24]	restricted: $O(1)$ -PA. [*] unit: NP-h. [24]		restricted: $O(t)$ -PA. [*] $t \geq 2$: NP-h. [24]
MVC	arbitrary: 5-PA. [41, 64, 65] unit: 3-PA. [65] $\frac{4}{3}$ -PA. is NP -h. [24, 37, 50]	arbitrary: $O(1)$ -PA. [57, 95] unit: $O(1)$ -PA. [65] NP-h. [37, 50]	arbitrary: $O(1)$ -PA. [57, 95] unit: NP-h. [37, 50]	$2t$ -PA. [10]	restricted: $O(t)$ -PA. [*]
MC	arbitrary: $O(1)$ -PA. [*] unit: $\in P$ [24]	arbitrary: $O(1)$ -PA. [*]	arbitrary: $O(1)$ -PA. [*]	$\frac{t^2-t+1}{2}$ -PA. [19] $O(t)$ -PA. [*] $t \geq 3$: NP-h. [19]	arbitrary: $O(t)$ -PA. [*] $t \geq 3$: NP-h. [19]
MWC	arbitrary: $O(1)$ -PA. [*]	arbitrary: $O(1)$ -PA. [*]	arbitrary: $O(1)$ -PA. [*]	$\frac{t^2-t+1}{2}$ -PA. [19] $O(t)$ -PA. [*]	arbitrary: $O(t)$ -PA. [*]
MCP	arbitrary: $O(1)$ PA. [*] unit: 3-PA. [20]	arbitrary: $O(1)$ -PA. [*]	arbitrary: $O(1)$ -PA. [*]	$O(\log^2 n /$ $\log(1 + 1/t))$ - PA. [*]	arbitrary: $O(t)$ -PA. [*]

Table 7.1.2: Approximation results. PA. and NP-h. is used as abbreviation for polynomial-time approximation algorithm and NP-hard, respectively. Moreover, n denotes the number of vertices of the intersection graph. [*] marks a result shown in this chapter.

As usual, disks and regular polygons should be defined in the plane \mathbb{R}^2 , intervals in \mathbb{R} and fat objects in \mathbb{R}^d , where we assume that $d = O(1)$. Concerning the results in table 7.1.2 including the hardness results, we assume that—beside an intersection graph itself—a representation of the intersection graph is given. More precisely, for the intersection graph of a set S of (1) disks, (2) r -regular polygons, (3) t -intervals, (4) fat objects, or (5) t -fat objects, we are given for each element in S its radius and the coordinates of its center in case 1, the coordinates of the center and at least one corner in case 2, the start and end point of each interval in case 3, and a representation in case 4 that, for each pair X, Y of objects, each point $p \in \mathbb{R}^d$, and each d -dimensional ball B represented by the coordinates of its center and its radius $r \leq R$, supports the following computations in polynomial time: Decide whether X and Y intersect, whether X and B intersect, and whether p is contained in X . Moreover, determine the size s of X as well as the center of the ball with a radius s containing all points of X , and find c points such that one is contained in every object of size $\geq r$ intersecting B . In addition, in case 5 each t -fat-object has a representation of its objects as described in case 4. For many applications, representations as described above are given explicitly.

Very related to the graph classes considered in this chapter is the so-called class of *sequentially k -independent graphs* introduced by Akcoglu, Aspnes, DasGupta, and Kao [1] and studied more extensively in a recent paper by Ye and Borodin [95]. Even though the results of Ye et al. and the results here are achieved completely independently, there are similarities between the paper and this chapter. Consequently, the results here of generalizing chordal graphs are quite natural, but surprisingly have not been studied more extensively before. Since some of the graph classes considered here are smaller than the classes studied by Ye et al.—but still including big classes of intersection graphs—we obtain good approximation algorithms for MDS and MIDS for the graph classes considered here, where especially approximation results for the latter problem were mentioned as an interesting open problem for k -sequentially independent graphs. We also generalize some results of Ye et al. for the MCP and the MWC. In contrast to Ye et al. we additionally study t -interval graphs and intersection graphs of t -fat objects.

Other generalized classes of graphs including the intersection graphs of unit disks or r -regular polygons of unit size are graph classes of so-called *polynomially bounded growth* studied by Nieberg, Hurink and Kern [49, 70]. Nieberg et al. presented a PTAS for MWIS, MDS and MIDS for these classes of graphs, but graphs of polynomially bounded growth do not include the intersection graphs of arbitrary disks, arbitrary r -regular polygons, t -interval graphs, etc.

Our results include the first polynomial-time approximation algorithms with constant approximation ratio for maximum clique and minimum clique partition on disk graphs and on intersection graphs of r -regular polygons. We also present a polynomial-time approximation algorithm with constant approximation ratio for minimum dominating set on the intersection graphs of a restricted set of r -regular polygons. Recently, Erlebach and van Leeuwen [29] presented an approximation algorithm with constant approximation ratio for the same problem on an arbitrary set of r -regular polygons, however, they do not allow to rotate the polygons in contrast to this thesis. Our results also imply an approximation algorithm with constant approximation ratio for minimum dominating set on intersection graphs of an arbitrary set of iso-oriented r -regular polygons. A set

of r -regular polygons is called *iso-oriented* if each polygon is obtained from each other by translation and scaling. We also improve the approximation ratio of maximum clique on t -interval graphs from $(t^2 - t + 1)/2$ [19] to $O(t)$. In general, the new results also extend to intersection graphs of a restricted set of t -fat objects and further classes of graphs not discussed in this thesis.

7.2 New Complexity Parameters

In this section, we introduce three complexity parameters. For each complexity parameter, we present examples of classes of intersection graphs for which the complexity parameter is bounded by a constant.

Definition 7.2.1 (*k*-perfectly groupable). A graph is *k*-perfectly groupable if the neighbors of each vertex v can be partitioned into k sets S_1, \dots, S_k such that $G[S_i \cup \{v\}]$ is a clique for each $i \in \{1, \dots, k\}$.

For each object S of a k -restricted set \mathcal{C} of fat objects and a smallest ball B containing S , there exists a set P_S of k points such that every object in \mathcal{C} intersecting B covers a point in P_S . For each S -intersecting and hence also B -intersecting object $S' \in \mathcal{C}$, choose one of the points in $S' \cap P_S$ as a representative. Then all S -intersecting objects having the same representative in P_S induce a clique in the intersection graph. Hence, the intersection graph of a k -restricted set of fat objects is *k*-perfectly groupable. Unit-disk graphs and unit-square graphs are *k*-perfectly groupable for a suitable constant k . Graphs of maximum degree k are also *k*-perfectly groupable.

Definition 7.2.2 (*k*-simplicial, *k*-simplicial elimination order, successor). A graph G is *k*-simplicial if there is an order v_1, \dots, v_n of the vertices of G such that, for each vertex v_i ($1 \leq i \leq n$), the subset of neighbors of v_i contained in $\{v_j \mid j > i\}$ can be partitioned into k sets S_1, \dots, S_k such that $G[S_j \cup \{v_i\}]$ is a clique for each $j \in \{1, \dots, k\}$. The vertices in $\{v_j \mid j > i, \{v_i, v_j\} \in E(G)\}$ are called the *successors* of v_i and the order above of the vertices in G is called a *k*-simplicial elimination order.

Let \mathcal{C} be a set of fat objects S_1, \dots, S_n ordered by non-decreasing size. Let k be the fatness of \mathcal{C} . Then, for each object S_i , we can find k points such that every S_i -intersecting object in $\{S_{i+1}, \dots, S_n\}$ contains one of the k points. If, for $i \in \{1, \dots, n\}$, we define v_i to be the vertex representing S_i in the intersection graph G of \mathcal{C} , then v_1, \dots, v_n defines a *k*-simplicial elimination order. Therefore, G is *k*-simplicial. Also note that disk graphs and square graphs are *k*-simplicial for a suitable constant k .

Definition 7.2.3 (*k*-perfectly orientable). A graph G is called *k*-perfectly orientable if each edge $\{u_1, u_2\}$ of G can be assigned to exactly one of its endpoints u_1 and u_2 such that, for each vertex v , the vertices connected to v by edges assigned to v can be partitioned into k sets S_1, \dots, S_k such that $G[S_i \cup \{v\}]$ is a clique for each $i \in \{1, \dots, k\}$. We write $a(\{u_1, u_2\}) = u_1$ if $\{u_1, u_2\}$ is assigned to u_1 .

We now show that the intersection graph $G = (V, E)$ of a set of t -fat-objects \mathcal{C} with a universe of fatness c is $(t \cdot c)$ -perfectly orientable. Let $V = \{v_1, \dots, v_n\}$ and, for each $i \in \{1, \dots, n\}$, let \mathcal{S}_i be the union of t objects $S_{i,1}, \dots, S_{i,t}$ represented by v_i . In particular, $(S_{1,1}, \dots, S_{1,t}, \dots, S_{n,1}, \dots, S_{n,t})$ is the universe of \mathcal{C} . Choose, for each edge $\{v_i, v_j\}$ with $i < j$, a pair $\{p, q\}$ of indices such that $S_{i,p}$ and $S_{j,q}$ intersect. Assign $\{v_i, v_j\}$ to v_i if the size of $S_{i,p}$ is smaller than the size of $S_{j,q}$ and to v_j otherwise. Then, for each vertex v_i , one can find $t \cdot c$ points—namely the c points for each of the objects $S_{i,1}, \dots, S_{i,t}$ —such that each \mathcal{S}_i -intersecting t -fat-object \mathcal{S}_j with $\{v_i, v_j\}$ being assigned to v_i must intersect \mathcal{S}_i in at least one of the $t \cdot c$ points. Here, once again, we exploit the fact that a larger object intersecting a smaller object must intersect the smaller object in at least one of the c points for the smaller object. Therefore, the set of vertices being endpoints of edges assigned to v_i can be partitioned into $\leq t \cdot c$ cliques. This proves that G is $(t \cdot c)$ -perfectly orientable. Note also that the intersection graphs of t -intervals are $2t$ -perfectly orientable. For these graphs, an edge $\{v_i, v_j\}$ with $i < j$ is assigned to v_i if the t -interval represented by v_j intersects one of $2t$ endpoints of the intervals whose union is represented by v_i . Otherwise, $\{v_i, v_j\}$ is assigned to v_j .

Moreover, intersection graphs of subforests of a tree are k -simplicial if each subforest consists of at most k trees. In this case an edge $\{v_i, v_j\}$ with $i < j$ is assigned to v_i if the subforest represented by v_j intersects one of k roots of the subforest represented by v_i . Otherwise, $\{v_i, v_j\}$ is assigned to v_j .

We next present explicit upper bounds for the three complexity parameters on some special intersection graphs. Before that let us define the *inball* and the *outball* of a square S to be the ball with largest radius containing only points of S and the ball with smallest radius containing all points of S , respectively. The *center* of S is the center of its outball.

Lemma 7.2.4. *An intersection graph of t -squares, i.e., of unions of t (not necessarily axis-parallel) squares, is*

1. *10-perfectly groupable if $t = 1$ and if the squares are of unit size,*
2. *10-simplicial if $t = 1$, and*
3. *10 t -perfectly orientable.*

Proof. For proving the first two cases, let G be the intersection graph of a set S of squares. It remains to show that, for a square Q of minimal side length ℓ , there are 10 points—called the *barriers* of Q —such that every Q -intersecting square Q' of length $\geq \ell$ must cover at least one of them. This fact also proves case 3 since the universe of a set of t -squares then must have fatness 10.

We first describe our choice of the 10 barriers of Q . See also the left side of Fig. 7.2.1 for the following construction. Let b_1 and b_2 be the two perpendicular bisectors of the sides of Q . Choose two barriers x and y of Q as points on b_1 such that the part of b_1 inside Q is divided into three parts of equal length. We call these two points the *inner barriers* of Q . Let C be the curve surrounding Q that consists of all points having a distance of exactly ℓ to one of the inner barriers and a distance of at least ℓ to the remaining inner barrier. The remaining 8 barriers, called *outer barriers*, are almost equidistant points on C . More exactly, 4 outer barriers of Q are placed on the $2 + 2$ intersection points of C with b_1 and b_2 . Choosing the other 4 outer barriers of Q is more sophisticated. Let x' and y' be the two points on b_1 having the same distance to the center of Q as to x and y , respectively. In addition, let r_1 and r_2 (r_3 and r_4) be the 2 rays

starting from x' (from y') such that each of the 4 rays intersects a corner of Q but neither b_1 nor b_2 . The four remaining outer barriers are placed on the intersection points of C with the rays r_1, \dots, r_4 .

By a simple mathematical analysis one can show that the distance between any two consecutive outer barriers on C is strictly smaller than ℓ . It remains to show that each square of side length at least ℓ intersecting Q also covers one of the barriers of Q . Assume for a contradiction that we can find a square Q' of side length at least ℓ intersecting Q but none of the barriers of Q . W.l.o.g. we can assume that Q' has side length exactly ℓ since otherwise Q' also contains a smaller square intersecting Q . Let \mathcal{H} be the convex hull of the outer barriers and let B be the largest circle contained in \mathcal{H} such that B has the same center as Q . B and thus also \mathcal{H} contain at least one corner of Q' since Q' intersects Q and B , and since a simple mathematical analysis shows that each chord of B with length at most ℓ does not intersect Q . We now distinguish two cases.

Case 1: No side of Q' is completely contained in the convex hull \mathcal{H} of the outer barriers. For each pair of consecutive outer barriers p and q on C , let us define $C_{p,q}$ to be the semi-circle inside \mathcal{H} with endpoints p, q and hence having a diameter equal to the distance between p and q . See again the left side of Fig. 7.2.1. Let z be the corner of Q' inside B with the smallest distance to a point in Q . Note that the two sides of Q' ending in z are not completely contained in \mathcal{H} . Consequently, by Thales' theorem and the fact that Q' does not contain any barriers there must be two consecutive outer barriers p and q on C such that z is contained in the face enclosed by $C_{p,q}$ and \overline{pq} . Again a simple mathematical analysis shows that none of our semi-circles intersects Q . Thus, neither z nor any other point of Q' is covered by Q . Contradiction.

Case 2: At least one side of Q' is completely contained in \mathcal{H} . Since each pair of consecutive outer barriers on C has a distance smaller than ℓ , the center q of Q' is inside \mathcal{H} .

By symmetry, w.l.o.g. we can assume that the distance between q and y is smaller or equal than the distance between q and x . Let \mathcal{H}' be the convex hull of x and the outer barriers having a distance of at most ℓ to y . On the one hand, for each pair of consecutive barriers q_1 and q_2 on \mathcal{H}' , there is at most one corner in the face bounded by $\overline{q_1q_2}$ and the semi-circle outside \mathcal{H}' with endpoints q_1 and q_2 . On the other hand, at least one corner of Q' is outside \mathcal{H}' since the inball of Q' , which does not contain y , must intersect the border of \mathcal{H}' . Consequently,

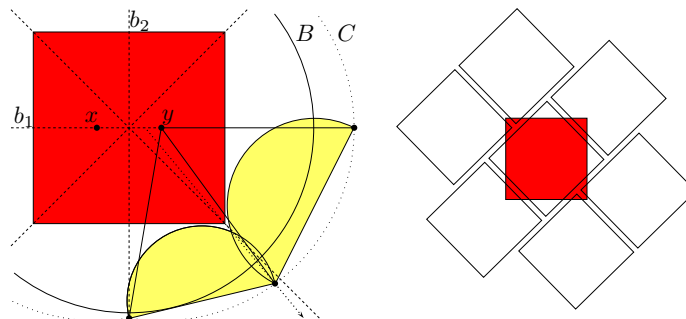


Figure 7.2.1: The left side shows a square with some barriers, and on the right side, we see a square intersecting 7 disjoint squares.

there are two sides s_1 and s_2 of Q' that have a common corner p outside \mathcal{H}' and that intersect \mathcal{H}' between to outer barriers, say q_1 and q_2 .

Let T be the triangle with corners y , q_1 and q_2 . Since Q' is a square of side length ℓ , since p is not covered by T , and since T is a triangle with two sides of length ℓ and with an s_1 -intersecting side of length at most ℓ , y has to be inside Q' . Contradiction. \square

Observation 7.2.5. *Some square graphs are not 6-perfectly groupable as shown on the right side of Fig. 7.2.1.*

Lemma 7.2.6. *The intersection graph of a set of rectangles, all having aspect ratio of α , is $10\lceil\alpha\rceil$ -simplicial.*

Proof. Replace each rectangle of side length $(\ell \times \alpha\ell)$ by a set of $\lceil\alpha\rceil$ (possibly overlapping) squares of side length $(\ell \times \ell)$ such that each point contained in one squares is contained in the rectangle and vice versa. For each rectangle r_1 replaced by squares of a size s_1 , one can find $10\lceil\alpha\rceil$ points P such that every r_1 -intersecting square of size $s_2 \geq s_1$ replacing another rectangle r_2 must cover one of the points in P . Here we use the fact that each rectangle can be replaced by $\lceil\alpha\rceil$ squares of the same size. \square

Lemma 7.2.7. *Let c be a fixed constant and G be an intersection graph, where each vertex represents a union of t polygons taken from a universe of iso-oriented c -regular polygons. Then G is $(t \cdot c)$ -perfectly orientable.*

Proof. The intersection of two iso-oriented c -regular polygons of the same size must contain at least one of the corners of the two polygons. Note that this does not hold for general rotated polygons. Let $\{v_1, \dots, v_n\}$ be the vertices of G . We assign an edge $\{v_i, v_j\}$ in G with $i < j$ to v_i if and only if one of the polygons in the union of polygons represented by v_i has a corner contained in the union of polygons represented by v_j . Otherwise, we assign it to v_j . The edges assigned to a vertex v can be partitioned into $t \cdot c$ sets such that the endpoints of the edges of each set induce a clique in G since we have one clique for each corner of the t polygons. \square

7.3 The 3 Complexity Parameters in Relation

In the following we study the relations between the complexity parameters defined in the last section to each other and to treewidth.

Observation 7.3.1. *Each k -perfectly groupable graph is k -simplicial because any ordering of the vertices defines a k -simplicial elimination order. Conversely, for all $n, k \in \mathbb{N}$ with $k < n - 1$, an n -vertex star, i.e., an n -vertex tree with $n - 1$ leaves, is not k -perfectly groupable, but it is 1-simplicial.*

Lemma 7.3.2. *A k -simplicial graph is also k -perfectly orientable, but for all $n, \ell \in \mathbb{N}$ with $n \geq 12$ and $\ell < \lfloor \sqrt{n/3} \rfloor$, there exists a 2-perfectly orientable graph with n vertices that is not ℓ -simplicial.*

Proof. Let G be a k -simplicial graph having a k -simplicial elimination order v_1, \dots, v_n . If, for a vertex v , all edges that are incident to v as well as to a

successor of v are assigned to v , the endpoints $u \neq v$ of the edges assigned to v can be partitioned into k sets S_1, \dots, S_k such that $G[S_i \cup \{v\}]$ is a clique for every $i \in \{1, \dots, k\}$. In other words, G is k -perfectly orientable.

Let us choose an arbitrary $n \in \mathbb{N}$ with $n \geq 12$ and let $k = \lfloor \sqrt{n/3} \rfloor$. We now construct a 2-perfectly orientable graph $G = (V, E)$ with n vertices that is not ℓ -simplicial for any $\ell < k$. For a sketch of the following construction, see also Fig. 7.3.1. The vertices of this graph are divided into three disjoint sets S_0, S_1 and S_2 of size k^2 and, if $n - 3k^2 > 0$, a further set $R = V \setminus (S_0 \cup S_1 \cup S_2)$ of isolated vertices. Each set S_i ($i \in \{0, 1, 2\}$) is divided into k subsets $S_{i,1}, \dots, S_{i,k}$ of size k . For each $i \in \{0, 1, 2\}$ and each $j \in \{1, \dots, k\}$, we introduce edges between each pair of vertices contained in the same subset $S_{i,j}$ and assign each of these edges arbitrarily to one of its endpoints. Let us define a numbering on the vertices of $S_{i,j}$ such that we can refer to the h -th vertex of $S_{i,j}$. For each $i \in \{0, 1, 2\}$ and each $h, j \in \{1, \dots, k\}$, we additionally introduce edges between the h -th vertex u of $S_{i,j}$ and all vertices of $S_{(i+1) \bmod 3, h}$. We assign them to u . The constructed graph G is 2-perfectly orientable since the endpoints of an edge assigned to a vertex u being the h -th vertex of a subset $S_{i,j}$ belong to one of the two cliques induced by the vertices of $S_{i,j}$ and $S_{(i+1) \bmod 3, h}$. However, u is also adjacent to k vertices in $S_{(i-1) \bmod 3}$, namely to the h -th vertex in each of the subsets $S_{(i-1) \bmod 3, 1}, \dots, S_{(i-1) \bmod 3, k}$ —see a blue vertex in Fig. 7.3.1. Since there is no edge between a vertex in $S_{(i-1) \bmod 3, j_1}$ and a vertex in $S_{(i-1) \bmod 3, j_2}$ for $j_1 \neq j_2$ and since u has neighbors in each of the sets $S_{(i-1) \bmod 3, 1}, \dots, S_{(i-1) \bmod 3, k}$, the neighbors of u contain an independent set of size at least k . This means that the neighbors cannot be covered by fewer than k cliques and that thus G cannot be ℓ -simplicial for any $\ell < k$. \square

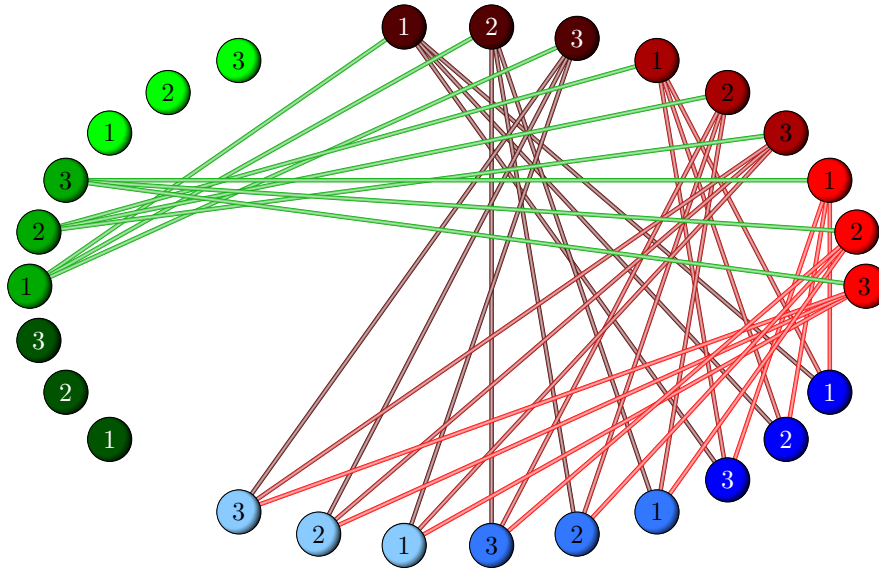


Figure 7.3.1: A sketch of the graph constructed in the proof of Lemma 7.3.2 for the case $n = 27$. Due to a better overview, several edges are omitted. S_0, S_1 , and S_2 are the vertices in red, blue, and green, respectively. Moreover, for $i, j \in \{0, 1, 2\}$, the subsets of each S_i are shown in slight color-variations, and the numbers in the vertices denote the numbering on the vertices of each $S_{i,j}$.

Lemma 7.3.3. *All graphs with treewidth k are k -simplicial and therefore also k -perfectly orientable.*

Proof. Let $G = (V, E)$ be a graph and (T, B) be a k -normal tree decomposition for G . Note that each leaf of T has a bag containing a vertex v that is not contained in any other bag of (T, B) . Consequently, v has at most k neighbors. We can choose v as the first vertex of a k -simplicial elimination order and recursively determine a k -simplicial elimination order on $G[V \setminus \{v\}]$. \square

Observation 7.3.4. *The n -vertex clique is an example for a 1-perfectly groupable graph G that does not have treewidth $n - 2$. Conversely, the n -vertex star is a graph with treewidth 1 that is not $(n - 2)$ -perfectly groupable.*

7.4 Recognition Problems

In this section we show that for a given graph—similar to treewidth—it is NP-hard to decide whether one of the three complexity parameters is bounded by a constant. All results are obtained by a reduction from the NP-hardness of minimum clique partition [58].

Lemma 7.4.1. *Given a constant k and a graph G , it is NP-hard to decide whether G is k -perfectly groupable.*

Proof. Given a graph G as an instance of the minimum clique partition problem, we replace $G = (V, E)$ by $G' = (V \cup \{v^+\}, E \cup \{\{v^+, v\} \mid v \in V\})$ for a new vertex $v^+ \notin V$. If G' is k -perfectly groupable, the neighbors of v^+ can be partitioned into at most k cliques. This means that G has a clique partition of size k . If G has a clique partition of size k , the neighbors of v^+ in G' can be partitioned into at most k cliques. The same is true for each other vertex v of G' if we cover the neighbors of v by the k cliques covering G and add v^+ to one of these cliques. Hence, G is k -perfectly groupable. \square

Lemma 7.4.2. *Given a constant k and a graph G , it is NP-hard to decide whether G is k -simplicial.*

Proof. Given a graph G as an instance of the minimum clique partition problem, we construct a graph G' on which we want to find a k -simplicial elimination order. G' is obtained from G by adding $k + 1$ new vertices to G and connecting each new vertex to each vertex of G . If G has a clique partition of size k , construct an ordering of the vertices of G' beginning with the $k + 1$ new vertices. All successors are vertices in G and hence can be covered by at most k cliques. Therefore G' is k -simplicial. In the reverse direction a k -simplicial elimination order v_1, \dots, v_n cannot start with a vertex of G since such a vertex is adjacent to all new vertices. Thus, the successors of v_1 contain all vertices of G and G has a clique partition of size k . \square

Lemma 7.4.3. *Given a constant k and a graph G , it is NP-hard to decide whether G is k -perfectly orientable.*

Proof. Given an n -vertex graph $G = (V, E)$ as an instance of the minimum clique partition problem, we add a set V' of $nk + 1$ new vertices to G and connect each new vertex to each vertex in V . Let G' be the graph obtained. For showing

that G having a clique partition of size at most k implies G' being k -perfectly orientable, assign all incident edges of a vertex $v' \in V'$ to v' and edges $e \in E$ to an arbitrary endpoint of e . Then a vertex v together with the endpoints of edges assigned to $v \in V \cup V'$ induce k cliques, i.e., G' is k -perfectly orientable.

Conversely, let us assume that G' is k -perfectly orientable and let $a : E \rightarrow V \cup V'$ be a suitable assignment of the edges to their endpoints. For each vertex $v \in V$ at most k of the $nk + 1$ new edges incident to v can be assigned by a to v since there are no edges between two vertices of V' . Thus, there is at least one $v' \in V'$ with all its edges assigned to itself. Thus, G must have a clique partition of size at most k . \square

7.5 Algorithms on Intersection Graphs

We present in the following polynomial time approximation algorithms for several NP-hard problems on graph classes with one of the three complexity parameters bounded by a constant. We always implicitly assume that we are given an explicit *representation* of a graph as a k -perfectly groupable, k -simplicial, or k -perfectly orientable graph G . By that we mean that we are given, for each vertex v , a partition of its neighbors, of its successors, and of the vertices connected to v by edges assigned to v , respectively, into k sets S_1, \dots, S_k such that $G[S_i \cup \{v\}]$ is a clique for all $i \in \{1, \dots, k\}$. In addition, we are given a k -simplicial elimination order in the case of a k -simplicial graph and, for each vertex of G , the edges assigned to it in the case of a k -perfectly orientable graph. These representations are sufficient even for intersection graphs. We do not need the explicit representations as intersection graphs described in Section 7.1. However, we can use a representation as intersection graph to construct the new representations above in polynomial time.

Lemma 7.5.1. *Minimum (independent) dominating set can be k -approximated on k -perfectly groupable graphs in polynomial time.*

Proof. As a k -approximative solution on a k -perfectly groupable graph G we output a maximal—not necessarily maximum—independent set S of G . To prove correctness, let us consider a minimum (independent) dominating set S_{opt} of G . For all $v \in S \setminus S_{\text{opt}}$, there must be a neighbor of v in $S_{\text{opt}} \setminus S$. However, each such neighbor cannot cover more than k vertices of S , since G is k -perfectly groupable. Consequently, S is an independent dominating set of size at most $k|S_{\text{opt}}|$. \square

Lemma 7.5.2. *Maximum weighted independent set, maximum weighted clique, and minimum clique partition are k -approximable on k -simplicial and thus also on k -perfectly groupable graphs in polynomial time.*

Proof (maximum weighted independent set). The original proof [1] as well as the proof below use the local ratio technique, but both proofs describe this technique in different ways. On a k -simplicial graph $G = (V, E)$ with weight function $w : V \rightarrow \mathbb{R}$ we obtain a k -approximative solution as follows: First remove all vertices with non-positive weights since they can be excluded from a solution. If the remaining graph G' is the empty graph, return the empty set. Otherwise, choose the vertex v that among the remaining vertices appears first in the k -simplicial elimination order of G . Decrease the weight of v and

its neighbors by the weight $w(v)$ of v . Compute an independent set I for the graph G' with the new weight function w' recursively. If a neighbor of v is in I , return I , and otherwise return $I \cup \{v\}$. In both cases, if we reincrease the weights of v and its neighbors, the weight of our solution increases by at least $w(v)$. Note that the weight of any other independent set can increase by at most $k \cdot w(v)$ since all neighbors of v in G' are successors of v in G and thus they can be partitioned into $\leq k$ cliques. Consequently, the difference between the weights of optimal solutions for G with weight function w and for G' with weight function w' is bounded by $k \cdot w(v)$. Since this is true for all recursive steps, the solution obtained has approximation ratio k . The algorithm terminates since each recursion sets the weight of at least one vertex to 0. Concerning the running time note that a representation of G' as a k -simplicial graph can be computed in polynomial time from such a representation of G . \square

Proof (maximum weighted clique). If we are given a k -simplicial graph, for each vertex v , choose a clique C_v of maximal weight among the $\leq k$ cliques induced by v and the successors of v . Return the clique with maximal weight among the cliques in $\{C_v \mid v \in V\}$. This solution has approximation ratio k since a maximum weighted clique C_{opt} must also contain a vertex v with C_{opt} consisting only of v and a subset of its successors. \square

Proof (minimum clique partition). Given a graph G and additionally a k -simplicial elimination order v_1, \dots, v_n for G , we first compute the graph G' obtained by removing v_1 and its neighbors from G . We then solve the problem recursively on G' . Let S' be the collection of vertex sets obtained as a solution for G' . Note that the graph induced by the removed vertices can be partitioned into a set Z of at most k cliques. We output $S = S' \cup Z$ as a solution for G . Note that v_1 is not incident to any vertex of G' . This guarantees that the difference between the size of a clique partition for G and for G' is at least 1. Therefore, we obtain a clique partition that uses at most k times as many cliques as an optimal clique partition for G . \square

For the proof of the next lemma, we need another definition.

Definition 7.5.3 (linear program, integer linear program, relaxation). A *linear program* (P) specifies a linear *objective function* in $n \in \mathbb{N}$ variables x_1, \dots, x_n and $m \in \mathbb{N}$ linear constraints of the form $a_1x_1 + \dots + a_nx_n \bowtie b$ with $a_1, \dots, a_n, b \in \mathbb{R}$ and \bowtie either $\leq, =$ or \geq . A *solution* of the linear program is a tuple $(x_1, \dots, x_n) \in \mathbb{R}^n$ that satisfies all the constraints and the goal is to find a solution that maximizes or minimizes the objective function. An *integer linear program* is a linear program where the solutions are restricted to $(x_1, \dots, x_n) \in \mathbb{Z}^n$ and a *relaxation* of an integer linear program removes this restriction.

Note that the relaxation (P') of an integer linear program (P) has all solutions of (P). Thus, the optimal solution of (P') is at least as good as the optimal solution of (P).

Lemma 7.5.4. *On k -perfectly orientable n -vertex graphs, there are polynomial-time algorithms with approximation ratio*

1. $2k$ for maximum weighted independent set, minimum vertex coloring and maximum weighted clique.
2. $O(k \log^2 |V|)$ for minimum clique partition.

For the following proofs let $G = (V, E)$ be a k -perfectly orientable n -vertex graph, and for each $u \in V$, let $V_{u,1}, \dots, V_{u,k}$ be k pairwise disjoint vertex sets such that their union is the set of the neighbors of u assigned to u and such that $C_{u,i} = G[V_{u,i} \cup \{u\}]$ is a clique for all $1 \leq i \leq k$. Moreover, define $\mathcal{C} = \{C_{u,i} \mid u \in V, 1 \leq i \leq k\}$.

Proof (maximum weighted independent set). The proof is based on the ideas in [10]. W.l.o.g. G has no vertex with weight ≤ 0 . Otherwise remove such vertices and their adjacent edges; if no vertex remains, return the empty set as solution.

Let us define $V = \{1, \dots, n\}$ and $w(i)$ ($1 \leq i \leq n$) to be the weight of vertex i . Let w be the vector $(w(1), \dots, w(n))$ and $x = (x_1, \dots, x_n)^T$ be an optimal solution of the following integer linear program (P). Then one can easily observe that $S = \{i \mid x_i = 1\}$ is an optimal solution for the maximum weighted independent set problem on G .

$$(P) : \max \quad wx$$

$$\text{s.t.} \quad \sum_{v \in V'} x_v \leq 1 \quad \forall (V', E') \in \mathcal{C}$$

$$x_v \in \{0, 1\} \quad \forall v \in V$$

We next consider the relaxation (P') of (P) obtained by replacing $x_v \in \{0, 1\}$ by $0 \leq x_v \leq 1$ for all $v \in V$. Let x be an optimal solution of (P') and, for each vertex $v \in V$, define $N[v]$ to be the set consisting of v and its neighbors. We next show that there is a vertex $v \in V$ with $\sum_{u \in N[v]} x_u \leq 2k$. Recall that $a(\{v, w\}) = v$ means that $\{v, w\}$ is assigned to v . By the first constraint of (P') we have

$$\forall v \in V : k x_v + \sum_{\substack{\{v,u\} \in E, \\ a(\{v,u\})=v}} x_u \leq k,$$

$$nk \geq \sum_{v \in V} \sum_{\substack{\{v,u\} \in E, \\ a(\{v,u\})=v}} x_u = \sum_{v \in V} \sum_{\substack{\{v,u\} \in E, \\ a(\{v,u\})=u}} x_u, \text{ and therefore}$$

$$\exists v : \sum_{\substack{\{v,u\} \in E, \\ a(\{v,u\})=u}} x_u \leq k,$$

which by the first inequality implies

$$\exists v : \sum_{u \in N[v]} x_u \leq 2k.$$

A $2k$ -approximative solution for the MWIS on G is computed as follows. Find a solution x for the linear program (P'). This can be done in time polynomial in $|V|$ and $|E|$ as shown in [86]. Afterwards it is easy to find a vertex \tilde{v} with $\sum_{u \in N[\tilde{v}]} x_u \leq 2k$. Then, decrease the weight of \tilde{v} and its neighbors by $w(\tilde{v})$ and find recursively an independent set S' of approximation ratio $2k$ for G with the decreased weight function, which we denote in the following by w' . More

precisely, we search for such a solution on the graph obtained from G after removing the vertices of weight 0 or lower with respect to w' since these vertices cannot help to increase the size of a maximal independent set on G . Take as a solution $S = S' \cup \{\tilde{v}\}$ if $S' \cap N[\tilde{v}] = \emptyset$, and $S = S'$ otherwise. For analyzing the approximation ratio, let OPT and OPT' be optimal solutions of (P') with respect to w and w' , respectively. Note that $w(OPT) \leq w'(OPT) + 2kw(\tilde{v}) \leq w'(OPT') + 2kw(\tilde{v})$. Since the replacement of w by w' decreases the weight of \tilde{v} and its neighbors by $w(\tilde{v})$, and since at least one of these vertices is contained in S , we have $w(S) \geq w'(S) + w(\tilde{v}) \geq w'(S') + w(\tilde{v})$. We therefore can conclude that the vector x with $x_v = 1$ for all $v \in S$ and $x_v = 0$ for all $v \in V \setminus S$ is a $2k$ -approximative solution of (P') and hence also for (P) . Each weight reduction step lowers the weight for at least one vertex from a positive value to 0 after which the vertex is removed as described above. We thus can conclude that the algorithm terminates after at most n recursive calls, i.e., we have indeed a polynomial running time. \square

Proof (minimum vertex coloring). Construct an order v_1, \dots, v_n of the vertices of G such that, for each vertex v_i ($i \in \{1, \dots, n\}$), at least half of the edges in $G[\{v_i, \dots, v_n\}]$ being adjacent to v_i are assigned to v_i . To see that such a vertex exists, assume that after determining v_1, \dots, v_{i-1} for some $i \in \mathbb{N}$, we direct each edge $\{u, v\}$ in G from the vertex to which it is assigned to the other endpoint. Then, in the subgraph G' of G induced by the vertices not in $\{v_1, \dots, v_{i-1}\}$, the number of incoming edges is the same than the number of outgoing edges. Hence there must be a vertex with at least as many outgoing than incoming edges. We then let v_i be this vertex. We now want to color the vertices v_n, \dots, v_1 in this order with numbers in $\{1, \dots, n\}$. We color each vertex $v \in V$ with the smallest number different from the colors of all already colored neighbors of v . Concerning the approximation ratio, for each vertex v , let us define D_v to be the largest set $V_{v,i}$ among the partition $V_{v,1}, \dots, V_{v,k}$ of the neighbors of v into k cliques. Then, each vertex v of G obtains a color smaller or equal $2k|D_v| + 1$, whereas an optimal coloring must color v and its neighbors with at least $|D_v| + 1$ colors. Thus, the coloring obtained is a $2k$ -approximation. \square

Proof (maximum weighted clique). As a $2k$ -approximative solution, return the clique $C \in \mathcal{C}$ of maximal weight. Let us compare the weight of C with the weight of a maximal clique C_{OPT} of G . The subgraph of G induced by the vertices of C_{OPT} contains at least one vertex u for which the sum of the weights of the neighbors not being endpoints of edges assigned to u does not exceed the sum of the weights of the neighbors being endpoints of edges assigned to u . Thus, there is one clique in $\{C_{u,1}, \dots, C_{u,k}\}$ for which the sum of the weights of its vertices is at least $(\sum_{v \in N[u]} w(v))/2k$. Hence there is indeed a clique in \mathcal{C} whose weight is at most a factor $2k$ smaller than the weight of C_{OPT} . \square

Proof (minimum clique partition). As part of the following computation, we want to find a minimal number of cliques in \mathcal{C} in polynomial time such that the union of their vertex sets is V . Unfortunately, this is an instance of the NP-hard set cover problem. However, using Johnson's algorithm [52] we can find a subset of the cliques in \mathcal{C} that covers V and that is at most a factor $O(\log |V|)$ larger than the minimal number of cliques in \mathcal{C} . We return this subset as an approximative solution. We achieve the approximation ratio $O(k \log^2 |V|)$ since

there is a clique partition of V using only cliques in \mathcal{C} that uses $O(k \log |V|)$ as many cliques as a minimum clique partition C_{OPT} : Let C_{OPT} consist of $q \leq n$ arbitrary cliques C_1, \dots, C_q of G . Choose a vertex v of C_1 such that in the subgraph of G induced by the vertices of C_1 at least half of the edges adjacent to v are assigned to v . Remove the clique among $C_{v,1}, \dots, C_{v,k}$ containing the largest number of not already deleted vertices in C_1 . This decreases the number of vertices of C_1 by a factor of at least $1 - \frac{1}{2^k} = \frac{2^k - 1}{2^k}$. Repeat this step recursively until, after $O(\log |V| / \log \frac{2^k - 1}{2^k}) = O(\log |V| / \log(1 + \frac{1}{k})) = O(k \log |V|)$ steps, C_1 contains no vertices any more. More precisely, when choosing a vertex v for which at least half of the adjacent edges are assigned to v , only count the edges not already being deleted. If we do the same for the remaining cliques, we obtain a clique partition with $O(q \log |V| / \log \frac{2^k - 1}{2^k})$ cliques part of \mathcal{C} . \square

Index

A page number in italics refers to a definition. Other occurrences are possibly missing. Moreover, some definitions of important variable names are listed.

Symbols

A *see* shortcut
 \downarrow *see* down closure
 \models *see* model
 \oplus *see* crest separator

Numbers

1-in-3 SAT *see* SAT
 3-Satisfiability *see* SAT
 3-coloring 41
 1912 *see* soccer
 1922 *see* soccer
 1930 *see* soccer
 1992 *see* soccer
 2000 *see* soccer

A

(A, \mathcal{A}^*) -replacing *see* function
 (\mathcal{A}^*, f) -essential *see* alley
 (\mathcal{A}^*, f) -forbidden *see* alley
 \mathcal{A}^* -idol *see* alley
 absolutely-minimal *see* coast separator
 add a set clockwise ... *see* embedding
 add an edge clockwise *see* embedding
 add an edge into a face *see* embedding
 adjacent *see* vertex
 African Cup of Nations ... *see* soccer
 agent pair *see* coloring
 alley 45, 46, 81, 82, 84, 86–90, 92, 94–97, 104, 106
 (\mathcal{A}^*, f) -essential 87, 88, 89, 92
 (\mathcal{A}^*, f) -forbidden 86
 A -alley 46
 \mathcal{A}^* -idol 86
 A -suballey 46
 crossing 46, 56, 59, 60, 66

cyclic alley 46, 94, 95, 96, 97, 99, 102–111
 depth 46
 endpoint 46
 extended alley condition 86
 good 86, 87, 89, 92
 inner vedge 79, 86
 inner vertex 46
 length 46, 79, 94
 maximum inner suballey 81
 non-planar crossing 46, 79
 planar crossing 46, 79
 s - t -connecting 46
 strong 81
 Z -skipping 78
 almost triangulated ... *see* embedded graph, embedded graph
 Alon 134
 ancestor 4
 proper 4
 Andrews 132
 appears clockwise around *see* embedding
 appears counter-clockwise around *see* embedding
 application 5
 approximation .. 2, 3, 24, 25, 42, 134, 172
 approximation ratio 3, 25, 41, 133–135, 171, 173, 174, 183
 approximative *see* embedding
 arbitrary *see* geometric object
 area *see* embedding
 arity *see* logic
 array ... 28, 36, 39, 44, 85, 87, 92, 152

B

backward vertex transformation .. 31, 100

bag *see* tree decomposition
 Baker 2, 113
 balanced *see* separator
 balanced version 26, 27, 33, 34
 Bar-Yehuda 3, 133, 139
 barrier 175, 176
 inner barrier 175
 outer barrier 175, 176, 177
 belongs to *see* graph
 benign *see* coloring
 biconnected *see* connectivity
 biconnected component *see*
 connectivity
 Bienstock 2, 113
 big bag *see* disjoint path problem
 binary ... *see* tree, tree decomposition
 block *see* connectivity
 block-cutpoint tree 47, 48, 108
 subtree component 125
 supertree component 125
 blockcut-point tree 111
 Bodlaender 1, 24, 71, 114
 Boolean formula *see* SAT
 border *see* crest separator
 border edge *see* crest separator
 Borodin 3
 bottom-up *see* traversal
 boundary *see* embedding,
 (\mathcal{S}, φ') -component
 boundary edge *see* embedding
 boundary vertex *see* embedding
 bounded variables *see* logic
 branch decomposition 41, 42
 branch width 41

C
C-connected *see* coloring
c-cost adaption *see* coloring
c-decision node *see* coloring
C-inner *see* (\mathcal{S}, φ') -component
C-internal face *see* embedding
C-outer *see* (\mathcal{S}, φ') -component
c-restricted set.. *see* geometric object
 canceled edge 30
 center *see* geometric object
 characteristic ... *see* coloring, disjoint
 path problem
 child *see* tree
 chord 11, 176
 chordal graph 3, 11, 12, 13,
 35–37, 40, 158, 159, 161, 166,
 167, 169, 170, 173
 chordless 11
 circular-arc graph.... *see* intersection
 graph
 clause *see* SAT
 clause gadget *see* disjoint path
 problem
 client 131, 133
 clique . 11, 12, 17, 19, 20, 35, 170, 174,
 175, 178–184
 maximal 37
 maximal clique 11, 37
 clique partition.... 170, 179–181, 184
 clique tree 3, 11, 35, 37, 159, 167, 169,
 170
 shrinking step 38, 39
 TDC 11, 37, 38
 TDC' 13, 38
 weak clique tree 13, 40, 159, 162,
 163
 clockwise order around *see*
 embedding
 clustering in pattern recognition ... 5
 coast 98, 100, 101, 104
 coast separator 98, 99–105, 107
 absolutely-minimal 100, 101, 103–
 105
 inner graph 100
 represent 99
 collection 4, 143
 colored graph *see* graph, 134
 coloring 2, 130, 132–134, 137, 142
 agent pair 137, 138
 benign 159, 160, 161, 163
 C-connected 131
 c-cost adaption 153, 154, 156
 c-decision node 153, 154–156
 characteristic 140, 141, 143, 145–
 147, 149, 150
 compatible 141, 145, 146–150,
 154
 consistent 145, 147, 149, 150, 153–
 156
 convex 130, 131, 132–138, 140,
 142, 144, 145, 148, 151–153,
 157
 cost 131, 132, 135, 137, 138, 140,
 142, 144, 150–153, 157

effective recoloring 139, 140, 141, 144
 extended in the same way 140
 extending 139, 143, 144, 153, 154
 extra rule 155
 forbidden color 140, 141, 143, 145–149, 151–153, 156
 free vertex 136, 137
 gadget 136, 137, 138
 good characteristic 145, 146–150, 153–156
 gray color 143–147, 149, 151
 initial coloring 131, 133, 134, 136, 139, 143, 145, 151, 153, 156
 legal recoloring 143, 144, 145, 148, 150–152
 literal-clause pair 137, 138
 LOW 145
 macro set 143, 145–152, 154, 155
 micro set 143, 144–147, 149, 151
 negative vertex 136, 137
 positive vertex 136, 137
 real color 130, 131, 135, 139, 142–147, 151, 156
 recolor 131
 recoloring 131, 132–138, 140, 142, 143, 145, 147–149, 151, 153–157, 159–161, 163–166
 represented by a characteristic 140
 restricted convex 137, 159, 163–166
 SEP 142
 sep 139
 uncolor 131
 uncolored 130
 weight reduction 157
 combinatorial embedding *see* embedding
 compatible *see* coloring, disjoint path problem
 complexity classes 4
 complexity parameter . 2, 3, 113, 139, 169, 170, 177, 179, 180
 component node *see* connectivity
 CONCACAF Gold Cup ... *see* soccer
 conform 8, 9
 conjunctive normal form *see* SAT
 connect *see* edge
 connected *see* graph, vertex
 connected component *see* connectivity, 33, 129
 connection face *see* embedding
 connective *see* logic
 connectivity 47
 biconnected 47, 49, 50, 122, 123
 biconnected component 47, 108, 124, 125
 biconnected components 125
 block 47
 component node 48
 connected component 13, 26, 30, 33, 44, 45, 53, 58, 64, 70, 74, 77, 109, 111, 124, 139–141, 146, 149, 157
 cutpoint 47, 48, 111, 125
 separation pair 47
 triconnected 47, 49, 53, 54, 56, 58–60, 64, 66, 68, 72, 82, 94, 109, 116, 117
 triconnected component 48, 49, 108–112, 122
 triconnectivity 59
 consistent *see* coloring
 contain *see* graph
 contained in *see* graph
 convex *see* coloring
 convex hull 176
 cop .. *see* one-robber- ℓ -cops-game, 22
 corresponds to 66
 cost *see* coloring
 count 3
 counter-clockwise order *see* embedding
 country 5
 Courcelle 2, 130
 cover *see* embedding
 crest 47, 50–52, 56, 59, 60, 63–69, 71, 72, 82, 96, 98, 101–103, 105–107, 109
 height 47
 crest alley 55, 56, 63, 79–83, 92, 94–96, 104, 106
 s_1 - s_2 -connecting 79
 crest separator 51, 55, 56–71, 73, 74, 77–85, 87, 91–94, 96–98, 102–107, 109, 110
 \in 55
 \oplus 55
 border 55

- border edge 55, 56, 58, 61–64, 66, 71
 - crossing-free 56, 57, 58, 61, 63, 64
 - enclose 60
 - essential border 55, 65, 71, 77–81, 83
 - extra edge 71, 72
 - go strongly between 56, 59, 60, 64–66
 - go weakly between 56, 60, 67, 69
 - height 55
 - induced by two alleys 55
 - largest 66
 - lowpoint 55, 56, 60, 61, 63, 65, 71, 77, 106
 - pseudo shortcut free 93, 94, 98, 102, 104, 106
 - size 55
 - top edge 57, 58–68, 70, 77, 78, 81, 82, 87, 91–93, 95, 96, 101–106, 109
 - top vertex 55, 59, 60, 65, 66, 69, 71, 74
 - crossing *see* alley
 - crossing-free *see* crest separator
 - cubic *see* SAT
 - curve 20
 - endpoint 20
 - point 20
 - cutpoint *see* connectivity
 - cycle 4, 29, 43
 - directed 4
 - length 4
 - undirected 4
 - cycle component 49, 122
 - cyclic alley *see* alley
- D**
- decision version 129, 130
 - decompose _{k, α} 101
 - degree *see* vertex
 - depth *see* tree, alley, path
 - depth-first search *see* DFS
 - descendant 1, 4
 - proper 4
 - deterministic bottom-up tree automaton *see* logic
 - DFS 28, 29, 30, 48, 103, 130, 142
 - depth-first search 27
 - DFS-forest 28
 - DFS-tree 28, 33, 48
 - discovery time 28, 48
 - finishing time 28, 48
 - start a DFS 28
 - tree edge 28
 - directed *see* graph, cycle
 - directed version *see* edge, graph
 - disconnected *see* graph, vertex
 - discovery time *see* DFS
 - disjoint path
 - edge-disjoint path 28–31
 - k -disjoint path 3
 - terminal 159, 160–162, 167, 168
 - vertex-disjoint path 28, 31, 32, 136, 137
 - disjoint path problem 3
 - big bag 162, 163
 - characteristic 159, 161
 - clause gadget 167
 - compatible 160, 161
 - full characteristic 159, 160, 161
 - gadget 167, 168
 - initial coloring 159, 161, 162
 - ℓ -VDPP 129, 158, 160–163, 166
 - ℓ -vertex-disjoint path problem 129, 130, 158, 166
 - reduced characteristic 159, 160, 161
 - small bag 162
 - square vertex 167, 168
 - suitable 160
 - triangular vertex 167, 168
 - valid 159, 160
 - variable gadget 167, 168
 - VDPP 129, 132, 159, 167, 168
 - vertex-disjoint path problem 129
 - $(w_1, w_2)_{\bar{B}}$ -count 162
 - disk *see* geometric object
 - disk graph *see* intersection graph
 - distance *see* vertex, embedding
 - divide-and-conquer approach i, 13, 16
 - domain list *see* function
 - dominated on the left 59
 - dominated on the right 59
 - dominating set 3, 41, 171
 - down closure 51, 52–54, 57–60, 66, 82, 83
 - \downarrow 51
 - down edge 51, 58, 61, 62, 64, 66, 72

down path 57, 58, 59, 66, 84
 down vertex 51, 54, 57, 62
 down edge *see* down closure
 down path *see* down closure
 down vertex *see* down closure
 down-connected 71
 dual graph 43, 58
 crossing 43, 58
 duplicate 72, 73, 74, 77

E

edge 3, 19
 connect 3
 directed version 42
 endpoint 3
 head 3
 horizontal 47
 incident 4
 tail 3
 unseen 19
 vertical 47
 virtual edge 49, 62, 63, 72, 76, 77,
 118–121, 123, 124
 edge-constrained optimal *see*
 embedding
 edge-contraction *see* minor
 edge-disjoint path .. *see* disjoint path
 effective recoloring *see* coloring
 elementary set theory 4
 embed *see* embedding
 embeddable *see* embedding
 embedded graph 43
 almost triangulated 43, 53, 54
 down-connected 50
 triangulated 43
 embedding 2, 23, 56, 169
 add a set clockwise 117
 add an edge clockwise 117
 add an edge into a face 45, 46
 appear around 43
 appears clockwise around 43
 appears counter-clockwise around
 43
 approximative 116, 122, 126
 area 44
 boundary 44
 boundary edge 44, 53, 54, 56, 66
 boundary edges 64
 boundary vertex 44, 53, 54, 66
 C-internal face 120

clockwise order around 43
 combinatorial embedding 42, 43–
 46, 55, 61, 71, 116, 123
 connection face 46
 counter-clockwise order 43
 cover 44
 distance 116
 edge-constrained optimal 121,
 122–125
 embed 46
 embeddable 117, 118, 121
 face 20, 43
 face-adjacent 46
 face-connected 46
 fence 43, 44, 56, 86, 87, 89–93
 fence edge 44, 61–64, 79, 86, 90,
 91
 fence vedge 79, 81, 91
 fence vertex 44, 61, 92
 geometrical embedding 20
 incident 20, 43, 44, 46
 inherited embedding 45
 inherited embedding of induced
 graphs 118
 inner face 43, 46, 50, 51, 53, 55,
 56, 58, 61, 64, 76
 inner graph 94, 118, 120
 leave on a side 44
 mapped 118
 non-connection face 46
 on different sides 44
 on the boundary 20, 43
 on the same side 44
 optimal 116, 122
 outer face 20, 43, 44–47, 50, 58–
 60, 64, 70, 72
 outerplanar 47, 70, 77, 78, 101,
 113, 116
 outside 117, 121
 path on the boundary 43
 planar 20, 23, 42, 43
 rooted 43, 45, 55, 56, 71
 splitting an edge into two edges 78
 splitting vertex 78
 strictly inner graph 94
 subarea 44
 vertex-constrained approximative
 125, 126
 vertex-constrained optimal 121,
 122–126

- enclose *see* crest separator
 - endpoint .. *see* edge, path, curve, alley
 - Eschenhof *i*
 - essential border .. *see* crest separator
 - Euler's formula 44
 - exponential growth 24
 - extended *see* (\mathcal{S}, φ') -component
 - extended alley condition *see* alley
 - extended component 61
 - extended peeling *see* peeling
 - extended peeling number . *see* peeling
 - extended row 114, 115
 - extending change *see* peeling
 - extra edge *see* crest separator
 - extra rule 154
- F**
- face *see* embedding, embedding
 - face-adjacent *see* embedding
 - face-connected *see* embedding
 - fat *see* geometric object
 - fat object *see* geometric object
 - fat-object graph *see* intersection graph
 - fatness 171
 - feasible solution *see* optimization problem
 - Feige 41
 - Feldman 3, 133, 139
 - fence *see* embedding, *see* (\mathcal{S}, φ') -component
 - fence edge *see* embedding
 - fence vertex *see* embedding
 - findOrder 35, 36
 - finishing time *see* DFS
 - first-order logic *see* logic
 - fixed-parameter algorithm 2, 113, 139
 - fixed-parameter algorithms 158
 - forbidden color *see* coloring
 - forest 4, 28, 137
 - Fortune 158
 - free variable *see* logic
 - free vertex *see* coloring
 - full characteristic ... *see* disjoint path problem
 - function
 - (A, \mathcal{A}^*) -replacing 90
 - domain list 86
 - fundamental cycle 75, 76, 77
- G**
- gadget *see* coloring, disjoint path problem
 - generalization ... 3, 8, 16, 31, 45, 116, 127, 130, 145, 169, 173
 - geometric object
 - arbitrary 171
 - c -restricted set 171
 - center 175
 - disk 169, 171, 173
 - fat 171
 - fat object 171, 173
 - inball 175
 - iso-oriented 174, 177
 - outball 175
 - regular polygon 171, 173, 177
 - size 171
 - square 169, 175–177
 - t -fat-object 171, 173, 175
 - triangle 177
 - unit 171
 - geometrical embedding *see* embedding
 - go strongly between *see* crest separator
 - go weakly between *see* crest separator
 - gold medal 5
 - good ... *see* mountain structure, alley
 - good characteristic *see* coloring
 - good peninsula *see* peninsula
 - grandchild *see* tree
 - grandparent *see* tree
 - graph 3
 - acyclic 158
 - belongs to 3
 - colored graph 130, 143
 - connected 4
 - contain 4
 - contained in 3
 - directed 3
 - directed version 42
 - disconnected 4
 - empty graph 3
 - induced 4
 - induced graph 49
 - multigraph 43
 - outerplanar 47, 58, 66, 71, 113, 114, 116
 - part of 3

- planar 2, 20, 22, 23, 41–45, 49, 62, 65, 73, 113, 114, 116, 117, 122–126
 - size 3
 - triconnected 94, 99–101
 - undirected 3
 - union 5
 - weighted 4
 - gray color 142
 - Y 142
 - grid 20, 22, 114
 - $(\ell_1 \times \ell_2)$ -grid 20
 - Gu 114
 - Gutwenger 108
- H**
- H -attached component *see* peninsula
 - h -high *see* pseudo shortcut
 - Hagerup i
 - Hajiaghayi 41
 - Hamiltonian circuit 41
 - head *see* edge, path
 - height *see* tree, rift, crest, crest separator
 - helicopter *see* one-robber- ℓ -cops-game
 - Hopcroft 158
 - horizontal *see* edge
- I**
- inball *see* geometric object
 - incident *see* edge, embedding, embedding
 - increasing change *see* peeling
 - independent dominating set 171
 - independent set 3, 5, 41, 129, 130, 139
 - individual variable *see* logic
 - induced *see* graph
 - induced by two alleys *see* crest separator
 - induced graph *see* graph
 - induced set of extra edges *see* peeling
 - information retrieval 5
 - inherited embedding .. *see* embedding
 - inherited embedding of induced graphs *see* embedding
 - initial coloring .. *see* coloring, disjoint path problem
 - inner barrier *see* barrier
 - inner face *see* embedding
 - inner graph *see* embedding, coast separator
 - inner tree decomposition problem 108
 - ITD 108, 111
 - inner vedge *see* alley, alley
 - inner vertex *see* tree, alley
 - inside peeling number *see* peeling
 - instance ... *see* optimization problem
 - integer linear program *see* linear program
 - internal *see* path
 - internally H -avoiding path *see* peninsula
 - intersection graph .. 3, 169, 170–173, 175, 177
 - circular-arc graph 169, 171
 - disk graph 169, 171–174
 - fat-object graph 169, 171–173
 - polygon graph 169, 171
 - regular-polygon graph 169, 171–173
 - square graph 169, 174, 175
 - t -fat-object graph 169, 172, 173
 - t -interval graph 169, 171–174
 - unit-disk graph 169, 171, 174
 - unit-square graph 169, 174
 - universe 171, 175, 177
 - intree *see* tree
 - IS *see* logic
 - iso-oriented *see* geometric object
- J**
- j -inner graph 108
 - JOINS *see* logic
- K**
- k -normal *see* tree decomposition
 - k -perfectly groupable .. 174, 175, 177, 179, 180
 - k -perfectly orientable 174, 175, 177–182
 - k -simplicial ... 174, 175, 177, 179–181
 - successor 174, 178–181
 - k -simplicial elimination order ... 174, 177, 179–181
 - k -tree 18
 - k -tree 16
 - partial 16, 18
 - Kammer 3
 - Karp 158

Knuth 132, 158

L

\mathcal{L} *see* peeling

ℓ -clique 11

ℓ -long *see* pseudo shortcut

ℓ -vertex-disjoint path problem *see* disjoint path problem

label *see* logic

labeled graph *see* logic

largest *see* crest separator

leaf 4

leave on a side *see* embedding

Lee 41

left *see* pseudo shortcut

left child *see* tree

legal recoloring *see* coloring

length *see* path, cycle, alley

linear program 181

 integer linear program 181

 objective function 181

 relaxation 181

 solution 181

literal *see* SAT

literal-clause pair *see* coloring

load 76

local ratio technique 3, 157

logic 4, 127–129

 arity 128

 bounded variables 127, 128

 connective 127, 128

 deterministic bottom-up tree automaton 129

 first-order logic 127, 128, 129

 formula 127, 128

 free variable 127, 128

 individual variable 128

 IS 129, 130

 JOINS 129

 label 127, 128, 129

 labeled graph 127

 labeled tree 130

 monadic second-order logic 128

 model 128, 129, 130

 monadic predicate 128

 monadic second order 2

 monadic second-order logic 41

 MSOL 41, 128, 129, 130

 PATH $_{\ell}$ 129

 predicate 127, 128, 129

 propositional 4

 quantifier 127, 128

 set for the label 127

 set variable 128

 SIZE $_{\geq \ell}$ 130

 universe 127, 128

LOW *see* coloring

low-cost vertex 84

lowpoint *see* crest separator

Lynch 132, 158

M

macro set *see* coloring

mapped *see* embedding

maximal clique *see* clique

maximum 3-coloring 2

maximum clique 170, 173, 174

 MC 170, 172

maximum independent set . 2, 5, 114, 139

 MIS 5, 7–10, 12, 172

maximum inner suballey *see* alley

maximum triangle matching 2

maximum weighted clique .. 170, 180, 182, 183

 MWC 170, 172, 173

maximum weighted independent set .. 170, 180, 182

 MWIS 170, 172, 173, 182

MBRP *see* minimum block recoloring problem

MC *see* maximum clique

MCP .. *see* minimum clique partition

MCRP *see* minimum convex recoloring problem

MDS ... *see* minimum dominating set

micro set *see* coloring

MIDS *see* minimum independent dominating set

minimization problems 132

minimum block recoloring problem ... 132

 MBRP 132, 134, 135, 151, 152

minimum clique partition .. 170, 173, 179, 180, 182–184

 MCP 170, 172, 173

minimum convex recoloring problem . 131

 MCRP 131–136, 139–141, 143, 148, 151, 153, 156, 157

- minimum dominating set . 2, 171, 180
 - MDS 171–173
 - minimum edge dominating set 2
 - minimum independent dominating set
 - 171, 180
 - MIDS 171–173
 - minimum maximal matching 2
 - minimum restricted convex recoloring
 - problem 131
 - MRRP 131–137, 151, 152, 156, 157
 - minimum set cover 134
 - minimum vertex coloring . . . 170, 171, 182, 183
 - MVC 171, 172
 - minimum vertex cover 2
 - minor 22, 23, 45, 70, 99, 101, 108, 109
 - edge-contraction 22, 23
 - minor-operation 22, 23
 - MIS . . *see* maximum independent set, 6, 130
 - monadic second-order logic . *see* logic
 - model *see* logic
 - monadic predicate *see* logic
 - Monma 2, 113
 - monotone *see* SAT
 - Moran 131–133
 - Moshkovitz 134
 - mountain . . . 47, 50, 54, 56, 57, 62, 65, 72, 74
 - mountain connection tree . 64, 65, 66, 68, 78, 98, 101, 102, 104, 110
 - mountain structure 56, 59, 60–62, 64–66, 68, 70, 77, 78, 81, 82, 93, 96, 98, 103, 109
 - good 65, 66, 68, 78, 89, 93, 96, 101, 104
 - MRRP *see* minimum restricted convex recoloring problem
 - MS 57, 58–60, 64–66
 - MSOL *see* logic
 - multicast communications 131
 - multigraph *see* graph
 - multiple-edge component 49, 122
 - Mutzel 108
 - MVC . . *see* minimum vertex coloring
 - MWC . *see* maximum weighted clique
 - MWIS *see* maximum weighted independent set
- N**
- natural numbers 4
 - negative vertex *see* coloring
 - neighbor *see* vertex
 - nice *see* tree decomposition
 - node *see* tree decomposition
 - non-connection face . . *see* embedding
 - non-planar crossing *see* alley
 - normal *see* k -normal
 - NP-complete 1
 - NP-hard . . . 1–3, 41, 42, 130, 132, 133, 136, 172, 179, 180, 183
 - \mathcal{N}_φ *see* peeling
- O**
- objective function *see* linear program
 - Okamoto 3
 - on different sides *see* embedding
 - on the boundary *see* embedding
 - on the same side *see* embedding
 - one-robber- ℓ -cops-game . . 21, 22, 114
 - cop 21, 22, 114, 115
 - helicopter 22, 115
 - raw deal 21
 - robber 21, 22, 114, 115
 - tracking transmitter 21
 - optical wavelength division multiplexing networks 131
 - optimal *see* embedding
 - optimization problem 24
 - feasible solution 24, 25
 - instance 24
 - value 24
 - order *see* path
 - outball *see* geometric object
 - outer barrier *see* barrier
 - outer component *see* peninsula
 - outer face *see* embedding, embedding
 - outerplanar . . . *see* graph, embedding
 - outerplanarity index 2, 113, 114, 116, 118, 125, 126
 - weighted 116, 117
 - outside *see* embedding
 - outside peeling number . . . *see* peeling
- P**
- packing into a tree 1, 7
 - parent *see* tree
 - parent edge *see* SPQR tree
 - part of *see* graph

partial *see k-tree*
 participant 5
 path 4
 depth 46
 endpoint 4
 head 4
 internal 4, 28, 31, 32, 136
 length 4
 order 4
 tail 4
 undirected 4
 v_1-v_ℓ -path 4
 path on the boundary *see embedding*
 PATH_ℓ *see logic*
 peeling 47
 extended peeling 117, 118–122,
 124, 125
 extended peeling number 117,
 120, 121, 123–126
 extended weight function 122
 extending change 121
 increased weight function 124, 125
 increasing change 121
 induced set of extra edges 121
 inside peeling number 123
 \mathcal{L} 119
 \mathcal{N}_φ 47
 outside peeling number 123
 peeling index 47, 123
 peeling number 46, 47, 50–54, 56–
 60, 62–66, 69, 70, 72–76, 80–
 83, 91, 93–98, 102–105, 108–
 111
 peeling step 117, 119, 120, 123,
 126
 \mathcal{P} 117
 total peeling number 117, 121,
 123
 peeling index *see peeling*
 peeling number *see peeling*
 peninsula 48, 49, 117–119, 122
 good peninsula 118, 121, 122
 H -attached component 48, 118,
 119, 121, 122
 internally H -avoiding path 48
 outer component 118, 119
 \mathcal{R}_H 49
 perfect elimination order .. 35, 36–38
 successor 35, 36, 37
 Perković 158
 Perl 158
 planar *see embedding, graph*
 planar crossing *see alley*
 point *see curve*
 pointer 28, 36
 polygon 3
 polygon graph *see intersection graph*
 polynomially bounded growth ... 173
 positive *see weight-function*
 positive vertex *see coloring*
 predicate *see logic*
 proper *see ancestor, descendant*
 propositional *see logic*
 pseudo A -shortcut *see pseudo*
 shortcut
 pseudo shortcut 80
 h -high 93
 ℓ -long 93
 left 96
 pseudo A -shortcut 80
 right 96
 s_1 - s_2 -connecting 80
 Z -skipping 80
 pseudo shortcut free *see* (\mathcal{S}, φ') -
 component, crest separator
 \mathcal{P} *see peeling*

Q
 quantifier *see logic*

R
 ranking 7
 raw deal . *see one-robber- ℓ -cops-game*
 Rawith 3
 Rawitz 133, 139
 real color *see coloring*
 recolor *see coloring*
 recoloring *see coloring*
 recursion 16, 34
 reduced characteristic *see disjoint*
 path problem
 reduced subtree module ... *see SPQR*
 tree
 reduction 134, 136, 137, 166, 167, 179
 Reed 25, 158
 regular polygon . *see geometric object*
 regular-polygon graph *see*
 intersection graph
 relax over a face 84
 relaxation *see linear program*

relaxing over a function f' from u 88
 relaxing over a vertex 84
 relaxing over the fence from u 88
 represent *see* coast separator
 representant 33, 57
 representation 180
 residual graph 29
 \mathcal{R}_H *see* peninsula
 ridge 54, 58–60, 63–66, 69
 rift 35, 36
 height 35
 v_i - v_j -rift 35
 right *see* pseudo shortcut
 right child *see* tree
 robber .. *see* one-robber- ℓ -cops-game,
 22
 Robertson 1, 158
 Röhrig 1, 24
 root *see* tree
 rooted .. *see* tree, tree decomposition,
 embedding
 rooted SPQR tree *see* SPQR tree
 router 131–133
 routing problem 131
 running 34
 running time 2, 11, 12, 24, 25, 33,
 34, 37, 39, 40, 58, 69, 77, 113,
 114, 130, 141, 144, 149, 150,
 152, 156–161, 166, 171, 183

S
 (\mathcal{S}, φ') -area 56
 (\mathcal{S}, φ') -component . 56, 58, 60–64, 70,
 77, 78, 102–107, 109
 boundary 56
 C -inner 56, 60, 63, 64, 91, 103,
 104, 107
 C -outer 56, 61, 63
 extended 56, 60, 61, 62, 63, 70, 71,
 77
 fence 56
 pseudo shortcut free 93, 98, 102,
 106
 virtual edge 61
 virtual vertex 61
 (\mathcal{S}, φ') -connected 55, 56
 s - t -connecting *see* alley
 s_1 - s_2 -connecting *see* crest alley,
 shortcut, pseudo shortcut
 Safra 134

SAT
 1-in-3 SAT 166
 3-CNF 136
 3-Satisfiability 136
 3-SAT 136
 Boolean formula 136, 166, 167
 clause 136, 137, 166–168
 conjunctive normal form 136, 166
 cubic 167
 literal 136, 137
 monotone 167
 variable 137, 167
 scheduling 5
 SEP *see* coloring
 sep *see* coloring
 separation pair *see* connectivity
 separator .. 1, 13, 16, 25–27, 31–33, 69
 balanced 26, 27, 33, 34
 disconnect 29–32, 34, 47, 52, 97,
 111
 size 13, 26
 strongly disconnect 53, 69, 70, 99
 unweighted 26, 32
 weak 26, 32, 33
 weakly disconnect 52, 97
 weight condition 26, 27, 33
 weighted 26, 32, 33, 34
 X-separator 26, 27, 28, 32–34
 sequentially k -independent graphs ...
 173
 set cover 134, 135
 size 134, 135
 universe 134
 set for the label *see* logic
 set variable *see* logic
 Seymour 1, 41, 114, 158
 Shiloach 158
 shortcut 79
 A 79
 s_1 - s_2 -connecting 79
 Z -skipping 79
 shrinking step *see* clique tree
 signal transmission 5
 simple-exponential 9
 simulation 129
 size *see* graph, tree decomposition, sep-
 arator, separator, crest sep-
 arator, set cover, geometric
 object
 $\text{SIZE}_{\geq \ell}$ *see* logic

small bag .. *see* disjoint path problem
 Snir 131–133
 soccer
 1912 5, 14
 1922 12
 1930 11
 1992 14
 2000 17
 African Cup of Nations 14
 CONCACAF Gold Cup 17
 soccer contest 5
 soccer teams 5
 South American Championship 12
 World Cup 11
 solution *see* linear program
 South American Championship .. *see*
 soccer
 sparsification 3
 sparsification technique 159
 split-component 49
 splitting an edge into two edges .. *see*
 embedding
 splitting vertex *see* embedding
 SPQR tree . 49, 50, 108, 110, 111, 122
 parent edge 122
 reduced subtree module 122, 123
 rooted SPQR tree 122, 124
 subtree module 122, 123
 square *see* geometric object
 square graph .. *see* intersection graph
 square vertex *see* disjoint path
 problem
 starting position 21
 stereo vision correspondence 5
 strictly inner graph .. *see* embedding
 strong *see* alley
 strongly disconnect *see* separator
 subarea *see* embedding
 subgraph 3, 19
 subtree *see* tree
 subtree component *see* block-cutpoint
 tree
 subtree module *see* SPQR tree
 successor .. *see* perfect elimination
 order, *see* k -simplicial, k -
 simplicial
 suitable ... *see* disjoint path problem
 Summer Olympics 5
 supergraph 4
 supertree component *see*
 block-cutpoint tree
T
t-fat-object *see* geometric object
t-fat-object graph *see* intersection
 graph
t-interval 171, 175
t-interval graph *see* intersection graph
 tail *see* edge, path
 Tamaki 42, 114
 Tarjan 36
 TD1 *see* tree decomposition, tree
 decomposition
 TD2 *see* tree decomposition, tree
 decomposition
 TDC *see* clique tree
 TDC' *see* clique tree
 TD_k 26, 33, 34
 telecommunication network 131
 terminal *see* disjoint path
 terminology 3
 Thales' theorem 176
 Thatcher 129
 Tholey i, 2, 3
 Thomas 41, 114
 tied *see* tree decomposition
 top edge *see* crest separator
 top vertex *see* crest separator
 top-down *see* traversal
 total peeling number *see* peeling
 tournament 5, 11
 knockout tournament 5, 11, 14
 representation 5, 11
 tracking transmitter *see*
 one-robber- ℓ -cops-game
 transformation 31
 transition set 159
 transportation network 131
 traversal .. 1, 4, 6–9, 15, 18, 24, 33, 38
 bottom-up 1, 4, 5, 6–9, 24, 33, 67,
 93, 139, 140, 143–145, 151–
 154, 160
 top-down 7, 9, 38, 93, 140, 144,
 145, 151, 160, 161, 166
 tree 1–3, 4, 5–7, 9, 13, 15,
 16, 19, 38, 130–133, 137, 139,
 142, 159, 169, 170, 175, 177
 binary 4
 child 4

- depth 4
- grandchild 125
- grandparent 125
- height 4
- inner vertex 4
- intree 103
- left child 4
- parent 4
- right child 5
- root 4
- rooted 4
- subtree 4
- tree decomposition
 - 1, 3, 7, 8, 13–21, 23–27, 34, 37, 41, 42, 45, 47, 69–71, 73, 75–78, 98, 108–114, 116, 130, 141, 142, 159, 179
- bag 7, 9, 11, 13, 15, 16, 18–20, 25, 26, 37, 39, 69–71, 75, 77, 78, 108–111, 142, 153, 155, 156, 170, 179
- binary 8
- k -normal 14, 16–19, 21, 27, 179
- nice 142
- node 3, 8, 19
- rooted 8, 15, 25
- size 8, 40
- TD1 7, 16, 18, 22, 25, 78
- TD2 7, 16, 18, 22, 25, 26, 37, 38, 77, 78
- tied 25
- width 8, 18, 23, 26, 41, 42, 70, 74–78
- X-tied 25
- tree-like 16
- $\text{TreeComp}_{k,\alpha}$ 108, 110–112
- $\text{TreeComp}_{k,\alpha}$ 108
- treewidth 1, 2, 8, 9, 13, 14, 16, 18–25, 27, 33, 34, 37, 40–42, 69, 70, 101–103, 107–109, 111–114, 116, 139, 141, 158, 159, 161, 163, 169–171, 177, 179
- bounded treewidth 2, 3, 8, 130, 133, 134, 139, 143, 144, 152, 156, 157
- logarithmically bounded 9
- triangle matching 41
- triangular vertex ... *see* disjoint path problem
- triangulated *see* embedded graph
- triconnected *see* connectivity
- triconnected component *see* connectivity
- T_w 4
- U**
- Uehara 3
- uncolor *see* coloring
- uncolored *see* coloring
- undirected *see* graph, cycle, path
- union *see* graph
- unit *see* geometric object
- unit-disk graph *see* intersection graph
- unit-square graph *see* intersection graph
- universe *see* logic, set cover, intersection graph
- Uno 3
- unseen *see* vertex, edge, vertex
- unweighted *see* separator
- up-connected 71
- V**
- valid *see* disjoint path problem
- value *see* optimization problem
- variable *see* SAT
- variable gadget *see* disjoint path problem
- VDPP *see* disjoint path problem
- vedge 78
- vertex 3
 - adjacent 3
 - connected 4
 - degree 3
 - disconnected 4
 - distance 4
 - neighbor 3
 - unseen 19, 28
- vertex-constrained approximative *see* embedding
- vertex-constrained optimal *see* embedding
- vertex-disjoint path *see* disjoint path, 159–161
- vertex-disjoint path problem *see* disjoint path problem, 159
- vertex-disjoint-to-edge-disjoint version 31, 100
- vertical *see* edge

virtual edge *see* edge,
 (\mathcal{S}, φ') -component, edge
 virtual vertex . *see* (\mathcal{S}, φ') -component
 Voepel 3

W

$(w_1, w_2)_{\bar{B}}$ -count *see* disjoint path
 problem
 Wagner's theorem 20
 weak *see* separator
 weak clique tree *see* clique tree
 weakly disconnect *see* separator
 weight 161
 weight condition *see* separator
 weight-function 4
 positive 4
 weighted *see* graph, separator,
 outerplanarity index
 width *see* tree decomposition
 winner 5
 winning strategy 21, 22, 115
 Woeginger 158
 Wright 129
 Wyllie 158

X

X-separator *see* separator, 33, 34
 X-tied *see* tree decomposition

Y

Y *see* gray color
 Yannakakis 36
 Ye 3

Z

Z-skipping *see* alley, shortcut, pseudo
 shortcut
 Zhang 132

Bibliography

- [1] K. Akcoglu, J. Aspnes, B. DasGupta, and M.-Y. Kao. Opportunity cost algorithms for combinatorial auctions. *CoRR*, **cs.CE/0010031**, 2000.
- [2] N. Alon, D. Moshkovitz, and S. Safra. Algorithmic construction of sets for k -restrictions. *ACM Transactions on Algorithms*, pp. 153–177, 2006.
- [3] M. Andrews and L. Zhang. Hardness of the undirected edge-disjoint paths problem. Proc. 37th Annual ACM Symposium on Theory of Computing (STOC 2005), pp. 276–283, 2005.
- [4] S. Arnborg, D. G. Corneil, and A. Proskurowski. Complexity of finding embeddings in a k -tree. *SIAM J. Alg. Disc. Math.*, **8**: pp. 277–284, 1987.
- [5] S. Arnborg and J. Lagergren. Finding minimal minors using a finite congruence. Proc. of the 18th International Colloquium on Automata, Languages and Programming (ICALP 91), pp. 544–555, 1991.
- [6] G. Augustson and J. Minker. An analysis of some graph-theoretical clustering techniques. *J. ACM* **17**, **4**: pp. 571–588, 1970.
- [7] B. S. Baker. Approximation algorithms for NP-complete problems on planar graphs. *Journal of the ACM*, **41**: pp. 153–180, 1994.
- [8] E. Balas and C. S. Yu. Finding a maximum clique in an arbitrary graph. *SIAM J. Comput.* **15**, **4**: pp. 1054–1068, 1986.
- [9] R. Bar-Yehuda, I. Feldman, and D. Rawitz. Improved approximation algorithm for convex recoloring of trees. Third International Workshop on Approximation and Online Algorithms, (WAOA '05), LNCS 3879, pp. 55–68, 2006.
- [10] R. Bar-Yehuda, M. M. Halldórsson, J. Naor, H. Shachnai, and I. Shapira. Scheduling split intervals. *SIAM J. Comput.*, **36**: pp. 1–15, 2006.
- [11] G. D. Battista and R. Tamassia. Incremental planarity testing. Proc. 30th IEEE Symp. on Foundations of Computer Science, pp. 436–441, 1989.
- [12] D. Bienstock and C. L. Monma. On the complexity of embedding planar graphs to minimize certain distance measures. *Algorithmica*, **5**: pp. 93–109, 1990.
- [13] H. L. Bodlaender. A tourist guide through treewidth. *Acta Cybernet.*, **11**(1–2): pp. 1–23, 1993.

- [14] H. L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, **25**(6): pp. 1305–1317, 1996.
- [15] H. L. Bodlaender. A partial k -arboretum of graphs with bounded treewidth. *Theoret. Comput. Sci.*, **209**(1–2): pp. 1–45, 1998.
- [16] H. L. Bodlaender and T. Kloks. Better algorithms for path width and tree width. Proc. of the 18th International Colloquium on Automata, Languages and Programming (ICALP 91), pp. 538–543, 1991.
- [17] P. Buneman. A characterisation of rigid circuit graphs. *Discrete Math*, **9**: pp. 205–212, 1974.
- [18] S. Butenko. Maximum independent set and related problems, with applications. Dissertation, University of Florida, 2003.
- [19] A. Butman, D. Hermelin, M. Lewenstein, and D. Rawitz. Optimization problems in multiple-interval graphs. Proc. 18th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2007), pp. 268–277, 2007.
- [20] M. R. Cerioli, L. Faria, T. O. Ferreira, and F. Protti. On minimum clique partition and maximum independent set on unit disk graphs and penny graphs: complexity and approximation. *Electronic Notes in Discrete Mathematics*, **18**: pp. 73–79, 2004.
- [21] T. M. Chan. Polynomial-time approximation schemes for packing and piercing fat objects. *J. Algorithms*, **46**: pp. 178–189, 2003.
- [22] X. Chen, X. Hu, and T. Shuai. Inapproximability and approximability of maximal tree routing and coloring. *J. Comb. Optim.*, **11**: pp. 219–229, 2006.
- [23] N. Chiba, T. Nishizeki, S. Abe, and T. Ozawa. A linear algorithm for embedding planar graphs using PQ-trees. *J. Comput. System Sci.*, **30**: pp. 54–76, 1985.
- [24] B. N. Clark, C. J. Colbourn, and D. S. Johnson. Unit disk graphs. *Discrete Math.*, **86**: pp. 165–177, 1990.
- [25] B. Courcelle. Graph rewriting: An algebraic and logic approach. *Handbook of Theoretical Computer Science, Vol. B: Formal Models and Semantics (J. van Leeuwen, ed.)*, Elsevier, Amsterdam, pp. 194–242, 1990.
- [26] B. Courcelle. The monadic second-order logic of graphs. i. recognizable sets of finite graphs. *Inform. and Comput.*, **85**: pp. 12–75, 1990.
- [27] R. O. Duda and P. E. Hart. Unsupervised learning and clustering. Section 6:11 in: *Pattern Classification and Scene Analysis*, Wiley, New York, Chapter 6., 1973.
- [28] T. Erlebach, K. Jansen, and E. Seidel. Polynomial-time approximation schemes for geometric intersection graphs. Proc. 12th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA), pp. 671–679, 2001.

- [29] T. Erlebach and E. J. van Leeuwen. Domination in geometric intersection graphs. Proc. of the 8th Latin American Symposium (LATIN 2008), LNCS 4957, pp. 747–758, 2008.
- [30] U. Feige, M. Hajiaghayi, and J. R. Lee. Improved approximation algorithms for minimum weight vertex separators. *SIAM J. Comput.*, **38**(2): pp. 629–657, 2008.
- [31] S. Fortune, J. Hopcroft, and J. Wyllie. The directed subgraph homeomorphism problem. *Theoret. Comput. Sci.*, **10**: pp. 111–121, 1980.
- [32] R. J. Fowler, M. S. Paterson, and S. L. Tanimoto. Optimal packing and covering in the plane are NP-complete. *Inform. Process. Lett.*, **12**: pp. 133–137, 1981.
- [33] A. Frank. Some polynomial algorithms for certain graphs and hypergraphs. Proc. 5th British Combinatorial Conference (Aberdeen 1975), Congr. Numer. XV, pp. 211–226, 1976.
- [34] H. B. Hunt III, M. V. Marathe, V. Radhakrishnan, S. S. Ravi, D. J. Rosenkrantz, and R. E. Stearns. NC-approximation schemes for NP- and PSPACE-hard problems for geometric graphs. *J. Algorithms*, **26**: pp. 238–274, 1998.
- [35] T. Gallai. Elementare Relationen bezüglich der Glieder und trennenden Punkte von Graphen. *Magyar Tud. Akad. Mat. Kutato Int. Kozl.*, **9**: pp. 235–236, 1964.
- [36] M. R. Garey and D. S. Johnson. Computers and intractability, a guide to the theory of NP-completeness. W. H. Freeman and Co., San Francisco, Calif., 1979.
- [37] M. R. Garey, D. S. Johnson, and L. Stockmeyer. Some simplified NP-complete graph problems. *Theoret. Comput. Sci.*, **1**: pp. 237–267, 1976.
- [38] D. R. Gaur and R. Krishnamurti. Scheduling intervals using independent sets in claw-free graphs, computational science and its applications. Proc. Computational Science and Its Applications (ICCSA 2003), LNCS 2667, pp. 254–266, 2003.
- [39] F. Gavril. Algorithms for minimum coloring, maximum clique, minimum covering by cliques, and maximum independent set of a chordal graph. *SIAM J. Comput.*, **1**: pp. 180–187, 1972.
- [40] F. Gavril. The intersection graphs of subtrees in trees are exactly the chordal graphs. *Journal of Combinatorial Theory, Series B*, **16**: pp. 47–56, 1974.
- [41] A. Gräf. Coloring and recognizing special graph classes. Technical Report Musikinformatik und Medientechnik Bericht 20/95, Johannes Gutenberg-Universität Mainz, 1995.
- [42] J. R. Griggs and D. B. West. Extremal values of the interval number of a graph. *SIAM Journal on algebraic and discrete methods*, **1**: pp. 1–7, 1980.

- [43] Q. P. Gu and H. Tamaki. Optimal branch-decomposition of planar graphs in $O(n^3)$ time. Proc. of the 32nd International Colloquium on Automata, Languages and Programming (ICALP 05), LNCS 3580, pp. 373–384, 2005.
- [44] Q. P. Gu and H. Tamaki. Optimal branch-decomposition of planar graphs in $O(n^3)$ time. *ACM Trans. Algor.*, **4**(3): pp. 1–13, 2008.
- [45] C. Gutwenger and P. Mutzel. A linear time implementation of SPQR-trees. Graph Drawing (GD 2000), LNCS 1984, pp. 77–90, 2001.
- [46] T. Hagerup. A very practical algorithm for the two-paths problem in 3-connected planar graphs. Proc. 33rd International Workshop on Graph-Theoretic Concepts in Computer Science (WG 2007), LNCS Vol. 4769, Springer, Berlin, 2007, **4769**: pp. 145–150, 2007.
- [47] F. Harary and G. Prins. The block-cutpoint-tree of a graph. *Publicationes Mathematicae Debrecen*, **13**: pp. 103–107, 1966.
- [48] R. Horaud and T. Skordas. Stereo correspondence through feature grouping and maximal cliques. *IEEE Trans. Pattern Anal. Mach. Intell.* **11** (11), pp. 1168–1180, 1989.
- [49] J. L. Hurink and T. Nieberg. Approximating minimum independent dominating sets in wireless networks. *Inform. Process. Lett.*, **109**: pp. 155–160, 2008.
- [50] H. Imai and T. Asano. Finding the connected components and a maximum clique of an intersection graph of rectangles in the plane. *J. Algorithms*, **4**: pp. 310–323, 1983.
- [51] R. E. Jamison and H. M. Mulder. Tolerance intersection graphs on binary trees with constant tolerance 3. *Discrete Math.*, **215**: pp. 115–131, 2000.
- [52] D. S. Johnson. Approximation algorithms for combinatorial problems. *J. Comput. System Sci.*, **9**: pp. 256–278, 1974.
- [53] F. Kammer. Determining the smallest k such that G is k -outerplanar. 15th Annual European Symposium on Algorithms (ESA 07), LNCS 4698, pp. 359–370, 2007.
- [54] F. Kammer and T. Tholey. The complexity of minimum convex coloring. 19th Annual International Symposium on Algorithms and Computation (ISAAC 08), LNCS 5369, pp. 16–27, 2008.
- [55] F. Kammer and T. Tholey. The k -disjoint paths problem on chordal graphs. In Proc. 35 International Workshop on Graph-Theoretic Concepts in Computer Science (WG 2009), LNCS 5911, pp. 190–201, 2009.
- [56] F. Kammer and T. Tholey. Approximate tree decompositions of planar graphs in linear time. Proc. 23th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2012), to appear, 2012.
- [57] F. Kammer, T. Tholey, and H. Voepel. Approximation algorithms for intersection graphs. Tech. Report 2009–6, Institut für Informatik, Universität Augsburg, 2009.

- [58] R. M. Karp. Reducibility among combinatorial problems, in complexity of computer computations. R. E. Miller and J. W. Thatcher, eds., Plenum Press, New York, pp. 85–103, 1972.
- [59] R. M. Karp. On the computational complexity of combinatorial problems. *Networks*, **5**: pp. 45–68, 1975.
- [60] S. Khuller, S. G. Mitchell, and V. V. Vazirani. Processor efficient parallel algorithms for the two disjoint paths problem and for finding a kuratowski homeomorph. *SIAM J. Comput.*, **21**: pp. 486–506, 1992.
- [61] T. Kloks. Treewidth: Computations and approximations. *Lecture Notes in Computer Science*, **842**: pp. 338–350, 1994.
- [62] S. V. Krishnan, C. P. Rangan, and S. Seshadri. A new linear algorithm for the two path problem on chordal graphs. Proc. 8th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS 1988), LNCS Vol. 338, Springer, Berlin, 1988, **338**: pp. 49–66, 1998.
- [63] J. F. Lynch. The equivalence of theorem proving and the interconnection problem. (*ACM SIGDA Newsletter*, **5**: pp. 31–36, 1975.
- [64] E. Malesińska. Graph-theoretical models for frequency assignment problems. Ph.D. Thesis, University of Berlin, 1997.
- [65] M. V. Marathe, H. Breu, H. B. Hunt III, S. S. Ravi, and D. J. Rosenkrantz. Simple heuristics for unit disk graphs. *Networks*, **25**: pp. 59–68, 1995.
- [66] K. Mehlhorn and P. Mutzel. On the embedding phase of the hopcroft and tarjan planarity testing algorithm. *Algorithmica*, **16**: pp. 233–242, 1996.
- [67] C. Moore and J. M. Robson. Hard tiling problems with simple tiles. *Discrete and Comput. Geom.*, **26**: pp. 573–590, 2001.
- [68] S. Moran and S. Snir. Convex recolorings of strings and trees: definitions, hardness results and algorithms. Workshop on Algorithms and Data Structures (WADS '05), LNCS, Vol. 3608, Springer, Berlin, 2005, **3608**: pp. 218–232, 2005.
- [69] S. Moran and S. Snir. Efficient approximation of convex recoloring. *J. Comput. System Sci.*, **73**: pp. 1078–1089, 2007.
- [70] T. Nieberg, J. Hurink, and W. Kern. Approximation schemes for wireless networks. *ACM Transactions on Algorithms*, **4**: p. Article No. 49, 2008.
- [71] T. Ohtsuki. The two disjoint path problem and wire routing design. Proc. Symposium on Graph Theory and Algorithms, LNCS, Vol. 108, Springer, Berlin, 1981, **108**: pp. 207–216, 1981.
- [72] Y. Okamoto, T. Uno, and R. Uehara. Linear-time counting algorithms for independent sets in chordal graphs. Proc. 31st International Workshop on Graph-Theoretic Concepts in Computer Science (WG 2005), LNCS Vol. 3787, Springer, Berlin, pp. 433–444, 2005.

- [73] C. H. Papadimitriou and M. Yannakakis. Optimization, approximation, and complexity classes. *J. Comput. System Sci.*, **43**: pp. 425–440, 1991.
- [74] L. Perković and B. Reed. An improved algorithm for finding tree decompositions of small width. *International Journal of Foundations of Computer Science (IJFCS)*, **11**: pp. 365–372, 2000.
- [75] Y. Perl and Y. Shiloach. Finding two disjoint paths between two pairs of vertices in a graph. *J. ACM*, **25**: pp. 1–9, 1978.
- [76] B. Reed. Finding approximate separators and computing tree-width quickly. Proc. 24th Annual ACM Symposium on Theory of Computing (STOC 1992), pp. 221–228, 1992.
- [77] N. Robertson and P. D. Seymour. Graph minors III. planar tree-width. *J. Combin. Theory Ser. B*, **36**(1): pp. 49–64, 1984.
- [78] N. Robertson and P. D. Seymour. Graph minors X. obstructions to tree-decomposition. *J. Combin. Theory Ser. B*, **52**(2): pp. 153–190, 1991.
- [79] N. Robertson and P. D. Seymour. Graph minors. XIII. the disjoint paths problem. *J. Combin. Theory Ser. B*, **63**: pp. 65–110, 1995.
- [80] H. Röhrig. Tree decomposition: A feasibility study. Master’s Thesis, Max-Planck-Institut für Informatik in Saarbrücken, 1998.
- [81] P. D. Seymour. Disjoint paths in graphs. *Discrete Math.*, **29**: pp. 293–309, 1980.
- [82] P. D. Seymour and R. Thomas. Graph searching and a min-max theorem for tree-width. *J. Comb. Theory, Ser. B*, **58**: pp. 22–23, 1993.
- [83] P. D. Seymour and R. Thomas. Call routing and the ratcatcher. *Combinatorica*, **14**(2): pp. 217–241, 1994.
- [84] Y. Shiloach. A polynomial solution to the undirected two paths problem. *J. ACM*, **27**: pp. 445–456, 1980.
- [85] S. Snir. Computational issues in phylogenetic reconstruction: Analytic maximum likelihood solutions, and convex recoloring. Ph.D. Thesis, Department of Computer Science, Technion, Haifa, Israel, 2004.
- [86] E. Tardos. A strongly polynomial algorithm to solve combinatorial linear programs. *Operations Research*, **34**: pp. 250–256, 1986.
- [87] R. E. Tarjan. Depth-first search and linear algorithms. *SIAM J. Comp.*, **1**: pp. 146–160, 1972.
- [88] J. Thatcher and J.B.Wright. Generalised finite automata theory with applications to decision problems of second-order logic. *Mathematical Systems Theory*, pp. 57–81, 1985.
- [89] T. Tholey. Improved algorithms for the 2-vertex disjoint paths problem. Proc. 35th Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM 2009), LNCS Vol. 5404, Springer, Berlin, 2009, **5404**: pp. 546–557, 2009.

- [90] C. Thomassen. 2-linked graphs. *European J. Combin.*, **1**: pp. 371–378, 1980.
- [91] R. E. T. und M. Yannakakis. Simple linear-time algorithms to test chordality of graphs, test acyclicity of hypergraphs, and selectively reduce acyclic hypergraphs. *SIAM J. Comput.*, **13**: pp. 566–579, 1984.
- [92] J. R. Walter. Representations of rigid cycle graphs. Ph.D. Thesis, Wayne State University, 1972.
- [93] H. Whitney. Non-separable and planar graphs. *Trans. Amer. Math. Soc.*, **34**: pp. 339–362, 1932.
- [94] G. Woeginger. A simple solution to the two paths problem in planar graphs. *Inform. Process. Lett.*, **36**: pp. 191–192, 1990.
- [95] Y. Ye and A. Borodin. Elimination graphs. In Proc. of the 36th International Colloquium on Automata, Languages and Programming (ICALP 05), LNCS 5555, pp. 774–785, 2009.

Curriculum Vitae

FRANK KAMMER

Personal Data

Day of birth 10.05.1977
Place of birth Friedberg (Hessen)
Nationality German
Family status Married

Education

1987 – 1996 **Secondary schooling**, Weidiggymnasium, Butzbach.
1997 – 2002 **Studies**, J. W. Goethe-Universität, Frankfurt.
1999: ‘Vordiplom’ in Mathematics
1999: ‘Vordiplom’ in Computer Science
2002: ‘Diplom’ (master’s degree) in Computer Science
2003 – 2010 **Ph.D. Research**, Universität Augsburg, Augsburg,
Faculty of Applied Computer Science.
Supervisor: Prof. Dr. Torben Hagerup