Approximate Tree Decompositions of Planar Graphs in Linear Time

Frank Kammer* and Torsten Tholey*

Institut für Informatik, Universität Augsburg, Germany

Abstract

Many algorithms have been developed for NP-hard problems on graphs with small treewidth k. For example, all problems that are expressable in linear extended monadic second order can be solved in linear time on graphs of bounded treewidth. It turns out that the bottleneck of many algorithms for NP-hard problems is the computation of a tree decomposition of width $O(k)$. In particular, by the bidimensional theory, there are many linear extended monadic second order problems that can be solved on n-vertex planar graphs with treewidth k in a time linear in n and subexponential in k if a tree decomposition of width $O(k)$ can be found in such a time.

We present the first algorithm that, on n-vertex planar graphs with treewidth k, finds a tree decomposition of width $O(k)$ in such a time. In more detail, our algorithm has a running time of $O(nk^2 \log k)$. We show the result as a special case of a result concerning so-called weighted treewidth of weighted graphs.

Keywords: planar graph, (weighted) treewidth, linear time, bidimensionality, branchwidth, rank-width.
ACM classification: F.2.2; G.2.2

1 Introduction

The treewidth, extensively studied by Robertson and Seymour [22], is one of the basic parameters in graph theory. Intuitively, the treewidth measures the similarity of a graph to a tree by means of a so-called tree decomposition. A tree decomposition of width r—defined precisely in the beginning of Section 2—is a decomposition of a graph G into small subgraphs part of a so-called bag such that each bag contains at most $r+1$ vertices and such that the bags are connected by a tree-like structure. The treewidth $\text{tw}(G)$ of a graph G is the smallest r for which G has a tree decomposition of width r.

Often, NP-hard problems are solved on graphs G with small treewidth by the following two steps: First, compute a tree decomposition for G of width $r \in \mathbb{N}$ and second, solve the problem by using this tree decomposition. Unfortunately, there is a trade-off between the running times of these two steps depending on our choice of r. Very often the first step is the bottleneck. For example, Arnborg, Lagergren and Seese [3] showed that, for all problems expressible in so-called linear extended monadic second order (linear EMSO), the second step

*E-mail address: {kammer,tholey}@informatik.uni-augsburg.de

runs on n-vertex graphs in a time linear in n. Demaine, Fomin, Hajiaghayi, and Thilikos [8] have shown that, for many so-called bidimensional problems that are also linear EMSO problems, one can find a solution of a size ℓ in a given n-vertex graph—if one exists—in a time linear in n and subexponential in ℓ as follows: First, try to find a tree decomposition for G of width $r = \hat{c}\sqrt{\ell}$ for some constant $\hat{c} > 0$. One can choose $\hat{c}$ such that, if the algorithm fails, then there is no solution of size at most ℓ. Otherwise, in a second step, use the tree decomposition obtained to solve the problem by well known algorithms in $O(nc^r) = O(nc^{\hat{c}\sqrt{\ell}})$ time for some constant c. Thus, it is very important to support the first step also in such a time. This, even for planar graphs, was not possible by previously known algorithms.

Recent results. In the following overview over related results, n denotes the number of vertices and k the treewidth of the graph under consideration. Tree decomposition and treewidth were introduced by Robertson and Seymour [22], which also presented the first algorithm for the computation of the treewidth and a tree decomposition with a running time polynomial in n and exponential in k [23]. There are numerous papers with improved running times, as, e.g., [2, 7, 19, 25]. Here we focus on algorithms with running times either being polynomial in both k and n, or being subquadratic in n. Bodlaender [6] has shown that a tree decomposition can be found in a time linear in n and exponential in k. However, the running time of Bodlaender's algorithm is practically infeasible already for very small k. The algorithm achieving the so far smallest approximation ratio of the treewidth among the algorithms with a running time polynomial in n and k is the algorithm of Feige, Hajiaghayi, and Lee [10]. It constructs a tree decomposition of width $O(k\sqrt{\log k})$ thereby improving the bound $O(k \log k)$ of Amir [1]. In particular, no algorithms with constant approximation ratios are known that are polynomial in the number of vertices and in the treewidth. One of the so far most efficient practical algorithms with constant approximation ratio was presented by Reed in 1992 [21]. His algorithm computes a tree decomposition of width $3k + 2$ in $O(f(k) \cdot n \log n)$ time for some exponential function f. More precisely, this width is obtained after slight modifications as observed by Bodlaender [5].

Better algorithms are known for the special case of planar graphs. Seymour and Thomas [26] showed that the so-called branchwidth $\mathrm{bw}(G)$ and a so-called branch decomposition of width $\mathrm{bw}(G)$ for a planar graph G can be computed in $O(n^2)$ and $O(n^4)$ time, respectively. A minimum branch decomposition of a graph G can be used directly—like a tree decomposition—to support efficient algorithms. For each graph G, its branchwidth $\mathrm{bw}(G)$ is closely related to its treewidth $\mathrm{tw}(G)$; in detail, $\mathrm{bw}(G) \leq \mathrm{tw}(G) + 1 \leq \max(3/2\,\mathrm{bw}(G), 2)$ [24]. Gu and Tamaki [12] improved the running time to $O(n^3)$ for constructing a branch decomposition and thus for finding a tree decomposition of width $O(\mathrm{tw}(G))$. They also showed that one can compute a tree decomposition of width $(1.5 + c)\mathrm{tw}(G)$ for a planar graph G in $O(n^{(c+1)/c} \log n)$ time for each $c \geq 1$ [13]. Recently, Gu and Xu [14] presented an algorithm to compute a constant factor approximation of the treewidth in time $O(n \log^4 n \log k)$. It is open whether deciding $tw(G) \leq k$ is NP-complete or polynomial time solvable for planar graphs G.

Our results. In this paper, a weighted graph (G, c) is a graph $G = (V, E)$ with a weight function $c : V \to I\!N$ ($I\!N = \{1, 2, 3, \ldots\}$). Moreover, $c_{\max}$ always denotes the maximum weight over all vertices. In contrast to our conference version [17], we now consider the problem of finding a tree decomposition on weighted planar graphs. Roughly speaking, the weighted treewidth of a weighted graph is defined analogously as the (unweighted) treewidth, but instead of counting the vertices of a bag, we sum up the weight of the vertices in the bag. All results shown in this paper for weighted graphs and weighted treewidth can be applied to unweighted graphs and unweighted treewidth by setting $c(v) = 1$ for all vertices v of G. In Section 6, we compute tree decompositions for ℓ-outerplanar graphs with an algorithm different to Bodlaender's algorithm [4], but we obtain the same time bound and the same treewidth. Our algorithm reduces ℓ-outerplanar graphs to weighted ℓ-outerplanar graphs, where the unweighted version is 1-outerplanar. Another application of weighted treewidth is that it allows us to triangulate graphs with only a little increase in the weighted treewidth, which improves our approximation ratio by about a factor of 4.

Interestingly, the generalization from unweighted to weighted graphs is possible without increasing the asymptotic running time. Moreover, we slightly modify our algorithm in such a way that we can afterwards bound the number of so-called $(\mathcal{S}, \varphi)$-components, which improves the running time by a factor k compared to the running time shown in our conference version.

Given a weighted planar graph (G, c) with n vertices and weighted treewidth k, our algorithm computes a tree decomposition for G in time $O(n \cdot k^2 \min\{k, c_{\max}\} \log k)$ such that the vertices in each bag have a total weight of $O(k)$. This means that, for unweighted graphs with n vertices and treewidth k, we obtain a tree decomposition of width $O(k)$ in $O(n \cdot k^2 \log k)$ time, which is a better running time than Gu and Xu [14] for all n-vertex planar graphs of treewidth $k = O(\log n)$. Graphs with a larger treewidth are usually out of interest since for such graphs it is not clear whether we can efficiently solve the second step of our two steps defined for solving NP-hard problems.

Our result can be used to find a solution of size $\ell \in \mathbb{N}$ for many bidimensional graph problems on planar graphs that are expressible in linear EMSO in a time linear in n and subexponential in ℓ. Such problems are, e.g., Minimum Dominating Set, Minimum Maximal Matching, and Minimum Vertex Cover, which all are NP-hard on planar graphs.

In contrast to general graphs, on planar graphs many graph parameters as branchwidth and rank-width differ only by a constant factor from the treewidth [24, 11, 20]. Thus, our algorithm also computes a constant factor approximation for these parameters on n-vertex planar graphs in a time linear in n.

2 Main Ideas

Before we can describe our ideas, we precisely define tree decompositions and (weighted) treewidth.

Definition 2.1 (tree decomposition, bag, width, weight of a bag)**.** *A* tree decomposition *for an unweighted graph $G = (V, E)$ or for a weighted graph $G = (V, E)$ with a weight function $c : V \to \mathbb{N}$ is a pair (T, B), where $T = (W, F)$ is a tree and B is a function that maps each node w of T to a subset of V—called the* bag *of w—such that*
1.) each $v \in V$ is contained in a bag and each $e \in E$ is a subset of a bag, and
2.) for each vertex $v \in V$, the nodes whose bags contain v induce a subtree of T.

In addition, the unweighted width *of* (T, B)*—or short, the* width *of* (T, B)*—is* $\max_{w \in W}\{|B(w)| - 1\}$ *and the* weighted width *is* $\max_{w \in W}\{c(B(w)) - 1\}$ *with* $c(B(w)) = \sum_{v \in B(w)} c(v)$*. The term* $c(B(w))$ *is also called the* weight *of the bag of* w*. The* (weighted) treewidth $\mathrm{tw}(G)$ *of a graph* G *is the smallest* k *for which* G *has a tree decomposition of (weighted) width* k*.*

For simplification, on weighted graphs the word treewidth means weighted treewidth. However, in this section and in Section 7 summarizing our main results we often refer explicitly to weighted or unweighted treewidth.

We next describe our ideas for the construction of a tree decomposition of small unweighted treewidth and subsequently generalize it to weighted treewidth. In the case of unweighted plane graphs it is very useful to model vertices as points in a landscape where we assign a *height* to each vertex v. For the time being, the height can be assumed to be the length of a shortest path from v to a vertex incident to the outer face for some given planar embedding φ where the length of a path is the number of its vertices—a more precise definition is given in the next paragraph. In particular, this is of interest for a graph G if we can bound the height of all vertices of G by $O(tw(G))$ since, as part of our computation of a tree decomposition, we split G in some kind similar to cutting a round cake into slices. More exactly, we use paths starting in a vertex v^* of largest height and following vertices with decreasing height until reaching a vertex adjacent to the outer face. Technically, we realize the splitting by putting the vertices of such a path into one bag. However, we find such a tree decomposition of width k only if the height of v^* is at most $k + 1$ since, otherwise, the vertices of such a path can not be all part of one bag. In the case of weighted plane graphs we have the problem that a large weight of a vertex increases the weight of a bag so much that we have to reduce the number of additional vertices that can be put together with this vertex in one bag of a tree decomposition. To translate the weight of a vertex into our landscape model, we consider a vertex not to be a single point with a single height in the landscape, but as a *cliff* leading from a lower height to an upper height. Thus, instead of a single height we assign a *height interval* to each vertex whose length can be considered as the length of the cliff and is the weight of the vertex minus one.

A weighted graph (G, c) with an embedding φ is called a *weighted plane graph* (G, φ, c), which we now consider. We now precisely define the height interval of each vertex v. It is referred to as $h_\varphi(v) = [h^-_\varphi(v), h^+_\varphi(v)]$. This means that one end of the cliff assigned to vertex v has height $h^-_\varphi(v)$ and the other end has height $h^+_\varphi(v)$. We also call $h^-_\varphi(v)$ the *lower height* of v and $h^+_\varphi(v)$ the *upper height* of v. If $i \in h_\varphi(v)$ for some $i \in I\!N$, we also say v is a vertex of *height* i. The set of all vertices incident to the outer face is called the *coast* (of the plane graph). To define the lower and upper heights of the vertices, we initially define a function η with $\eta(v) = c(v)$ for all vertices v. We now use the concept of a so-called peeling consisting of a sequence of peeling steps. A *peeling step* decrements $\eta(v)$ by one for all vertices v that are part of the coast and subsequently removes all vertices with $\eta(v) = 0$. Let us number the peeling steps by $1, 2, 3, \ldots$. After the removal of all vertices, we set $h_\varphi(v) = [i - c(v) + 1, i]$ for all vertices v that are removed in the ith peeling step ($i \in I\!N$). The height interval of a vertex consists exactly of the numbers of the peeling steps the vertex is incident to the outer face including the peeling step that removed the vertex. For an example see also Fig. 1. A weighted graph G is called *weighted ℓ-outerplanar* if there is an embedding φ of G such that all vertices have upper height at most ℓ. In this case, φ is also called *weighted ℓ-outerplanar*.

We also want to remark that, if we assign weight one to all vertices, the definitions of weighted treewidth and of weighted ℓ-outerplanar graphs correspond to usual unweighted treewidth and to usual unweighted ℓ-outerplanar graphs, respectively.

Observe that each vertex of lower height $q \geq 2$ is incident to a face with a vertex of upper height $q-1$. For technical reasons and to simplify our definitions, our observations and our lemmas, in the rest of the paper we usually consider only *almost triangulated graphs*, i.e., plane graphs in which the boundary of each inner face consists of exactly three vertices and edges. As a consequence, each vertex of lower height $q \geq 2$ is adjacent to a vertex of upper height $q-1$. If a weighted plane graph (H, ψ, c) of treewidth $k-1$ and maximal vertex weight $c_{\max}$ is not almost triangulated, we can multiply each weight of a vertex by $x \in I\!N$ to obtain a weighted plane graph (H', ψ, xc) of treewidth $xk-1$. Afterwards, it can be triangulated by simply adding a new vertex of weight 1 into each inner face and by connecting this vertex by edges with all vertices on the boundary of that inner face. Let (H'', ψ'', xc'') be the graph obtained. Theorem 2 in [18] shows that a tree decomposition (T', B') for H' can be turned into a tree decomposition for H'' by adding at most $3k-2$ of the new vertices into each bag of (T, B). Thus, we have a tree decomposition for H'' where the weight of every bag is bounded by $xk+3k-2$. For some $\alpha, \beta \in I\!N$, assume that we can compute a tree decomposition (T'', B'') for H'' of width $\alpha \cdot \mathrm{tw}(H'') + \beta c'_{\max} - 1$ where $c'_{\max}$ is the maximal weight of a vertex in H''. Then the size of the bags of (T'', B'') is bounded by $\alpha(xk + 3k - 2) + \beta c'_{\max}$. Removing the new vertices from (T'', B''), we get a tree decomposition for H'. If we finally take this tree decomposition as a tree decomposition for H, which has the vertices of smaller weight, the weight of every bag is bounded by $\lfloor(\alpha(xk + 3k - 2) + \beta x c_{\max})/x\rfloor = \lfloor \alpha k + \alpha(3k - 2)/x + \beta c_{\max}\rfloor$. This means that we can compute a tree decomposition for H of

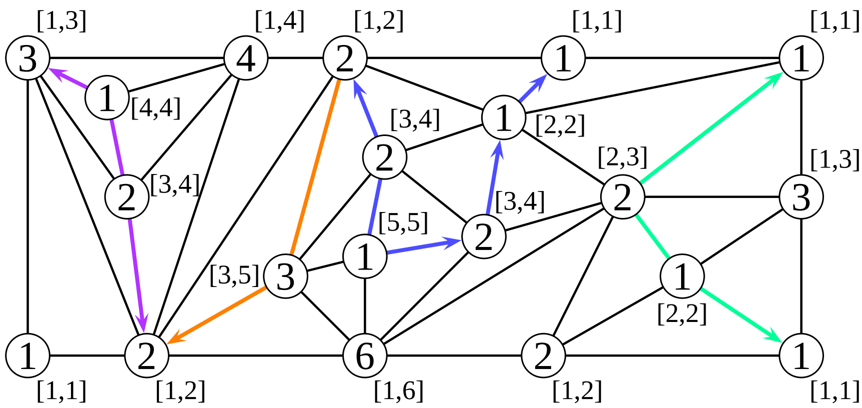

Figure 1: A weighted plane graph G with its vertices labeled by their weights and the resulting height intervals written beside the vertices. Arrows indicate a neighbor with smallest upper height. The thick edges define so-called crest separators defined in Section 3. In our example where G is assumed to be a cake, the crest separators are the paths used to cut the cake into slices.

width $(\alpha+\epsilon)k+\beta c_{\max}+O(1)$ if we choose x large enough.

To describe our ideas, we need some more definitions. For a subgraph G' of a weighted plane graph (G,φ,c), we use $\varphi|_{G'}$ to denote the embedding of G restricted to the vertices and edges of G'. For a graph $G=(V,E)$ and a vertex set $V'\subseteq V$, we let $G[V']$ be the subgraph of G induced by the vertices of V'; and we define $G-V'$ to be the graph $G[V\setminus V']$. If a graph G is a subgraph of another graph G', we write $G\subseteq G'$. Through the whole paper, path and cycles are *simple*, i.e., no vertex and no edge appears more than once in it. For a graph $G=(V,E)$, we also say that a vertex set $S\subseteq V$ *disconnects* two vertex sets $A,B\subseteq V$ *weakly* if no connected component of $G-S$ contains vertices of both A and B. S *disconnects* A and B *strongly* if additionally $S\cap(A\cup B)=\emptyset$ holds. If a vertex set S strongly disconnects two non-empty vertex sets, we say that S is a *separator* (for these vertex sets). We also call a set Y that strongly disconnects a vertex set U from the coast a *coast separator* (for U). As a consequence, the vertices of a coast separator are disjoint from the coast. Finally, we define the *weighted size of a separator* and the *weighted length of a path or cycle* as the sum over the weights of its vertices.

As observed by Bodlaender [4], one can easily construct a tree decomposition of width $3\ell-1$ for an ℓ-outerplanar unweighted graph $G=(V,E)$ in $O(\ell|V|)$ time. In Section 6, we show that his algorithm can be extended to weighted ℓ-outerplanar graphs. One idea to find a tree decomposition of weighted width $O(k)$ for an $\omega(k)$-weighted-outerplanar graph of weighted treewidth k is to search for a coast separator Y of weighted size $O(k)$ that disconnects the vertices of large lower height strongly from the coast by applying Theorem 2.4 below. For proving the theorem we use the following two well-known observations, which follow from the definition of a tree decomposition.

Observation 2.2. *Let (T,B) be a tree decomposition for a weighted graph (G,c), and let W and F be the set of nodes and arcs, respectively, of T. Take $\{w',w''\}\in F$. For each pair of subtrees (W_1,F_1) and (W_2,F_2) part of different trees in the forest $(W,F\setminus\{w',w''\})$, $B(w')\cap B(w'')$ weakly disconnects $\bigcup_{w\in W_1}B(w)$ and $\bigcup_{w\in W_2}B(w)$.*

Observation 2.3. *Let (T,B) and (G,c) be defined as in Obs. 2.2, and let V' be a subset of the vertices of G such that $G[V']$ is connected. Then the nodes of T whose bags contain at least one vertex of V' induce a connected subtree of T.*

Theorem 2.4. *Let (G,φ,c) be a weighted plane graph of weighted treewidth $k\in\mathbb{N}$. Moreover, let V_1 and V_2 be connected sets of vertices of G such that $(\min_{v\in V_2}h^-_\varphi(v))-(\max_{v\in V_1}h^+_\varphi(v))\geq k+1$. Then, there exists a set Y of weighted size at most k that strongly disconnects V_1 and V_2.*

Proof. We exclude the case $k=1$ since it is well known that graphs of treewidth 1 are forests, i.e., $h^-_\varphi(v)=1$ holds for all vertices v. The same is true for graphs of weighted treewidth 1. Consequently, no sets V_1 and V_2 with the properties described in the theorem exist in the case of forests.

Let (T,B) be a tree decomposition of width k with a smallest number of bags containing both at least one vertex of V_1 and at least one vertex of V_2. If there is no such bag, then for each $i\in\{1,2\}$, the nodes of T whose bags contain at least one vertex of V_i induce a subtree of T (Obs. 2.3), and we thus can find two closest nodes w_1 and w_2 in T with $V_1\cap B(w_1)\neq\emptyset\neq V_2\cap B(w_2)$. For the node

w' adjacent to w_2 on the w_1-w_2-connecting path in T, the set $B(w') \cap B(w_2)$ is a separator of weighted size at most k for V_1 and V_2 (Obs. 2.2). Hence, let us assume that there is at least one node w in T with its bag containing at least one vertex of V_1 and of V_2. Since the weight of $B(w)$ is at most $k+1$, for at least one number i with $h^+_\varphi(v_1) < i < h^-_\varphi(v_2)$, there is no vertex in $B(w)$ that has a height interval containing i. In other words, no vertex in $B(w)$ is a vertex of height i. Since V_2 is connected and $\min_{v \in V_2} h^-_\varphi(v)) > i$, there is a connected set Y that consists exclusively of vertices of height i with Y disconnecting V_2 from all vertices u with $h_\varphi(u)^+ < i$, i.e., in particular from V_1. Since Y is connected, the nodes of T, whose bags contain at least one vertex of Y, induce a subtree T' of T (Obs. 2.3). Therefore, it is possible to root T such that w is a child of the root and such that the subtree T_w of T rooted in 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 similarly the edge between the root r of T and the root of T_w by two edges connecting r with the root of T_1 and T_2, respectively. For each node w' in T_w with copies w'_1 and w'_2 in T_1 and T_2, respectively, we define the bag $B(w'_1)$ to consist of those vertices of the bag $B(w')$ that are also contained in the connected component of $G[V \setminus Y]$ containing V_2, and $B(w'_2)$ should contain the remaining vertices of $B(w')$. Since there are no edges between the vertices of the connected component of $G[V \setminus Y]$ containing V_2 and the vertices of other connected components of $G[V \setminus Y]$ and since the bags of T' contain no vertex of Y, for each edge, both of its endpoints still appear in at least one bag after the replacement described above. To sum up, the replacement leads to a tree decomposition of width k with a lower number of bags containing both a vertex of V_1 and of V_2. Contradiction. □

Let us define a maximal connected set H of vertices of the same upper height to be a *crest* if no vertex of H is connected to a vertex of larger upper height. The *(upper) height* of a crest is the upper height of its vertices, and the *lower height* is the minimal lower height among the lower heights of its vertices. Assume that we are given a connected weighted plane graph (G, φ, c) with $G = (V, E)$ and weighted treewidth $k \in \mathbb{N}$ that contains exactly one crest. W.l.o.g., $|V| > 1$. Thus, every vertex is incident to an edge, which implies that $c_{\max} \leq k$, i.e., $c(v) \leq k$ for all vertices v since each vertex must be contained with another vertex of weight at least 1 in a common bag of total weight at most $c(v) + 1$. We can try to construct a tree decomposition for G as follows:

We will construct a series of weighted subgraphs (G', c') of (G, c), where the weight function c' should be implicitly defined by the restriction of c to the vertices of the subgraph. Initialize $G'_1 = (V'_1, E'_1)$ with G. For $i = 1, 2, \ldots$, as long as G'_i has only one crest and has vertices of lower height at least $2k+1$ (which are connected since G'_i has only one crest) apply Theorem 2.4 to obtain a separator Y_i of weighted size at most k separating the vertices of lower height at least $2k+1$ from all vertices of upper height at most k. This means that we separate the vertices of large lower height from all vertices of the coast since a vertex v of the coast can have an upper height of at most $c(v) \leq k$. Thus, Y_i is a coast separator. Then, define $G'_{i+1} = (V'_{i+1}, E'_{i+1})$ as the subgraph of G'_i induced by the vertices of Y_i and of the connected component of $G'_i \setminus Y_i$ that contains the crest of G'_i. Moreover, let $G_i = G'_i[Y_i \cup (V'_i \setminus V'_{i+1})]$. If the recursion stops with a $O(k)$-weighted-outerplanar graph G'_j ($j \in \mathbb{N}$), we set $G_j = G'_j$ and construct a tree decomposition for G as follows: First, compute a tree decomposition

(T_i, B_i) of weighted width $O(k)$ for each G_i ($i \in \{1, \ldots, j\}$). This is possible since G_i is $O(k)$-weighted-outerplanar. Second, set $Y_0 = Y_j = \emptyset$. Then, for all $i \in \{1, \ldots, j\}$, add the vertices of $Y_i \cup Y_{i-1}$ to all bags of (T_i, B_i). Finally, for all $i \in \{1, \ldots, j-1\}$, connect an arbitrary node of T_i with an arbitrary node of T_{i+1}. This leads to a tree decomposition for G of weighted width $O(k)$.

If we are given a weighted plane graph (G, φ, c) with weighted treewidth $k \in I\!N$ that has more than one crest of lower height at least $2k+1$, (or if this is true for one of the subgraphs G'_i defined above) we cannot apply Theorem 2.4 to find one coast separator separating simultaneously all vertices of lower height at least $2k+1$ from the coast since these vertices may not be connected. For cutting off the vertices of large height, one might use several coast separators; one for each connected component induced by the vertices of lower height at least $2k+1$. However, if we insert the vertices of all coast separators into every bag of a tree decomposition for the graph with the vertices of small lower height, this may increase the width of the tree decomposition by more than a constant factor since there may be more than a constant number of coast separators. This is the reason why, for some suitable linear function $f : I\!N \to I\!N$ and some constant $c \in I\!N$, we search for further separators called *perfect crest separators* that are disjoint from the crests and that partition our graph G (or G'_i) into smaller subgraphs—called *components*—such that, for each component C containing a non-empty set V' of vertices of lower height at least $f(k)$, there is a set Y_C of vertices with the following properties:

(P1) Y_C is a coast separator for V' of weighted size $O(k)$.
(P2) Y_C is contained in C.
(P3) Y_C is disjoint to the set of vertices with lower height $\leq c$.

The main idea is that—in some kind similar to the construction above—we want to construct a tree decomposition separately for each component and afterwards to combine these tree decompositions to a tree decomposition of the whole graph. The properties above should guarantee that, for each tree decomposition computed for a component, we have to add the vertices of at most one coast separator into the bags of that tree decomposition. We next try to guarantee (P1)-(P3).

By making the components so small that each component has at most one maximal connected set of vertices of height at least $f(k)$, we can easily find a set of coast separators satisfying (P1) by using Theorem 2.4. We next try to find some constraints for the perfect crest separators such that we can also guarantee (P2). Recall that we assume that our graph is almost triangulated. Thus, the vertices of a coast separator of minimal weighted size induce a unique cycle. Suppose for a moment that it is possible to choose each perfect crest separator as the vertices of a path with the property that, for each pair of consecutive vertices u and v with u before v, the upper height of v is one smaller than the lower height of u, i.e., $h^+_\varphi(v) = h^-_\varphi(u) - 1$. In Section 3, we call such a path a *down path* if it also has some additional properties. For a down path P, there is no path that connects two vertices u and v of P with a strictly shorter weighted length than the subpath of P from u to v. Consequently, whenever we search for a coast separator of smallest weighted size for a connected set V' in a component C, there is no need to consider any coast separator with a subpath Q consisting of vertices that are strongly disconnected from the vertices of C by the vertex set of P. Note that it is still possible that vertices of a coast separator belong to a perfect crest separator. To guarantee that (P2) holds, in a more precise definition of the components, we let the vertices and edges of a crest separator belong to two components on 'both sides' of the crest separator. Then we can observe that, if we choose the perfect crest separators as down paths not containing any vertex of any crest, each maximal connected set of vertices of lower height at least $f(k)$ in a component C has a coast separator Y_C that is completely contained in C.

Unfortunately, the vertex set of one down path can not be a separator. For two down paths P_1 and P_2 that start in two adjacent vertices, we use $P_1 \circ P_2$ to denote the concatenation of the reverse path of P_1, a path P', and the path P_2, where P' is the path induced by the edge connecting the first vertices of P_1 and P_2. The idea is to define a perfect crest separator as the vertex set of such a concatenation $P_1 \circ P_2$. For more information on crest separators, see Section 3. With our new definition of a perfect crest separator we cannot avoid in general that a coast separator Y_C for the crest of a component C crosses a perfect crest separator and uses vertices outside C. Thus (P2) may be violated. To handle this problem we define a *minimal coast separator* for a connected set S in a graph G to be a coast separator Y for S that has minimal weighted size such that among all such coast separators the subgraph of G induced by the vertices of Y and the vertices of the connected component of $G \setminus Y$ containing S has a minimal number of inner faces. Whenever a minimal coast separator for a connected set S in a component C crosses a crest separator, the part of the minimal coast separator outside C forms a so-called pseudo shortcut. Further details on pseudo shortcuts and their computations are described in Section 4. Lemma 5.2 shows that, if a pseudo shortcut P is part of a minimal coast separator Y_C for a connected set of vertices in a component C and if it passes through another component C', Y_C also separates all vertices of lower height at least $2k+1$ in C' from the coast. Then, we merge C and C' to one super component C^*. After a similar merging for each pseudo shortcut passing through another component, both properties (P1) and (P2) hold. For guaranteeing property (P3), vertices of height $\leq c$ and, in particular, the vertices of the coast play a special role for several definitions, e.g., for the pseudo shortcuts or the so-called lowpoints of a crest separator.

Given a perfect crest separator $X = P_1 \circ P_2$, the idea is to find tree decompositions (T_1, B_1) and (T_2, B_2) for the two components of G on 'either sides' of X such that T_i, for each $i \in \{1, 2\}$, has a node w_i with $B_i(w_i)$ containing all vertices of P_1 and P_2. By inserting an additional edge $\{w_1, w_2\}$ we then obtain a tree decomposition for the whole graph. However, in general we are given a set $\mathcal{X}$ of perfect crest separators that splits our graph into components for which (P1)-(P3) holds. If we want to construct a tree decomposition for a component, we usually have to guarantee that, for each perfect crest separator $X \in \mathcal{X}$, there is a bag containing all vertices of X. Since we can use the techniques described above to cut off the vertices of large height from each component, it remains to find such a tree decomposition for the remaining $O(k)$-weighted-outerplanar subgraph of the component. Because of the simple structure of our perfect crest separators we can indeed find such a tree decomposition by extending Bodlaender's algorithm [4] for $O(k)$-outerplanar graphs. For more details see Section 6. Finally, we can iteratively connect the tree decompositions constructed for the several components in the same way as it is described in case of only one perfect crest separator. For an example, see also Fig. 2 and 3—the concepts of 'top vertex' and 'ridge' are defined in the next section. Our algorithm to compute a tree decomposition is presented in Section 7.

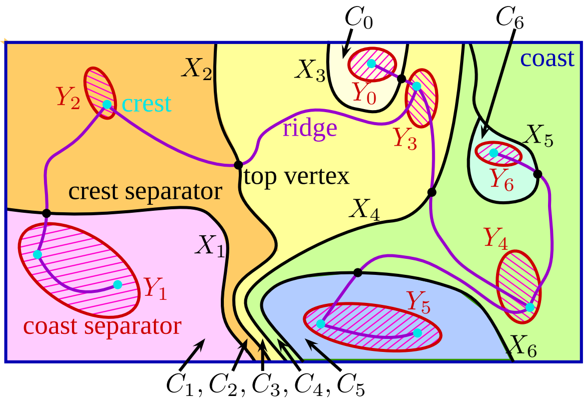

Figure 2: A weighted plane graph (G, φ, c) decomposed by a set $\mathcal{S} = \{X_1, \ldots, X_6\}$ of crest separators into several components $C_0, \ldots, C_6$. In addition, each component C_i has a coast separator Y_i for all crests in C_i.

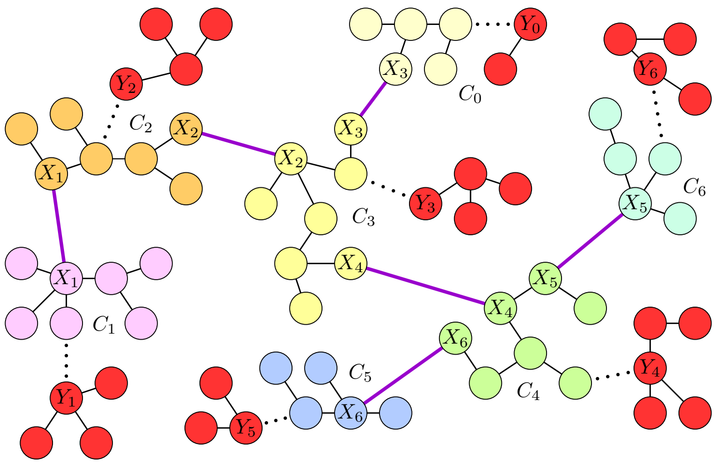

Figure 3: A sketch of a tree decomposition for the graph G of Fig 2. In detail, each colored component C_i has its own tree decomposition (T_i, B_i) for the vertices of small height in C_i with the bags of (T_i, B_i) being colored with the same color than the component C_i. We implicitly assume that Y_i is part of all bags of (T_i, B_i). A tree decomposition for the vertices of large height in C_i is constructed recursively and contains a bag—marked with Y_i—containing all vertices Y_i and being connected to one bag of (T_i, B_i). For connecting the tree decompositions of different components, there is an edge connecting a node w' of (T_i, B_i) with a node w'' of (T_j, B_j) if and only if C_i and C_j share a certain edge (on their boundary) that later is defined more precisely. In this case, the bags w' and w'' contain the vertices of the crest separators disconnecting C_i and C_j—and are marked by the name of this crest separator.

3 Decomposition into Mountains

In the next three sections we let (G, φ, c) be a weighted, almost triangulated, and biconnected graph with vertex set V and edge set E. If a weighted graph is not biconnected one can easily construct a tree decomposition for the whole graph by combining tree decompositions for each biconnected subgraph. Moreover, let n be the number of vertices of G and take ℓ as the smallest number such that φ is weighted ℓ-outerplanar. Recall that, for each vertex v, $h_\varphi(v) = [h^-_\varphi(v), h^+_\varphi(v)]$ is the height interval of v with $h^-_\varphi(v)$ and $h^+_\varphi(v)$ being called the lower and upper height of v. Recall also that a crest is a maximal connected set H of vertices of the same upper height such that no vertex in H is connected to a vertex of larger upper height. A weighted plane graph with exactly one crest is called a *mountain*. In this section we show a splitting of (G, φ, c) into several mountains. Since 'certain' crests are not of interest, we also show that, given a set of crests, a splitting of (G, φ, c) into several so-called components is possible such that each component contains one crest part of the set.

As indicated in Section 2 our splitting process makes use of so-called perfect crest separators and down paths. Since it is not so easy to compute perfect crest separators, we start to define and to consider crest separators more general and later describe how to find perfect crest separators as a subset of the crest separators. First of all, we have to define down paths precisely. Let us assume w.l.o.g. that the vertices of all graphs considered in this paper are numbered with pairwise different integers called the *vertex number*. For each vertex u with a lower height $q \geq 2$, we define the *down vertex* of u to be the neighbor of u that among all neighbors of u with upper height $q - 1$ has the smallest vertex number. We denote the down vertex of u by $u\!\downarrow$. The down edge of u is the edge $\{u, u\!\downarrow\}$. The *down path* (of a vertex v) is a path that (starts in v,) consists completely of down edges, and ends in a vertex of the coast. In particular, a vertex v of lower height 1 is a down path that only consists of itself. Note that every vertex has a down path.

Definition 3.1 (crest separator, top edge, exterior/interior lowpoint)**.** *A* crest separator *in a weighted, almost triangulated, and biconnected graph is a tuple $X = (P_1, P_2)$ with P_1 being a down path starting in some vertex u and P_2 being a down path starting in a neighbor $v \neq u\!\downarrow$ of u with $h^+_\varphi(v) \leq h^+_\varphi(u)$, where in the case $h^+_\varphi(v) = h^+_\varphi(u)$ the vertex number of v is smaller than that of u. The edge $\{u, v\}$ is called* top edge *of X. The first vertex v on P_1 that also is part of P_2, if it exists, is called the* lowpoint *of X. If v belongs to the coast, we call v an* exterior lowpoint*, otherwise an* interior lowpoint.

In the remainder of this paper, we let $\mathcal{S}(G, \varphi, c)$ be the set of all crest separators in (G, φ, c). Note that, for a crest separator $X = (P_1, P_2)$, the vertex set of $P_1 \circ P_2$ usually defines a separator. This explains the name crest separator, but formally a crest separator is a tuple of paths. Note also that a top edge is never a down edge and uniquely defines a crest separator. Moreover, the top edges of two different crest separators are always different. Since an n-vertex planar graph has at most $O(n)$ edges, which can possibly be a top edge, and since, in a weighted ℓ-outerplanar graph, each crest separator consists of at most 2ℓ vertices, the next lemma holds.

Lemma 3.2. *The set $\mathcal{S}(G, \varphi, c)$ can be constructed in $O(\ell n)$ time.*

Since crest separators are in the main focus of our paper, we use some additional terminology: Let $X = (P_1, P_2)$ be a crest separator. Then, the *top vertices* of X consist of the first vertex of P_1 and the first vertex of P_2. A top vertex of X is called *highest* if its upper height is at least as large as the upper height of the other top vertex of X. We write $v \in X$ and say that v *is a vertex of* X to denote the fact that v is a vertex of P_1 or P_2. The *border edges* of X are the edges of $P_1 \circ P_2$. The *height* of X is the maximum upper height over all its vertices, which is the upper height of the first vertex of P_1. The *weighted length* of X is the weighted length of $P_1 \circ P_2$. The *essential boundary of* X is the subgraph of G induced by all border edges of X that appear on exactly one of the two paths P_1 and P_2. In particular, if X has a lowpoint, the vertices of the essential boundary consists exactly of the vertices appearing before the lowpoint on P_1 or P_2 and of the lowpoint itself. If X has no lowpoint, the essential boundary is the subgraph of G induced by the edges of $P_1 \circ P_2$. For two vertices s_1 and s_2 being part of the essential boundary of X, the *(long and short) crest-separator path* from s_1 to s_2 is the longest and shortest path, respectively, from s_1 to s_2 that consists only of border edges of X and that does not contain the lowpoint of X as an inner vertex. Note that, if neither s_1 nor s_2 is the lowpoint of X, the longest and the shortest crest-separator path of X are the same. If s_1 and s_2 are vertices of $P_1 \circ P_2$, but not both are part of the essential boundary of X, the *crest-separator path* from s_1 to s_2 is the shortest path from s_1 to s_2 consisting completely of edges of $P_1 \circ P_2$.

We say that two paths P' and P'' *cross*, if after merging the endpoints of the common edges of P' and P'', there is a vertex v with incident edges e_1 of P', e_2 of P'', e_3 of P', e_4 of P'' appearing clockwise in this order around v. The vertices that are merged into v are called the *crossing vertices of* P' and P''. Moreover, we also say that P and P' cross if adding a new endpoint v' to the outer face as well as an edge e from v' to one endpoint of either P or P' with lower height 1 together with an appropriate planar embedding of e in φ makes the resulting paths cross with respect to the definition of the previous sentence. A crest separator $X = (P_1, P_2)$ and a path P cross if $P_1 \circ P_2$ and P cross.

Each set $\mathcal{S}$ of crest separators splits G into several subgraphs. More precisely, for a set $\mathcal{S} \subseteq \mathcal{S}(G, \varphi, c)$, let us define two inner faces F and F' of (G, φ, c) to be $(\mathcal{S}, \varphi)$*-connected* if there is a list $(F_1, \ldots, F_j)$ $(j \in I\!N)$ of inner faces of (G, φ, c) with $F_1 = F$ and $F_j = F'$ such that, for each $i \in \{1, \ldots, j-1\}$, the faces F_i and F_{i+1} share a common edge not being a border edge of a crest separator in $\mathcal{S}$. A set $\mathcal{F}$ of inner faces of (G, φ, c) is $(\mathcal{S}, \varphi)$*-connected* if each pair of faces in $\mathcal{F}$ is $(\mathcal{S}, \varphi)$-connected. Hence, a graph is splitted by crest separators into the following kind of subgraphs.

Definition 3.3 (($\mathcal{S}, \varphi$)-component). *Let $\mathcal{S} \subseteq \mathcal{S}(G, \varphi, c)$. For a maximal non-empty $(\mathcal{S}, \varphi)$-connected set $\mathcal{F}$ of inner faces of (G, φ, c), the subgraph of G consisting of the set of vertices and edges that are part of the boundary of at least one face $F \in \mathcal{F}$ is called an* $(\mathcal{S}, \varphi)$-component.

By the fact that an $(\mathcal{S}, \varphi)$-component consists of the vertices on the boundary of an $(\mathcal{S}, \varphi)$-connected set of faces, we can observe.

Observation 3.4. *The $(\mathcal{S}, \varphi)$-component of a biconnected graph is biconnected.*

For a single crest separator X in (G, φ, c), the set $\{X\}$ splits (G, φ, c) into exactly two $(\{X\}, \varphi)$-components, which, for an easier notation, are also called

(X,φ)-*components*. For the (X,φ)-components D and $\tilde{D}$, we say that $\tilde{D}$ is *opposite* to D. We say that X *goes weakly between two vertex sets* U_1 and U_2 if we can number the two (X,φ)-components with $C_1=(V_1,E_1)$ and $C_2=(V_2,E_2)$ such that $U_1\subseteq V_1$ and $U_2\subseteq V_2$. If additionally $U_1\cup U_2$ does not contain any vertex of X, we say that X *goes strongly between* the sets. We also say that X goes strongly (or weakly) between two subgraphs if X goes strongly (or weakly) between their corresponding vertex sets. We want to remark that these definitions focus on the disconnection of faces instead of vertex sets. Nevertheless, if a crest separator X weakly (strongly) goes between two non-empty vertex sets A and B, then the set of vertices of X weakly (strongly) disconnects A and B.

Recall that our goal is to find a subset of the set of all crest separators large enough to separate all crests from each other. As a first step to restrict the set of crest separators, we define a special kind of path, a so-called ridge, at the end of this paragraph. The only crest separators that we need are those that start at an inner vertex of a ridge. Moreover, we define the ridge in such a way that each inner vertex of the ridge defines a crest separator. We start with the definition of a *height-vector* of a path P. This is a vector $(n_1,\ldots,n_\ell)$ where n_i ($i\in\{1,\ldots,\ell\}$) is the number of vertices of P whose upper height is i. We say that a height vector $(n_1,...,n_\ell)$ is smaller than a height vector $(n'_1,...,n'_\ell)$ if it is smaller with respect to the lexicographical order. For vertices s and t, a *ridge* R between s and t is a path connecting s and t with a smallest height-vector among all paths connecting s and t. A vertex of R with smallest upper height h is called a *deepest vertex* of R, and h is called the *depth* of R.

Lemma 3.5. *For every inner vertex u of a ridge R, there is a neighbor v of u of at most the same upper height such that the down path P_1 of u and the down path P_2 of v define a crest separator $X=(P_1,P_2)$ or $X=(P_2,P_1)$ that crosses R.*

Proof. Consider the down path P_1 of u. Let $\bar{u}\neq u$ be the other endpoint of P_1. W.l.o.g., P_1 has an edge $\{u',\bar{u}\}$ that is not part of R. Otherwise, extend P_1 by an edge $\{\bar{u},u^*\}$ to a new virtual vertex u^* in the outer face. P_1 can be extended by an edge $\{u,v'\}$ such that the resulting path P'_1 crosses the ridge. Intuitively speaking, v' is a neighbor of u 'on the other side of the ridge' than P_1. Note that there must be indeed at least one vertex v' on the other side of the ridge since, otherwise, u must be a vertex of the coast, the downpath of u has only u as vertex, and u is incident on both sides to the outer face, which is a contradiction to the biconnectivity of G.

Let L be the cyclic list of neighbors of u in clockwise order, and let r_1 and r_2 be the two vertices of L that belong to R. We split $L\setminus\{r_1,r_2\}$ into two sublists L_1 and L_2 where L_i ($i\in\{1,2\}$) starts with the successor of r_i and ends with the predecessor of r_{3-i}. Note that, for all vertices v'' of the list L_j containing v', the concatenation of v'' and P_1 crosses R. If such a vertex v'' exists with $h^+_\varphi(v'')\leq h^+_\varphi(u)$, then we can take $v=v''$ and the lemma holds. Let us assume that no such vertex v'' exists. Let R' be the path obtained from R where u is replaced by the vertices in L_j. Then R' has a smaller height-vector than R since R' has one vertex less of height $h^+_\varphi(u)$ whereas the number of vertices with a smaller upper height does not change; a contradiction to the fact that R is a ridge. □

Let $\mathcal{H}$ be a set of crests of (G,φ,c). A *mountain structure* for (G,φ,c) and $\mathcal{H}$ is a tuple $(G,\varphi,c,\mathcal{H},\mathcal{S})$ with $\mathcal{S}\subseteq\mathcal{S}(G,\varphi,c)$ such that, for each pair of different crests H_1 and H_2 in $\mathcal{H}$ and for each ridge R in (G, φ, c) with one endpoint in H_1 and the other in H_2, the following property holds:

(a) There is a crest separator $X \in \mathcal{S}$ with one of its highest top vertices being a vertex of R such that X strongly goes between H_1 and H_2 and such that X has smallest weighted length among all such crest separators in $\mathcal{S}(G, \varphi, c)$.

The next lemma shows that the simple structure of a crest separator as a tuple of two down paths suffices to separate each pair of crests of the graph; in particular, property (a) can be easily satisfied for each set $\mathcal{H}$ of crests by setting $\mathcal{S} = \mathcal{S}(G, \varphi, c)$.

Lemma 3.6. *Let R be an ridge between two vertices s and t in (G, φ, c). Choose v as a deepest inner vertex of R. If the depth of R is smaller than the upper height of both s and t, there is a crest separator in $\mathcal{S}(G, \varphi, c)$ that*

- *goes strongly between $\{s\}$ and $\{t\}$, and*
- *contains v as a highest top vertex.*

Proof. By Lemma 3.5, there is a crest separator $X = (P_1, P_2)$ with top vertex v that crosses R. Since v is a deepest vertex of R and since P_1 and P_2 are down path whose highest vertices have upper height $\leq h^+_\varphi(v)$, R and X can cross only once. Note that this also implies that X neither contains s nor t. Thus, X goes strongly, between s and t. □

As a consequence of the last lemma, the tuple $(G, \varphi, c, \mathcal{H}, \mathcal{S}(G, \varphi, c))$ is a mountain structure for (G, φ, c) and $\mathcal{H}$. It appears that some crest separators of a mountain structure may be useless since they split one crest into several crests or they cut of parts of our original graph not containing any crests. Hence, we define a mountain structure $(G, \varphi, c, \mathcal{H}, \mathcal{S})$ to be *good* if, in addition to property (a), also the following properties hold:

(b) No crest separator in $\mathcal{S}$ contains a vertex of a crest in $\mathcal{H}$.

(c) Each $(\mathcal{S}, \varphi)$-component contains vertices of a crest in $\mathcal{H}$.

Let $(G, \varphi, c, \mathcal{H}, \mathcal{S})$ be a mountain structure. Note that the properties (a) and (c) imply that each $(\mathcal{S}, \varphi)$-component contains the vertices of exactly one crest in $\mathcal{H}$, and property (b) guarantees that the crest is completely contained in one $(\mathcal{S}, \varphi)$-component. For each ridge R of a pair of crests H_1 and H_2 in $\mathcal{H}$, the set $\mathcal{S}$ of crest separators contains a crest separator that strongly goes between H_1 and H_2 and that has a highest top vertex with upper height equal to the depth of R (Lemma 3.6 in combination with property (a)).

Corollary 3.7. *For each pair of crests H_1 and H_2 in $\mathcal{H}$, there is a crest separator in $\mathcal{S}$ that strongly goes between H_1 and H_2 and that has upper height equal to the depth of a ridge connecting H_1 and H_2.*

We next want to show that a good mountain structure exists and can be computed efficiently. For that let us define a *mountain connection tree* T of a mountain structure $(G, \varphi, c, \mathcal{H}, \mathcal{S})$ to be a graph defined as follows: Each node of T is identified with an $(\mathcal{S}, \varphi)$-component of G, and two nodes w_1 and w_2 of T are connected if and only if they—or more precisely the $(\mathcal{S}, \varphi)$-components with which they are identified—have a common top edge of a crest separator in $\mathcal{S}$.

Recall that the border edges of a crest separator $X = (P_1, P_2)$ consist of one top edge e of X and further down edges, and that a down edge cannot be a top edge of any crest separator. Since the edges of X are the only edges that are part of both (X, φ)-components, the top edge e is the only top edge of a crest separator that is contained in both (X, φ)-components. Moreover, since two down paths can not cross by definition, for each crest separator $X \in \mathcal{S}$ of a good mountain structure, we can partition the set of all $(\mathcal{S}, \varphi)$-components into a set $\mathcal{C}_1$ of $(\mathcal{S}, \varphi)$-components completely contained in one (X, φ)-component and the set $\mathcal{C}_2$ of $(\mathcal{S}, \varphi)$-components contained in the other (X, φ)-component. Then, X is the only crest separator with a top edge belonging to $(\mathcal{S}, \varphi)$-components in $\mathcal{C}_1$ as well as in $\mathcal{C}_2$. Consequently, T is indeed a tree.

Lemma 3.8. *The mountain connection tree of a mountain structure is a tree.*

The fact that the top edge of a crest separator X is the only top edge belonging to both (X, φ)-components shows also the correctness of the next lemma.

Lemma 3.9. *Let C_1 and C_2 be two $(\mathcal{S}, \varphi)$-components that are neighbors in the mountain connection tree of a mountain structure $(G, \varphi, c, \mathcal{H}, \mathcal{S})$. Then there is exactly one crest separator in $\mathcal{S}$ going weakly between C_1 and C_2, which is also the only crest separator with a top edge belonging to both C_1 and C_2.*

Since each of the $O(n)$ crest separators consists of at most $O(\ell)$ edges, we can determine in $O(\ell n)$ time all $(\mathcal{S}, \varphi)$-components and afterwards construct the mountain connection tree by a simple breadth-first search on the dual graph of (G, φ, c).

Lemma 3.10. *Given a mountain structure $(G, \varphi, c, \mathcal{H}, \mathcal{S})$ for a set $\mathcal{S}$ of crest separators, its mountain connection tree can be determined in $O(\ell n)$ time.*

We also can construct a good mountain structure.

Lemma 3.11. *Given (G, φ, c) and $\mathcal{H}$, a good mountain structure $(G, \varphi, c, \mathcal{H}, \mathcal{S})$ can be constructed in $O(\ell n)$ time.*

Proof. For a simpler notation, in this proof we call a crest separator of a set $\tilde{\mathcal{S}}$ of crest separators to be a *heaviest crest separator* of $\tilde{\mathcal{S}}$ if it has a largest weighted length among all crest separators in $\tilde{\mathcal{S}}$.

We first construct the set $\mathcal{S}' = \mathcal{S}(G, \varphi, c)$ of all crest separators in $O(\ell n)$ time (Lemma 3.2). Lemma 3.6 guarantees that, for each pair of crests in $\mathcal{H}$ and each ridge R with endpoints in both crests, there is a crest separator X in $\mathcal{S}(G, \varphi, c)$ strongly going between the two crests with a highest top vertex of X being a deepest vertex of R. This in particular means that all vertices of X have an upper height smaller than or equal to the depth of the ridge, which is strictly smaller than the upper height of the two crests and of all crests through which R passes, i.e., no vertex of X is part of a crest in G. Hence, in $O(\ell n)$ time, we can remove all crest separators from $\mathcal{S}'$ that contain a vertex of a crest and property (a) of a good mountain structure is maintained. Afterwards property (b) holds. Let $\mathcal{S}''$ be the resulting set of crest separators. For guaranteeing property (c), we have to remove further crest separators from $\mathcal{S}''$. We start with constructing the mountain connection tree T of $(G, \varphi, c, \mathcal{H}, \mathcal{S}'')$ in $O(\ell n)$ time (Lemma 3.10). We root T at an arbitrary node of T.

In a sophisticated bottom-up traversal of T we dynamically update $\mathcal{S}''$ by removing superfluous crest separators in $O(\ell n)$ time. For a better understanding, before we present a detailed description of the algorithm, we roughly sketch some ideas. Our algorithms marks some nodes as finished in such a way that the following invariant (I) always holds: If a node C of T is marked as finished, the $(\mathcal{S}'', \varphi)$-component C contains exactly one crest in $\mathcal{H}$. The idea of the algorithm is to process a so far unfinished node that has only children already marked as finished and that, among all such nodes, has 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 $(\mathcal{S}'', \varphi)$-components identified with w and 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^*. Additionally, we mark w^* as finished only if w' is a child of w since in this case we already know that w' is already marked as finished, i.e., w' contains a crest in $\mathcal{H}$ that is now part of w^*.

We now describe our algorithm in detail. We start with some preprocessing steps. In $O(n)$ time, we determine and store with each node w of T a value $\mathrm{Crest}(w) \in \{0, 1\}$ that is set to 1 if and only if the $(\mathcal{S}'', \varphi)$-component identified with w contains a vertex that is part of a crest in $\mathcal{H}$. In $O(\ell n)$ time, we additionally store for each crest separator in $\mathcal{S}''$ its weighted length, mark each node as unfinished, and store with each non-leaf w of T in a variable $\mathrm{MaxCrestSep}(w)$ a heaviest crest separator of the set of all crest separators going weakly between the $(\mathcal{S}'', \varphi)$-component identified with w and an $(\mathcal{S}'', \varphi)$-component identified with a child of w. For each leaf w of T, we define $\mathrm{MaxCrestSep}(w) = \mathrm{nil}$. As a last step of our preprocessing phase, which also runs in $O(n)$ time, for each node w of T, we initialize a value $\mathrm{MaxCrestSep}^*(w)$ with nil. $\mathrm{MaxCrestSep}^*(w)$ is defined analogously to $\mathrm{MaxCrestSep}(w)$ if we restrict the crest separators to be considered only to those crest separators that go weakly between two $(\mathcal{S}'', \varphi)$-components identified with w and with a finished child of w. We will see that it suffices to know the correct values of $\mathrm{MaxCrestSep}(w)$ and $\mathrm{MaxCrestSep}^*(w)$ only for the unfinished nodes and therefore we do not update these values for finished nodes.

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 current set of remaining crest separators, which is updated dynamically. First we exclude the case, where w has a parent $\tilde{w}$ with an unfinished child $\hat{w}$, and where $\mathrm{MaxCrestSep}(\tilde{w})$ is equal to the crest separator going weakly between the $(\mathcal{S}'', \varphi)$-components identified with w and $\tilde{w}$. More precisely, in this case we delay the processing of w and continue with the processing of $\hat{w}$.

Second, we test whether the $(\mathcal{S}'', \varphi)$-component C identified with w contains a vertex belonging to a crest in $\mathcal{H}$, which is exactly the case if $\mathrm{Crest}(w) = 1$. In this case, we mark w as finished. If there is a parent $\tilde{w}$ of w with X being the crest separator going weakly between the two $(\mathcal{S}'', \varphi)$-components identified with w and to $\tilde{w}$, we replace $\mathrm{MaxCrestSep}^*(\tilde{w})$ by a heaviest crest separator in $\{X, \mathrm{MaxCrestSep}^*(\tilde{w})\}$.

Let us next consider the case where $\mathrm{Crest}(w) = 0$. We then remove a heaviest crest separator X of the set of all crest separators going weakly between the $(\mathcal{S}'', \varphi)$-component identified with w and an $(\mathcal{S}'', \varphi)$-component identified with a neighbor of w, i.e., either with (Case i) a child $\hat{w}$ of w or (Case ii) the parent $\tilde{w}$ of w. X can be taken as either $\mathrm{MaxCrestSep}(w)$ or the crest separator going weakly between the two $(\mathcal{S}'', \varphi)$-components identified with w and $\tilde{w}$.

Case i: We remove X from $\mathcal{S}''$ and mark the node w^* obtained from merging w and $\hat{w}$ as finished and set $\mathrm{Crest}(w^*) = 1$. Note that this is correct since $\hat{w}$ is already marked as finished and therefore $\mathrm{Crest}(\hat{w}) = 1$. In addition, if the parent $\tilde{w}$ of w exists, we replace $\mathrm{MaxCrestSep}^*(\tilde{w})$ by the heaviest crest separator contained in $\{X', \mathrm{MaxCrestSep}^*(\tilde{w})\}$ where X' is the crest separator in $\mathcal{S}''$ going strongly between the $(\mathcal{S}'', \varphi)$-components identified with w and $\tilde{w}$.

Case ii: We mark the node w^* obtained from merging the unfinished nodes w and $\tilde{w}$ as unfinished, set $\mathrm{Crest}(w^*) = \mathrm{Crest}(\tilde{w})$, and define the value $\mathrm{MaxCrestSep}^*(w^*)$ as the heaviest crest separator in $\{\mathrm{MaxCrestSep}^*(w), \mathrm{MaxCrestSep}^*(\tilde{w})\}$ or nil if this set contains no crest separator. If $\tilde{w}$ beside w has another unfinished child, we define $\mathrm{MaxCrestSep}(w^*)$ as the heaviest crest separator in $\{\mathrm{MaxCrestSep}(w), \mathrm{MaxCrestSep}(\tilde{w})\}$. Note that $\mathrm{MaxCrestSep}(\tilde{w}) \neq X$ since otherwise the processing of w would have been delayed. If $\tilde{w}$ has no other unfinished child, we take $\mathrm{MaxCrestSep}(w^*)$ as the heaviest crest separator that is contained in $\{\mathrm{MaxCrestSep}(w), \mathrm{MaxCrestSep}^*(\tilde{w})\}$ or nil if no crest separator is in this set. Note that $\mathrm{MaxCrestSep}^*(\tilde{w}) \neq X$ since w is unfinished before its processing.

We can conclude by induction that our algorithm correctly updates $\mathrm{Crest}(w)$ for all nodes w as well as $\mathrm{MaxCrestSep}(w)$ and $\mathrm{MaxCrestSep}^*(w)$ for all unfinished nodes w. If the processing of a node in T is delayed, the processing of the next node considered is not delayed. Hence the running time is dominated by the non-delayed processing steps. If the processing of a node is not delayed either two $(\mathcal{S}'', \varphi)$-components are merged or a node w in T is marked as finished. Hence the algorithm stops after $O(n)$ processing steps, i.e., in $O(n)$ time, with all nodes of T being marked as finished.

Note that, if during the processing of a so far unfinished node w, we remove a crest separator X weakly going between the $(\mathcal{S}'', \varphi)$-components identified with w and a neighbor of w, we know that w itself contains no crest in $\mathcal{H}$. Hence, if X goes strongly between two crests in $\mathcal{H}$, then there is another crest separator going strongly between these two crests that also goes strongly between two $(\mathcal{S}'', \varphi)$-components identified with w and one of its neighbors. This together with the fact that we have chosen X as heaviest crest separator guarantees that property (a) is maintained during our processing. Since no crest separators are added into $\mathcal{S}''$, property (b) also holds. The fact that the algorithm marks a node as finished only if its identified $(\mathcal{S}'', \varphi)$-component contains the vertices of a crest in $\mathcal{H}$ implies that the invariant (I) holds before and after each processing of a node w. At the end of the algorithm invariant (I) guarantees that property (c) holds. To sum up, $(G, \varphi, c, \mathcal{H}, \mathcal{S}'')$ defines a good mountain structure at the end of the algorithm. $\square$

4 Connection between Coast Separators and Pseudo Shortcuts

As in the last section, $\mathcal{H}$ is a set of crests in a weighted, almost triangulated, and biconnected graph (G, φ, c). Let $(G, \varphi, c, \mathcal{H}, \mathcal{S})$ be a good mountain structure. As part of our algorithm, for each $(\mathcal{S}, \varphi)$-component, we want to compute a coast separator that strongly disconnects its crest in $\mathcal{H}$ from the coast and that is of weighted size $c \cdot \mathrm{tw}(G)$ for some constant c. In more detail, we are interested in a special kind of coast separators defined in the next paragraph and in Section 5, we choose $\mathcal{H}$ as a special set of crests that allow us to compute such coast separators.

For a cycle Q in (G, φ, c), we say that Q *encloses* a vertex u, a vertex set U, and a subgraph H of G, if the set of the vertices of the cycle weakly disconnects the coast from $\{u\}$, U, and the vertex set of H, respectively. A crest separator X *encloses* a vertex u, a vertex set U, and a subgraph H of G if it has a lowpoint and the cycle induced by the edges of the essential boundary of X. The *inner graph* of a cycle Q or a coast separator Q is the plane graph induced by the vertices that are weakly disconnected by Q from the coast. Recall that a coast separator for a set U is *minimal* for U if it has minimal weighted size among all coast separators for U and if among all such separators its inner graph has a minimal number of faces.

Observation 4.1. *The vertex set of a minimal coast separator for a connected set of vertices H in a weighted, almost triangulated graph induces a cycle.*

Unfortunately, for a crest $H \in \mathcal{H}$ in an $(\mathcal{S}, \varphi)$-component C, there might be no minimal coast separator Y for H of weighted size $O(tw(G))$ in G that is completely contained in C. In this case, there is a crest separator $X \in \mathcal{S}$ such that the cycle induced by Y contains a subpath P with the following properties: P starts and ends with vertices of X and, additionally, P is contained in the (X, φ)-component not containing C. Then, since P can not be replaced by a crest-separator path of X of at most the same length, P must be a pseudo shortcut as defined next. (For property (1) of the following definition, see the remark immediately after the definition.)

Definition 4.2 (pseudo shortcut, composed cycle, inner graph)**.** *Let $X = (P_1, P_2)$ be a crest separator in (G, φ, c) with an (X, φ)-component D. Let X^{CP} be a crest-separator path of X connecting vertices s_1 and s_2 part of the essential boundary of X. Then, a path P from s_1 to s_2 in D is called an* (s_1-s_2-connecting) (D-)pseudo shortcut (*of X^{CP}*) *if the following three conditions hold:*

(1) *if X encloses the (X, φ)-component opposite to D, then X has an exterior lowpoint, i.e, a lowpoint belonging to the coast.*

(2) *P has a strictly shorter weighted length than X^{CP}.*

(3) *P does not contain any vertex of the coast.*

We call the cycle consisting of the edges of P and of X^{CP} the composed cycle *of (X^{CP}, P). Moreover, a path P is a* pseudo shortcut of a crest separator X *if it is a pseudo shortcut for some crest-separator path of X. The* inner graph *of a pseudo shortcut P of a crest-separator path X^{CP} is the inner graph of the composed cycle of (X^{CP}, P).*

Roughly speaking, for a crest separator X with an interior lowpoint, we are only interested in pseudo shortcuts that are enclosed by X. These can be used for the construction of a coast separator for a crest $H \in \mathcal{H}$ in the (X, φ)-component D not enclosed by X. For the crests in $\mathcal{H}$ enclosed by X, we can use the essential boundary of X as a coast separator. This is the reason why we distinguish between exterior and interior lowpoints., and why we restrict our definition of pseudo shortcuts by Condition 1. In return, we so can avoid complicated special cases in the use of pseudo shortcuts.

Note that, for two vertices s_1 and s_2 with neither s_1 nor s_2 being equal to the lowpoint of a crest separator X, there is only one crest-separator path connecting s_1 and s_2, and we compare the weighted length of a weighted path P connecting s_1 and s_2 with the unique weighted length of the crest-separator path connecting s_1 and s_2. Moreover, we restrict the endpoints of a pseudo shortcut P to be part of the essential boundary since, otherwise, one of P_1 and P_2 must contain both endpoints. Then P cannot have a shorter length than the subpath of P_1 or P_2 connecting the two endpoints as shown by part (a) of the next lemma.

Lemma 4.3. *For each crest separator $X = (P_1, P_2)$ the following holds:*

(a) *For each $i \in \{1, 2\}$, no path with endpoints v_1 and v_2 in P_i can have a shorter weighted length than the shortest v_1-v_2-connecting crest-separator path.*

(b) *Let R be a ridge connecting two vertices of different crests H_1 and H_2 in $\mathcal{H}$. If the weighted length of X is not larger than the weighted length of a crest separator of shortest weighted length separating H_1 and H_2, R and a pseudo shortcut P of a crest-separator path X^{CP} of X cannot cross.*

Proof. To show part (a), note that every path from v_1 to v_2 has weighted length of at least $\max(h^+_\varphi(v_1) - h^-_\varphi(v_2), h^+_\varphi(v_2) - h^-_\varphi(v_1)) + 1$ whereas the crest-separator path with endpoints v_1 to v_2 has exactly that length. Thus, (a) holds.

To show part (b), let $\tilde{X}$ be a crest separator with shortest weighted length among all crest separators separating H_1 and H_2. See Fig. 4. Note that $\tilde{X}$ must cross R. Assume for a contradiction that (b) does not hold, i.e., R and a pseudo shortcut P of X^{CP} cross. Choose u among all crossing vertices of R and P with minimal upper height. With s_1 and s_2 being the endpoints of P, let Q be the

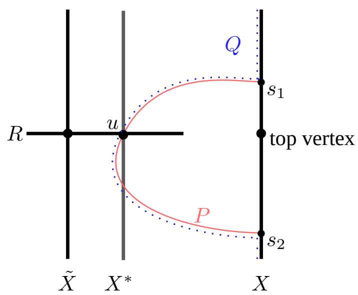

Figure 4: The crest separators $X, \tilde{X}, X^*$, the ridge R, the pseudo shortcut P, and the path Q as described in the proof of Lemma 4.3(b).

path obtained from $P_1 \circ P_2$ by replacing X^{CP} by P. Note that Q has a smaller weighted length than X since P is a pseudo shortcut of X^{CP}. Let P_1^* be the down path of u, and choose P_2^* as a down path starting in a neighbor of u such that $X^* = (P_1^*, P_2^*)$ is a crest separator of shortest weighted length that crosses R. X^* exists by Lemma 3.5.

The weighted length of P_1^* plus the weighted length of P_2^* can not be larger than the weighted length of Q since Q consists of a subpath Q' from a vertex of the coast to u and another subpath Q'' back to the coast. More precisely, assume that P_1^* and Q' leave R on the same side, whereas P_2^* and Q'' leave R on the other side. Let u' be the first vertex of Q''. If u' is not part of R, then, since $\{u, u'\}$ is not a down edge and hence the top edge of a crest separator crossing R, by definition of X^*, P_2^* must start in a vertex with lower or equal upper height than that of u'. Otherwise, u' has upper height at least $h_\varphi^+(u)$, whereas the upper height of the first vertex of P_2^* is at most $h_\varphi^+(u)$.

So far we can conclude that X^* has weighted length smaller than or equal to the weighted length of Q. As long as X^* and R cross more than once, there is another common vertex $\tilde{u}$ of R and X^* with $h_\varphi^+(\tilde{u}) < h_\varphi^+(u)$ and we redefine X^* as a crest separator crossing R with $\tilde{u}$ being a top vertex of X^*. Finally, X^* and R cross only once, and thus X^* strongly disconnects H_1 and H_2. Since our iteration of choosing X^* only shrinks the weighted length of X^*, X^* has weighted length smaller than or equal to the weighted length of Q, which is strictly smaller than the weighted length of X. Since the weighted length of X is smaller than or equal to the weighted length of $\tilde{X}$, the weighted length of X^* is strictly smaller than the weighted length of $\tilde{X}$. Contradiction. □

For the rest of this section, let $(G, \varphi, c, \mathcal{H}, \mathcal{S})$ be a good mountain structure of (G, φ, c). The endpoints of a pseudo shortcut for a crest separator $X \in \mathcal{S}$ are—intuitively speaking—the vertices between which a coast separator can leave one (X, φ)-component and later reenter the (X, φ)-component. We describe this intuition more precisely in Lemma 4.4. Let X be a crest separator, and let D be an (X, φ)-component. A subpath P'' of a path P' in G is called to be a (D, X)-*subpath of* P' if it is a path contained in D, that starts either with a lowpoint of X or an edge not being part of the essential boundary of X and that also ends either with a lowpoint of X or an edge not being part of the essential boundary of X. P'' is called *maximal* if no other (D, X)-subpath of P' contains P'' as a proper subpath.

A D-pseudo shortcut P of a crest-separator path X^{CP} of a crest separator $X \in \mathcal{S}$ is called *strict* if it has shortest weighted length among all D-pseudo shortcuts of X^{CP} and if among those the composed cycle (X^{CP}, P) encloses a minimum number of faces. Intuitively, the next lemma shows that in some cases, the part of a coast separator behind a crest separator is a so-called strict pseudo shortcut and that an analogous result holds for parts of a strict pseudo shortcut behind a crest separator. See also the examples left from the arrows in Fig. 5.

Lemma 4.4. *Let D and D^* be the (X, φ)-components for a crest separator $X \in \mathcal{S}$ such that D^* is not enclosed by X or X has an exterior lowpoint. Moreover, let P either be*

(a) the part contained in D of a minimal coast separator Y for a crest $H \in \mathcal{H}$ in D^* or

(b) a strict D-pseudo shortcut of a crest-separator path X^{CP} of X.

Then, for all crest separators $X' = (P_1', P_2') \in \mathcal{S}$ with an (X', φ)-component $D' \subseteq D$, possibly $X' = X$, P has at most one maximal (D', X')-subpath. If P has such a path P^*, P^* is a strict D'-pseudo shortcut of the crest-separator path $(X')^{\mathrm{CP}}$ of X' such that $(X')^{\mathrm{CP}}$ has the same endpoints as P^* and is contained in the inner graph I of

- *the cycle induced by the vertex set of Y (Case (a)) or*
- *the composed cycle Q of (X^{CP}, P) (Case (b)).*

Proof. Roughly speaking, we first want to show that, for all $j \in \{1, 2\}$, there is at most one crossing between P and P_j'. In fact, we want to show something more with respect to two concerns.

1. If P starts in a common vertex of X and X', we also want to consider this entering of D' as a crossing vertex. Therefore, we let $\tilde{P}$ be a path obtained from P by adding two edges of D^* not being border edges of X at the beginning and the ending of P.

2. We consider one of possible several crossings of $\tilde{P}$ and P_j' and we mark all crossing vertices that belong to exactly this crossing. More precisely, we choose the crossing whose crossing vertices are the last crossing vertices on P_j'. Note, that the marked vertices induce a subpath of P_j'. In the next paragraph, we show that no other vertex of $\tilde{P}$ can appear before the marked vertices on P_j'. This implicitly shows that, for each $j' \in \{1, 2\}$, there can be at most one crossing between $\tilde{P}$ and P_j' and, more important, that there can be only at most one maximal (D', X')-subpath.

Let v be the last marked vertex on P_j'. Since the coast is not part of I, all vertices after v on P_j' are not part of I. Hence, if P_j' contains vertices of $\tilde{P}$ before the marked vertices, then the subpath of P_j' between the last such vertex s' and the first marked vertex s'' is contained in I and can be replaced by the crest-separator path of X between s' and s'' (see Fig. 5 for two possible examples). This replacement does not increase the weighted length of $\tilde{P}$/of Y (Lemma 4.3(a)), but reduces the number of inner faces of I; which is a contradiction to the definition of $\tilde{P}$ being a strict pseudo shortcut or being part of a minimal coast separator. Thus, our assumption that there are vertices of $\tilde{P}$ before the marked vertices on P_j' is wrong.

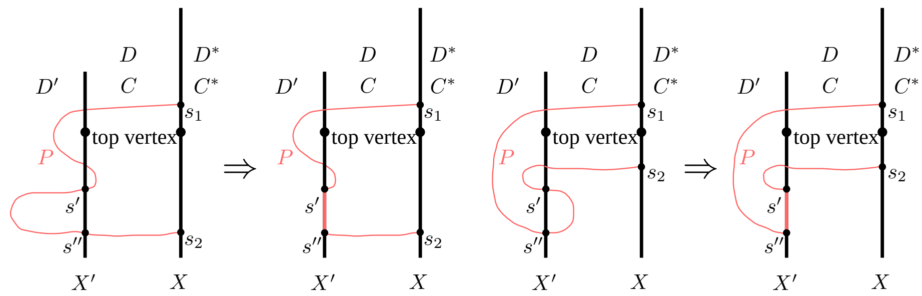

Figure 5: The replacement of a pseudo shortcut P crossing X' more than once.

We next conclude that, if there is a maximal (D', X')-subpath P', since it is the only one and since P is a strict pseudo shortcut or part of a minimal coast separator, P' must have a shorter weighted length than the crest-separator path of X' that is part of I and between the endpoints of P'. Note also that, if X' has an interior lowpoint, it cannot enclose the (X', φ)-component opposite to D' since, otherwise, X must also have an interior lowpoint and must enclose D^*. Hence P' is a pseudo-shortcut of $(X')^{\mathrm{CP}}$. Since $\tilde{P}$ is a strict pseudo shortcut or part of a minimal coast separator, P' as the only maximal (D', X')-subpath must be strict. □

In the following we want to compute pseudo shortcuts for the different crest separators by a bottom-up traversal in the mountain connection tree T of $(G, \varphi, c, \mathcal{H}, \mathcal{S})$. Therefore, let us assume that T is rooted at some arbitrary node. Let X be a crest separator going weakly between an $(\mathcal{S}, \varphi)$-component C and its parent in T, and let D be the (X, φ)-component D containing C. The idea to construct a D-pseudo shortcut of X can be described as follows: A D-pseudo shortcut first follows a path in C and possibly, after reaching a vertex v of C that is part of a crest separator $X' \in \mathcal{S}$ weakly going between C and a child C' of C in T, it follows a (precomputed) D'-pseudo shortcut of P' for the (X', φ)-component D' containing C', then it follows again a path in C and, after possibly containing further pseudo shortcuts, it returns to C and never leaves C anymore. Indeed, if a D-pseudo shortcut of X, immediately after reaching a vertex v of C that is part of a crest separator X' with the properties described above, contains an edge outside the (X', φ)-component containing C, then we show that P contains a D'-pseudo shortcut P' for the (X', φ)-component D' containing C'. However, since X may not completely contained in C, a D-pseudo shortcut of X may not start in a vertex of C. This is the reason why, for subsets $\mathcal{S}', \mathcal{S}'' \subseteq \mathcal{S}(G, \varphi, c)$ with $\mathcal{S}' \subseteq \mathcal{S}''$ (an example for $S' \neq S''$ can be found in Lemma 4.6) and for an $(\mathcal{S}', \varphi)$-component C, we define the *extended component* $\mathrm{ext}(C, \mathcal{S}'')$ as the plane graph obtained from C by adding the border edges of all crest separators $X \in \mathcal{S}''$ with a top edge in C. This should mean of course that the endpoints of the border edges are also added as vertices to C. As embedding of $\mathrm{ext}(C, \mathcal{S}'')$, we always take $\varphi|_{\mathrm{ext}(C, \mathcal{S}'')}$.

The next three lemmas prove some properties of extended components that allows us to guarantee the existence of pseudo shortcuts with nice properties in Lemma 4.8 from which we show in Lemma 4.9 that they can be constructed efficiently.

Lemma 4.5. *Let C be an $(\mathcal{S}, \varphi)$-component and let e be an edge with exactly one endpoint v in $\mathrm{ext}(C, \mathcal{S})$. Then, v is part of a crest separator in $\mathcal{S}$ with a top edge in C.*

Proof. By the definition of $\mathrm{ext}(C, \mathcal{S})$ the lemma holds if v is not contained in C. It remains to consider the case that v is in C. The definition of an $(\mathcal{S}, \varphi)$-component implies that v is a vertex of a crest separator in $\mathcal{S}$. We define the *boundary* of C to be the graph that consists of the vertices and edges of C that are incident to a face f of $\varphi|_C$ such that f is not a face of φ. (Roughly speaking, f is the union of several faces of φ.)

Let u be a vertex of largest upper height such that there is a down path from u to v that is contained in the boundary of C. Since G is biconnected, Obs. 3.4 implies that C is biconnected. Thus, u is incident to two edges $\{u, v_1\}$

and $\{u, v_2\}$ on the boundary of C. Assume for a moment that both edges are down edges. Since each vertex is connected by down edges to at most one vertex of smaller upper height and since down edges only connect vertices of different upper heights, either v_1 or v_2 must have larger upper height than u. This is a contradiction to our choice of u. Consequently, one of $\{u, v_1\}$ and $\{u, v_2\}$ is a top edge in C that belongs to a crest separator $X \in \mathcal{S}$ with a top edge in C. Since the down path from u contains v, it follows that X contains v. □

For the next two lemmas, let D and $\tilde{D}$ be the two (X, φ)-components of a crest separator X in $\mathcal{S}(G, \varphi, c)$, and let $\mathcal{S}$ be a set of crest separators with $\{X\} \subseteq \mathcal{S} \subseteq \mathcal{S}(G, \varphi, c)$.

Lemma 4.6. *The extended component* $\mathrm{ext}(D, \mathcal{S})$ *consists only of the vertices and edges of D, the border edges of X, and their endpoints.*

Proof. Each vertex $v \in \mathrm{ext}(D, \mathcal{S})$ either belongs to D or is an endpoint of a border edge of a crest separator with a top edge $\{v', v''\}$ in D, i.e., v is reachable by a down path from vertex v' or v''. A down path P starting in a vertex of D can leave D only after reaching a vertex $x \in X$. But after reaching x, P must follow the down path of x and therefore all edges after x on P must be border edges of X. □

Lemma 4.7. *If e is an edge with exactly one endpoint v in* $\mathrm{ext}(D, \mathcal{S})$*, then v is part of X.*

Proof. Note that there are no direct edges from a vertex $v \in D$ to a vertex $\tilde{v} \in \tilde{D}$ with neither v nor $\tilde{v}$ being part of X. Hence, v must be part of X by Lemma 4.6. □

As mentioned above, we want to precompute pseudo shortcuts for the crest separators in $\mathcal{S}$ since we later want to use them to construct coast separators. This only works if the pseudo shortcuts have some nice properties (in some kind similar to the properties of strict pseudo shortcuts, but the strict pseudo shortcuts have the problem that they cannot be computed efficiently). Therefore, let us consider a crest separator X_0 and an (X_0, φ)-component D of X_0. Moreover, let us root the mountain connection tree T of $(G, \varphi, c, \mathcal{H}, \mathcal{S})$ such that for the $(\mathcal{S}, \varphi)$-components C and C_0 containing the top edge of X_0, the $(\mathcal{S}, \varphi)$-component C_0 not being contained in D is the parent of the other $(\mathcal{S}, \varphi)$-component C. Moreover, let $C_1, \ldots, C_j$ be the children of C in T and, for $i \in \{1, \ldots, j\}$, X_i be the crest separator with a top edge in C and C_i. Finally, let D_i ($i \in \{1, \ldots, j\}$) be the (X_i, φ)-component that contains C_i. Then, an s_1-s_2-connecting D-pseudo shortcut P of X_0 is called *nice* if it has shortest weighted length among all s_1-s_2-connecting D-pseudo shortcuts of X_0 and if it consists exclusively of subpaths in $\mathrm{ext}(C, \mathcal{S})$ and, for each $i \in \{1, \ldots, j\}$, of at most one nice D_i-pseudo shortcut P_i for some crest-separator path X_i^{CP} of X_i that is part of the inner graph of the composed cycle of (X_0^{CP}, P). The following lemma guarantees the existence of such pseudo shortcuts.

Lemma 4.8. *If there is an s_1-s_2-connecting D^*-pseudo shortcut for the (X^*, φ)-component D^* of a crest separator $X^* \in \mathcal{S}$, then there is also such a pseudo shortcut that is nice.*

Proof. First, observe that with an s_1-s_2-connecting D^*-pseudo shortcut of X^*, there is also a strict s_1-s_2-connecting D^*-pseudo shortcut of X^*. We next show that, for each crest separator X_0 and each (X_0, φ)-component D of X_0, each strict D-pseudo shortcut P is nice. This is shown by induction over the number of crest separators $X \in \mathcal{S}$ for which there is a crossing of X and P. Let us define $C, C_1, \ldots, C_j$, $X_0, X_1, \ldots, X_j$, and $D, D_1, \ldots, D_j$ as in the definition of nice pseudo shortcuts. If there are no crossings between P and crest separators $X \neq X_0$ in $\mathcal{S}$, P is just a pseudo shortcut of shortest weighted length connecting the endpoints of P and is contained in $\text{ext}(C, \mathcal{S})$. Then P is clearly nice. Otherwise, let us consider the first edge e of P not contained in $\text{ext}(C, \mathcal{S})$. Hence, one endpoint v' of e must be part of a crest separator X_i with $i \in \{1, \ldots, j\}$—here we use Lemma 4.5 and the fact that $X_0, X_1, \ldots, X_j$ are the only crest separators with a top edge in C; $i \neq 0$ since e is in D—and the other endpoint is contained in the (X_i, φ)-component opposite to D_i. Thus, there must be a vertex v'' after v' on P that is contained in $\text{ext}(C, \mathcal{S})$. W.l.o.g., let v'' be the first such vertex. Because of Lemma 4.7, v'' must be also part of X_i. Since P is a strict D-pseudo shortcut of X_0, by Lemma 4.4, it has at most one maximal D_i-subpath P_i from v' to v'', and it must be a strict pseudo shortcut of the crest-separator path X_i^{CP} of X_i with endpoints v' and v'' such that X_i^{CP} is contained in the composed cycle (X_0^{CP}, P). Since the number of crest separators for which there is a crossing of the crest separator and P_i is smaller than the corresponding number for the whole path P, we can conclude that the subpath from v' to v'' is nice. If there are further parts of P not contained in $\text{ext}(C, \mathcal{S})$, they can similarly shown to be the only and nice pseudo shortcut for one of the other crest separators in $\{X_1, \ldots, X_j\} \setminus \{X_i\}$. Together with the fact that P as a strict pseudo shortcut is a D-pseudo shortcut of shortest weighted length between its endpoints, we can conclude that P is nice. $\square$

For an (X, φ)-component D of a crest separator $X \in \mathcal{S}$, let us define a *d-bounded D-pseudo shortcut set for X* to be a set consisting of an s_1-s_2-connecting nice D-pseudo shortcut P of weighted length at most d for each pair s_1 and s_2 of vertices of X for which such an s_1-s_2-connecting D-pseudo shortcut exists. For our mountain structure $\mathcal{M} = (G, \varphi, c, \mathcal{H}, \mathcal{S})$, we also define a *$d$-bounded $\mathcal{M}$-shortcut set* to be a set consisting of the union of a d-bounded D-pseudo shortcut set for each crest separator X and each (X, φ)-component D. The main aim of the following Lemma 4.9 is to efficiently construct a d-bounded $\mathcal{M}$-shortcut set. The pseudo shortcuts in such a set are later used for the construction of coast separators. However, in order to avoid an "overlapping" of coast separators, we will remove some of the constructed pseudo shortcuts. For making such a removal more efficient, we have to store some additional informations and to introduce some further definitions.

An $(\mathcal{S}, \varphi)$-component C is called the *root component* of a D-pseudo shortcut P of a crest separator $X \in \mathcal{S}$ if C is the $(\mathcal{S}, \varphi)$-component containing a top edge of X and, additionally, C is contained in D. A set $\mathcal{P}$ of pseudo shortcuts of crest separators in $\mathcal{S}$ is *consistent w.r.t. $\mathcal{S}$* if, for each pseudo shortcut P contained in $\mathcal{P}$ the following holds: If P has a subpath P' such that (1) P' is a pseudo shortcut of a crest separator $X' \in \mathcal{S}$ and (2) the vertex sets of P' and X are weakly disconnected by the vertex set of X'. Intuitively speaking, condition (2) guarantees that P' is not a pseudo shortcut on the wrong side of X'. If $\mathcal{S}$ is clear from the context, we call $\mathcal{P}$ simply consistent. The *consistence* *graph* of a consistent set $\mathcal{P}$ of pseudo shortcuts is a directed graph whose node set consists of the pseudo shortcuts $P \in \mathcal{P}$ and and whose edge set has an edge from a pseudo shortcut $P \in \mathcal{P}$ to a pseudo shortcut $P' \in \mathcal{P}$ of a crest separator $X' \neq X$ if and only if (1) P' is a subpath of P and (2) the root component of P' is a neighbor of the root component of P in the mountain connection tree.

For the next lemma keep in mind that G is weighted ℓ-outerplanar; in particular, all crest separators have weighted length at most 2ℓ.

Lemma 4.9. *Let $d \leq \ell$ and $\mathcal{M} = (G, \varphi, c, \mathcal{H}, \mathcal{S})$. Then a consistent d-bounded $\mathcal{M}$-shortcut set $\mathcal{P}$ and the consistence graph of $\mathcal{P}$ can be constructed in $O(|\mathcal{H}|\ell^3 + |V|\ell)$-time. Within the same time one can determine the root component of each pseudo shortcut in $\mathcal{P}$.*

Proof. We already know that we can compute the mountain connection tree T of $(G, \varphi, c, \mathcal{H}, \mathcal{S})$ in $O(|V|\ell)$ time (Lemma 3.10). We then root T and compute the pseudo shortcut sets in a bottom-up traversal of T followed by a top-down traversal. Roughly speaking, the bottom-up traversal computes the pseudo shortcut sets for the (X, φ)-components below crest separators X whereas the top-down traversal computes the pseudo shortcut sets for the (X, φ)-components above crest separators X. More precisely, let us assume that in the bottom-up traversal of the mountain connection tree, we want to compute a D-pseudo shortcut set for a crest-separator X_0 such that D consists of the union of $(\mathcal{S}, \varphi)$-components of a complete subtree of the mountain connection tree. Let us define $C, C_0, C_1, \ldots, C_j$, $X_0, \ldots, X_j$, and $D, D_1, \ldots, D_j$ in the same way as in the definition of a nice pseudo shortcut before Lemma 4.8. Then we can assume that we have already computed a D_i-pseudo shortcut set named $\mathcal{L}(D_i)$ for X_i for all $i \in \{1, \ldots, j\}$.

If X_0 has an interior lowpoint and does not enclose C, by definition of the pseudo shortcuts, there are no D-pseudo shortcuts of X_0, i.e., the D-pseudo shortcut set $\mathcal{L}(D)$ for X_0 is empty. Otherwise, we determine the set by using a single-source shortest path algorithm for each vertex s of X_0 as source vertex on the vertex-and-edge-weighted graph C' described in the next paragraph. In contrast to the rest of the paper, in this proof we consider also graphs with both, vertex and edge weights. We define the *weighted length* of a path in a graph with vertex and edge weights as the total sum of the weights of the vertices and edges of the path. In addition, the *distance* of two vertices in a graph (with or without edge weights) is the minimal weighted length of a path connecting the vertices. Note that the classical Dijkstra-algorithm [9] can be easily modified to compute the distances from a source vertex to all other vertices in a graph with vertex and edges weights in time linear to the size of the given graph plus the maximal number of different considered distances during the computation. In the same time, the algorithm can additionally compute a so-called *shortest-path tree* that, given a vertex t, allows us to find a path of shortest weighted length connecting the source vertex and t in a time linear to the number of vertices of the path.

The graph C' is obtained from $\mathrm{ext}(C, \mathcal{S})$ by deleting all vertices of the coast and, for each $i \in \{1, \ldots, j\}$ and each nice pseudo shortcut P in $\mathcal{L}(D_i)$, inserting an edge e_P connecting the endpoints of P. We say that e_P *represents* P. The weight of a vertex is equal to its weight in G. Whereas the weight of each edge of $\mathrm{ext}(C, \mathcal{S})$ is zero, the weighted length for an edge representing a pseudo shortcut is equal to the weighted length of the pseudo shortcut plus ϵ minus the weights of its endpoints, where $\epsilon = 1/(2\ell)$ is a small penalty that makes a shortest weighted path with many pseudo shortcuts a little bit more expensive than a shortest weighted path with fewer pseudo shortcuts. We subtract the weight of its endpoints since they are already taken into account as weights of the endpoints. Note that, for each D-pseudo shortcut P of X_0, the integer parts of the distances of the two endpoints of P in C' and in D are the same (Lemma 4.8) since P is of length $\leq (2\ell - 1)$ so that the sum of the fractional parts is of size $\leq (2\ell - 1)\epsilon < 1$.

Hence, after running the modified Dijkstra-algorithm $O(\ell)$ times, we know the distance in D of each pair of vertices s and t of X_0. Thus, we can test if their distance is shorter than the weighted length of a crest-separator path of X_0 from s to t and if the integer part of their distance is at most d. If both is true, there is a pseudo shortcut connecting s and t of length at most d. Then, using the shortest path tree we compute such a pseudo shortcut P^* from a path of shortest weighted length in C' by replacing each edge $e_P = \{x, y\}$ representing a pseudo shortcut P by P itself. Thereby we also add an edge (P^*, P) into the consistence graph of $\mathcal{P}$. Finally, we add P^* to the pseudo shortcut set $\mathcal{L}(D)$ and store C as the root component of P^*.

Before analyzing the running time, let us define $m_{C'}$ to be the number of edges in C' and n_C to be the number of vertices in C. Since all pseudo shortcuts have a weighted length less than 2ℓ—they are shorter than the weighted length of their crest separators—we can terminate every single-source shortest path computation after the computation of all vertices for which the distance to the source vertex is smaller than 2ℓ. Since there are $(2\ell - 1)^2$ possible distances values to consider, namely $0, \epsilon, 2\epsilon, \ldots, 2\ell - 1 + (2\ell - 1)\epsilon$, and since there are at most ℓ^2 edges introduced for the pseudo shortcuts of each crest separator in $\mathcal{S}$, each of the $O(\ell)$ single source-shortest paths problems on C' can be solved in $O(m_{C'} + \ell^2) = O(n_C + (j+1)\ell^2)$ where j is the degree $\deg_T(w)$ for the node w in T identified with C. Note that the number of nodes in T is $O(|\mathcal{H}|)$. Therefore, the whole bottom-up traversal runs in in $O(|\mathcal{H}|\ell^3 + |V|\ell)$ time.

Afterwards in a top-down traversal, we consider each node C of T with its crest separators $X_0, \ldots, X_j$ defined as before. Let D' be the (X_0, φ)-component not containing C, which is opposite to D, and let D'_i $(i \in \{1, \ldots, j\})$ be the (X_i, φ)-component containing C, which is opposite to D_i. For each such crest separator $X_i \in \{X_1, \ldots, X_j\}$, we can compute a D'_i-pseudo shortcut set as follows. We first determine $q \in \{1, \ldots, j\}$ such that X_q is a crest separator with the largest weighted length among all crest separators in $\{X \in \{X_1, \ldots, X_j\} \mid (X$ has no lowpoint) or (X has an exterior lowpoint) or (X encloses C)$\}$. Intuitively, X_q is the crest separator of largest weighted length among the crest separators in $\{X_1, \ldots, X_j\}$ for which we need to compute pseudo shortcuts.

Then, we compute the pseudo shortcut set $\mathcal{L}(D'_q)$ for X_q analogously as described in the bottom-up traversal. For the remaining crest separators $X_i \in \{X_1, \ldots, X_j\}/\{X_q\}$, we proceed differently: Let C' be the graph obtained from $\mathrm{ext}(C, \mathcal{S})$ by deleting all vertices of the coast and adding edges e_P for all pseudo shortcuts P in $\mathcal{L}(D') \cup \bigcup_{i \in \{1,\ldots,j\} \setminus \{q\}} \mathcal{L}(D_i)$ such that e_P connects the endpoints of P, where $\mathcal{L}(D')$ is the empty set if C is the root of the mountain connection tree. Note that a non-empty pseudo shortcut set $\mathcal{L}(D')$ was computed in the previous step of the top-down traversal whereas all remaining pseudo shortcut sets were already computed during the bottom-up traversal. Assign a weight to each such edge e_P that is equal to the weighted length of P plus ϵ minus the weights of its endpoints, where $\epsilon = 1/(2\ell)$. All other edges of C' have weight 0 and the vertices v of C' have weight $c(v)$; thus, equal to their weights in G. In C', determine the distance $d(x, y)$ of x and y for all vertices $x \in X_q$ and y of C'.

After the computation of these distances we can determine the D'_i-pseudo shortcut set in a reduced subgraph G_i for each $i \in \{1, \ldots, j\} \setminus \{q\}$. For its definition, let us consider the $(\{X_i, X_q\}, \varphi)$-component C^*_i that contains the top edges of X_i and X_q. If we remove all vertices of the coast from the plane graph $\mathrm{ext}(C^*_i, \mathcal{S})$, the resulting graph is divided into two sides by two ridges connecting the crest $H \in \mathcal{H}$ contained in C with the crests $H_i \in \mathcal{H}$ and $H_q \in \mathcal{H}$ of the (S, φ)-components $\neq C$ containing the top edge of X_i and X_q, respectively. Vertices of the ridge belong to both sides. More formally, a *side* is the graph induced by a maximal set of vertices in $\mathrm{ext}(C^*_i, \mathcal{S})$ that do not belong to the coast and that are not weakly separated by the vertices part of the two ridges. For an illustration, see Fig. 6. G_i contains the vertices and the edges of X_i and X_q except the vertices belonging to the coast. In addition, G_i has edges between each vertex x on X_q and each vertex y of X_i with x and y being part of the same side of $\mathrm{ext}(C^*_i, \mathcal{S})$. We assign to each such edge $\{x, y\}$ a weight $d_{G_i}(x, y) = d(x, y) - c(x) - c(y)$. Recall that D_q is the (X_q, φ)-component opposite to D'_q. Finally for each pair of vertices x and y of X_q, G_i has an edge $\{x, y\}$ with weighted length $d_{G_i}(x, y) = d_{X_q}(x, y) - c(x) - c(y)$, where $d_{X_q}(x, y)$ is either ϵ plus the weighted length of a D_q-pseudo shortcut of X_q, if it exists, or otherwise, the weighted length of the crest-separator path of X_q from x to y.

For each pair of vertices s_1 and s_2 on the essential boundary of X_i belonging to different sides of C', we then determine a path P of shortest weighted length from s_1 to s_2 in G_i and test whether its weighted length is shorter than the weighted length of the crest-separator path of X_i from s_1 to s_2 and if the integer part of their distance is at most d. If both is true, we replace each edge $\{x, y\}$ of P of weighted length $d_{G_i}(x, y)$ by a path in D'_i with length $d_{G_i}(x, y)$ and endpoints x and y. Finally, we add the so modified path P^* to the pseudo shortcut set $\mathcal{L}(D'_i)$ for X_i and store C as root component of P^*. Moreover, analogously to the bottom-up traversal, if some subpaths of P^* result from replacing an edge $e_{P'}$ representing a pseudo shortcut P' by the pseudo shortcut P' itself, we add an edge (P^*, P') into the consistence graph.

We next show the correctness of those steps of the top-down traversal that are different from those of the bottom-up traversal. As mentioned above the concatenation R of the ridges between H and H_i and between H and H_q divides C^*_i into two sides. If there is a D'_i-pseudo shortcut P connecting s_1 with s_2, this path must lead from the side containing s_1 to the other side containing s_2.

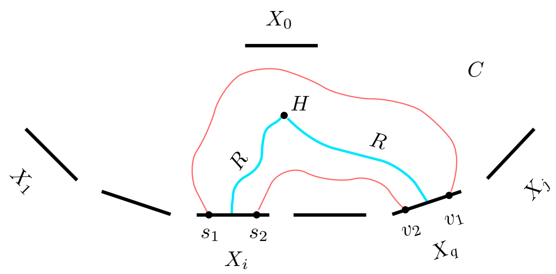

Figure 6: Computing a pseudo shortcut for X_i.

It cannot cross R by Lemma 4.3(b). Hence the only possibility for a pseudo shortcut to change sides is to use a pseudo shortcut of X_q, but not several times because of Lemma 4.3(a). To sum up, P consists of the concatenation of

- a path from s_1 to a vertex v_1 of X_q such that the path is completely contained in the same side as s_1,
- a pseudo shortcut from v_1 to a vertex v_2 of X_q that is on the other side
- a path from v_2 to s_2 being completely contained in the same side as s_2.

Therefore, the construction of G_i implies that, for each D'_i-pseudo shortcut P for X_i connecting a vertex s_1 and a vertex s_2, there is a path from s_1 to s_2 of the same length as P in G_i. Note also that the distance of two vertices in G_i is never smaller than the distance of that vertices in D'_i since each path in G_i can be replaced by a path in D'_i with the same endpoints and the same length. Thus, it is correct to use the graph G_i for our computation of a pseudo shortcut set $\mathcal{L}(D'_i)$ for X_i.

Concerning the running time, note that the distances for the edges $\{x, y\}$ of C' or of G_i with x and y being endpoints of a pseudo shortcut are already computed during the bottom-up traversal or in a previous step of the top-down traversal (as already mentioned). Recall that $j = \deg_T(w)$. All other distances needed for the construction of all graphs $G_1, \ldots, j$ can be computed by $O(\ell)$ single-source shortest paths computations in C', one for each vertex of X_q as source vertex. If once again, $m_{C'}$ and n_C are the number of edges of C' and vertices of C, respectively, each of the $O(\ell)$ single-source shortest path computations runs in $O(m_{C'} + \ell^2) = O(n_C + (j+1)\ell^2)$ time. Each subgraph G_i consists of $O(\ell^2)$ edges, and we have to consider only $4\ell^2$ distances values. Thus, the $O(\ell)$ single-source shortest path computations on all graphs $G_1, \ldots, G_j$ run in total time $O(j \cdot \ell(\ell^2 + 4\ell^2)) = O(j \cdot \ell^3)$. Consequently, the whole top-down traversal can be done in $O(|\mathcal{H}|\ell^3 + |V|\ell)$ time. $\square$

5 Computing Coast Separators

In this section we want to show how one can construct a set of pairwise non-crossing coast separators such that the inner graphs of the coast separators contain all crests of 'large' height. For this purpose, we use coast separators of three different types and for each such type we present at least one central technical lemma that helps us to guarantee the disjointness of the constructed coast separators. Recall that $c_{\max}$ is the maximum weight over all vertices.

Take $k \in I\!N$. Let (G^+, φ^+, c^+) be a weighted, almost triangulated, and biconnected plane graph with treewidth $k' \leq k$, where φ^+ is a $(2k + 2c_{\max})$-weighted-outerplanar embedding. Moreover, let (G, φ, c) be the weighted, almost triangulated, and biconnected plane graph obtained from (G^+, φ^+, c^+) as follows: each maximal connected set of vertices of lower height at least $k + 2c_{\max}$ is merged to one vertex. Let $\mathcal{H}$ be the set of merged vertices where each vertex is obtained from merging a maximal connected set of vertices of lower height at least $k + 2c_{\max}$. Then we define $c(v) = 1$ for all $v \in \mathcal{H}$ and $c(v) = c^+(v)$ for all other vertices v of G. Note that $\mathcal{H}$ is a set of different crests in (G, φ, c).

In the following, assume that we are given the integer k and, beside the graph (G, φ, c) and the crest set $\mathcal{H}$, a set $\mathcal{S}$ of crest separators such that $(G, \varphi, c, \mathcal{H}, \mathcal{S})$ is a good mountain structure as well as the mountain connection tree T of this good mountain structure.

Enclosing crest separators. The first type of coast separators that we use is a cycle induced by the essential boundary of a crest separator with an interior lowpoint. The following lemma describes an important property for these kind of coast separators:

Lemma 5.1. *An $(\mathcal{S}, \varphi)$-component C can not be enclosed by more than one crest separator with a top edge in C.*

Proof. Assume that there are two crest separators X_1 and X_2 enclosing C that have a top edge in C. Let I_i ($i \in \{1, 2\}$) be the inner graph of the essential boundary of X_i. If X_1 encloses C, then the top edge e_2 of X_2 is part of I_1, but not part of the essential boundary of X_1. By definition of the down paths the essential boundary of X_2 cannot cross the essential boundary of X_1. Thus, I_2 is a subgraph of I_1. Moreover, since the top edge e_1 of X_1 can not be an edge of X_2, edge e_1 can not be part of I_2. Then X_2 can not enclose C since C contains e_1. □

Composed cycles. The next two lemmas describe important properties of pseudo shortcuts and composed cycles. A composed cycle is the second type of coast separators that we use.

Lemma 5.2. *Let P be a D-pseudo shortcut for a crest-separator path X^{CP} of a crest separator $X \in \mathcal{S}$ with an (X, φ)-component D. For the $(\mathcal{S}, \varphi)$-component C containing the top edge of X and being contained in D, the composed cycle of (X^{CP}, P) encloses the crest in $\mathcal{H}$ contained in C.*

Proof. We define C' to be the $(\mathcal{S}, \varphi)$-component different from C containing the top edge of X. H and H' should denote the crests in $\mathcal{H}$ contained in C and C', respectively. Let R be a ridge between H and H'. Since P and R cannot cross (Lemma 4.3(b)), the composed cycle of (X^{CP}, P) encloses H. □

Lemma 5.3. *Let $C_0, C_1, \ldots, C_r$ consecutive nodes of a path in the mountain decomposition tree of $(G, \varphi, c, \mathcal{H}, \mathcal{S})$, and, for $i \in \{1, \ldots, r\}$, let X_i be the crest separator with a top edge in C_{i-1} and C_i (compare Fig. 7). Moreover, let D_1 be the (X_1, φ)-component containing C_1, and let D'_r be the (X_r, φ)-component containing C_{r-1}. Then, for a nice D_1-pseudo shortcut P_1 of X_1 that is completely contained in D'_r and for a nice D'_r-pseudo shortcut P_r of X_r that is completely contained in D_1, there exists an $i \in \{2, \ldots, r-1\}$ such that P_1 is completely contained in the (X_i, φ)-component containing C_1 and P_r is completely contained in the (X_i, φ)-component containing C_{r-1}.*

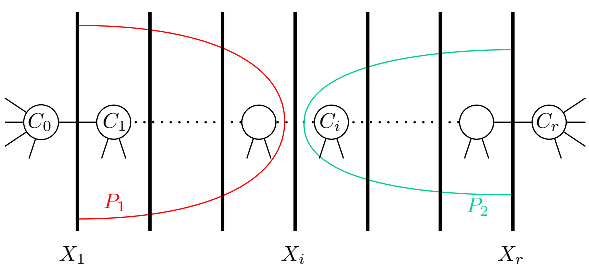

Figure 7: The (S, φ)-components and crest separators described in Lemma 5.3.

Proof. For $i \in \{2, \ldots, r-1\}$, let D_i be the (X_i, φ)-component containing C_i, and let D'_i be the (X_i, φ)-component containing C_{i-1}. Assume that the lemma does not hold, i.e., for at least one $i \in \{1, \ldots, r-1\}$, P_1 must have a maximal (D_i, X_i)-subpath P_1^* and P_r a maximal (D'_{i+1}, X_{i+1})-subpath P_r^*. Let us assume w.l.o.g. that we have chosen i as large as possible. Since P_1 and P_r are nice, P_1^* and P_r^* also define nice pseudo shortcuts and together with the crest separator paths between their endpoints enclose H (Lemma 5.2). Hence, P_1 together with the crest separator path between its endpoints also encloses H and, since we have chosen i as large as possible, P_1 is completely contained in D'_i whereas P_r^* together with the crest separator path between its endpoints encloses H (Lemma 5.2) and is completely contained in D_1 (see also Fig. 8). Hence P_1 and P_r^* intersect. As illustrated in Fig. 8. one can interchange subpaths of P_1 with subpaths of P_r^* of the same length such that afterwards the inner graphs of the new composed cycles do not intersect anymore except in some common vertices of the old paths P_1 and P_r^* and such that the new composed cycles do not enclose the crest H anymore. In the case of the modified version of P_r^*, this is a contradiction to Lemma 5.2. □

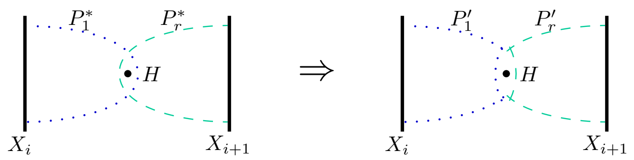

Figure 8: The interchange of a subpath of P_1 with a subpath of P_r^*.

Let $\mathcal{M} = (G, \varphi, c, \mathcal{H}, \mathcal{S})$. For a d-bounded $\mathcal{M}$-shortcut set $\mathcal{P}$, let us define a *valid d-bounded $\mathcal{M}$-shortcut set* to be a subset $\mathcal{P}'$ of $\mathcal{P}$ such that, for each $(\mathcal{S}, \varphi)$-component C, for which there is at least one crest separator $X \in \mathcal{S}$ with a top edge in C that has a D-pseudo shortcut in $\mathcal{P}$ for the (X, φ)-component D containing C, $\mathcal{P}'$ contains at least one D'-pseudo shortcut for a crest separator $X' \in \mathcal{S}$ with a top edge in C where D' is the (X', φ)-component containing C. Roughly speaking, each $(\mathcal{S}, \varphi)$-component C with a d-bounded pseudo shortcut has a d-bounded pseudo shortcut in a $\mathcal{P}'$. If "at least one" in the above definition is replaced by "exactly one", the valid d-bounded $\mathcal{M}$-shortcut set is also called *non-overlapping*. We next show that such a shortcut set can be computed efficiently.

Lemma 5.4. *Given a consistent d-bounded $\mathcal{M}$-shortcut set, the consistence graph of $\mathcal{P}$ as well as, for each pseudo shortcut $P \in \mathcal{P}$, its root component, a consistent non-overlapping d-bounded $\mathcal{M}$-shortcut set can be constructed in $O(|\mathcal{H}|k^2)$ time.*

Proof. Let us call a vertex of a directed graph to be a *source* if it has no incoming edge. If F is the consistence graph of $\mathcal{P}$, we repeatedly apply the following step.

> Take a not-yet-considered source P of F, and let C be the root component of P. If there is another pseudo shortcut with root component C, then remove P from $\mathcal{P}$ and F.

The existence of P' guarantees that the new set $\mathcal{P}$ is still valid. Since we only remove sources of F, the new set is also consistent. Thus, apart from the running time, we only have to show that, after having applied the above removal step as long as possible, we obtain a non-overlapping set of pseudo shortcuts. Therefore, let us assume that after the removal steps there is still one (S, φ)-component C for which there are two different crest separators X'_1 and X'_2 with top edges in C such that, for the (X'_1, φ)-component D'_1 and the (X'_2, φ)-component D'_2 containing C, there exist a D'_1-pseudo shortcut P'_1 of X'_1 and a D'_2-pseudo shortcut P'_2 of X'_2 in $\mathcal{P}$. Then neither P'_1 nor P'_2 can be a source of F since, otherwise, the removal step could be applied once more. Hence, let P_1 and P_2 be sources of F for which there are paths from P_1 to P'_1 and from P_2 to P'_2 in F. In particular, this means that P'_1 is a subpath of P_1 and P'_2 a subpath of P_2 in G. Let X_1 and X_2 be the crest separators such that P_1 and P_2 are pseudo shortcuts of X_1 and X_2, respectively. Since the removal step was applied as long as possible, for the root component of P_1 and equivalently for that of P_2, there is exactly one crest separator X with a pseudo shortcut in the (X, φ)-component containing the root component of P_1, namely $X = X_1$ and equivalently—concerning the root component of P_2—we have $X = X_2$. Hence P_i $(i = 1, 2)$ is completely contained in the (X_{3-i}, φ)-component that contains C. Since P_i with its subpath P'_i encloses the crest in C (Lemma 5.2), there is no crest separator disconnecting P_1 and P_2; a contradiction to Lemma 5.3.

Note that no source has to be considered a second time; if a source can not be removed because of a missing second pseudo-shortcut in an $(\mathcal{S}, \varphi)$-component, then this is the case until the end of the algorithm. Consequently, the running time is linear in the number of pseudo shortcuts if, for each source of F, we can decide whether it has to be removed in constant time. For that we store initially with each $(\mathcal{S}, \varphi)$-component C the number n_C of pseudo shortcuts with root component C. If such a pseudo shortcut is a source in F and removed by the algorithm, we decrease n_C by one. The update caused by a removal of a pseudo shortcut then can be done in constant time. The decision whether a pseudo shortcut P has to be removed reduces to the question whether n_C is greater than one for the root component C of P. This is correct because of the following: If there is another pseudo shortcut P' that allows us to remove P, then a subpath of P' is a pseudo shortcut with root component C, i.e., n_C is indeed greater than one. The initialization of all numbers stored with the components can be done in $O(|\mathcal{P}|)$ time. So the whole running time is bounded by $O(|\mathcal{P}|) = O(|\mathcal{H}|k^2)$. $\square$

Minimal coast separators. The third type of coast separators that we use are minimal coast separators. Such coast separators are used in a special kind of (S, φ)-components defined now. An $(\mathcal{S}, \varphi)$-component C is called *pseudo shortcut free* if (1) C is not enclosed by a crest separator in $\mathcal{S}$ with a top edge in C and an interior lowpoint and (2) for all crest separators $X \in \mathcal{S}$ with a top edge in C and the (X, φ)-component D containing C, there is no D-pseudo shortcut of weighted length $\leq k - 1$ for X.

Moreover, let us define a crest separator to be *pseudo shortcut free* if it has neither a pseudo shortcut of weighted length $\leq k - 1$ nor an interior lowpoint. Intuitively speaking, a pseudo shortcut is of interest if it is possibly part of a coast separator of weighted size at most k. This is the reason why we only consider pseudo shortcuts of weighted length $\leq k - 1$. As the next lemma shows, pseudo shortcut free components are separated by pseudo shortcut free crest separators.

Lemma 5.5. *Let $H', H'' \in \mathcal{H}$ be crests of different pseudo shortcut free $(\mathcal{S}, \varphi)$-components. Let $\mathcal{S}' \subseteq \mathcal{S}$ be the set of pseudo shortcut free crest separators. Then there is a crest separator in $\mathcal{S}'$ strongly going between the crests. In particular, a minimal coast separator for H' of weighted size $\leq k$ can only enclose those crests in $\mathcal{H}$ that are part of the $(\mathcal{S}', \varphi)$-component that contains H'.*

Proof. As illustrated in Fig. 9, let $C_0, \ldots, C_r$ be the $(\mathcal{S}, \varphi)$-components on the path in the mountain connection tree T of $(G, \varphi, c, \mathcal{H}, \mathcal{S})$ from the $(\mathcal{S}, \varphi)$-component C_0 containing H' to the $(\mathcal{S}, \varphi)$-component C_r containing H''. For $i \in \{1, \ldots, r\}$, let $X_i \in \mathcal{S}$ be the crest separator with a top edge part of both C_{i-1} and C_i. First note that no crest separator $X \in \{X_1, \ldots, X_r\}$ can have an interior lowpoint. Otherwise, such a crest separator would enclose either C_0 or C_r. This would imply that either X_1 encloses C_0 and has an interior lowpoint or X_r encloses C_r and has an interior lowpoint. This contradicts our choice of C_0 and C_r as pseudo shortcut free $(\mathcal{S}, \varphi)$-components.

For $i \in \{1, \ldots, r\}$, let D_i be the (X_i, φ)-component that contains C_r. Take X_i as the first crest separator in $X_1, \ldots, X_r$ with no D_i-pseudo shortcut of weighted length $\leq k-1$. X_i exists since X_r has no D_r-pseudo shortcut of weighted length $\leq k-1$. Then X_i also has no pseudo shortcut for the (X_i, φ)-component opposite to D_i. This follows either from the fact that C_0 is pseudo shortcut free if $i = 1$ and from Lemma 5.3 otherwise. Hence X_i is pseudo shortcut free. Finally note that a minimal coast separator of weighted size $\leq k$ for H' can only enclose those crests in $\mathcal{H}$ that are part of the $(\mathcal{S}', \varphi)$-component that contains H', since, otherwise, it must cross a crest separator X in $\mathcal{S}'$ and X then has a pseudo shortcut of weighted length $\leq k-1$ by Lemma 4.4. □

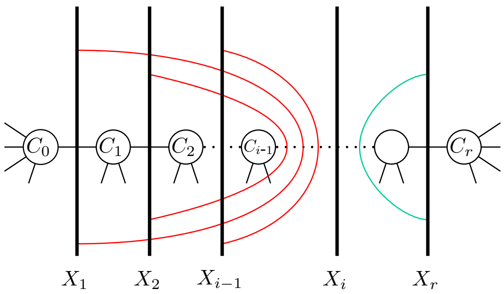

Figure 9: A path $\tilde{P}$ in the mountain connection tree T of the mountain structure $(G, \varphi, c, \mathcal{H}, \mathcal{S})$ with the crest separators $X_1, \ldots, X_r$ between the $(\mathcal{S}, \varphi)$-components corresponding to the nodes of $\tilde{P}$. If in the figure a curve representing a pseudo shortcut P together with its crest-separator path X^{CP} encloses a node representing an $(\mathcal{S}, \varphi)$-component C, this should mean that at least some vertices of C are enclosed by the composed cycle of (X^{CP}, P).

We next show that minimal coast separators can be computed efficiently.

Lemma 5.6. *There is an algorithm that computes a minimal coast separator Y for a given crest H in a weighted, almost triangulated graph (G, φ, c) in $O(nw(Y))$ time where n is the number of vertices of G and $w(Y)$ is the total weight of the vertices in Y.*

Proof. In an unweighted graph $G' = (V', E')$, by network flow theory, one can construct a separator Y' of minimal size for two connected sets S and T as follows: Merge the vertices of S and T to two single vertices s and t, respectively, and compute a maximal set P of internally vertex-disjoint paths from s to t in the graph $G^+ = (V^+, E^+)$ obtained. Let $V(P)$ be the set of the vertices of the paths in P. Now, define U as the set that, for each path P, contains the last vertex u of P for which there is a path from s to u in $G^+[(V^+ \setminus V(P)) \cup \{s, u\}]$. It is a well-known fact and not hard to see that, by the construction from above, U is a separator of minimal size for $\{s\}$ and $\{t\}$. For each separator U' of minimal size for the two sets $\{s\}$ and $\{t\}$, let us define $\mathrm{CC}(U')$ as the set of vertices that are in the same connected component as s in $G^+[V^+ \setminus U')]$. We now show that U is an $\{s\}$-*minimal separator*, i.e., $\mathrm{CC}(U) \subseteq \mathrm{CC}(U')$ for all separators U' of minimal size for $\{s\}$ and $\{t\}$. Note that a separator U' of minimal size for $\{s\}$ and $\{t\}$ can not have vertices outside $V(P)$ since it contains one vertex on each path of P and since $|P| = |U'|$. Since each vertex in the set $\mathrm{CC}(U)$ is connected to s by a path without any inner vertices in $V(P)$, these vertices must be contained in $\mathrm{CC}(U')$ for any separator U' for $\{s\}$ and $\{t\}$ of minimal size and therefore U is indeed $\{s\}$-minimal.

If we are given a weighted, almost triangulated graph (G, φ, c) with $G = (V, E)$, we can generalize the approach above to find a minimal coast separator for a set H: First, replace each vertex v of G by $c(v)$ copies. For each edge $\{u, v\}$ of G, add an edge between each copy of u with each copy of v. Let $G' = (V', E')$ be the unweighted graph obtained. Define S to be the vertex set consisting of the copies of the vertices of H, and let T be the vertex set consisting of the copies of the vertices of the coast. Then construct a separator Y' strongly disconnecting S and T as above. Let Y be the set of vertices of G whose copies are all in Y'. Then Y is an H-minimal separator. By the fact that G is almost triangulated, Y is a minimal coast separator for H in G.

For an efficient implementation, there is no need to replace (G, φ, c) by an unweighted graph. Instead, we use classical network flow techniques to construct a maximum number of paths connecting a vertex of H and a vertex of H' such that each vertex $v \notin H \cup H'$ is part of at most $c(v)$ paths and then read of a separator from these paths in a similar way as described above. Since we can construct each path in $O(n)$ time, and since we have to construct $O(w(Y))$ paths, we can compute a minimal coast separator in $O(nw(Y))$ time. $\square$

Algorithm. Recall that (G^+, φ^+, c^+) is $(2k + 2c_{\max})$-weighted-outerplanar. We next present an algorithm to construct a set $\mathcal{Y}$ of coast separators of size $O(k)$ for $\mathcal{H}$ in (G, φ, c). As we will see, these sets also define coast separators for the crests of height $2k + 2c_{\max}$ in (G^+, φ^+, c^+). By Observation 5.7 and Lemma 5.12, we will conclude that each crest in $\mathcal{H}$ is enclosed by exactly one coast separator in $\mathcal{Y}$. In Lemma 5.9, we show that the (S, φ)-components of the crests enclosed by a coast separator in $\mathcal{Y}$ induce a connected subtree of T.

Step 1: Run the algorithm from Lemma 4.9 to compute a consistent $(k-1)$-bounded $\mathcal{M}$-shortcut set $\mathcal{P}$ for $\mathcal{M} = (G, \varphi, c, \mathcal{H}, \mathcal{S})$, the consistence graph of $\mathcal{P}$ and for each pseudo shortcut in $\mathcal{P}$ its root component. This subsequently allows us to determine the set $\mathcal{S}'$ of all pseudo shortcut free crest separators in $\mathcal{S}$ as well as all $(\mathcal{S}', \varphi)$-components.

For each crest $H \in \mathcal{H}$ contained in a pseudo shortcut free $(\mathcal{S}, \varphi)$-component contained in an $(\mathcal{S}', \varphi)$-component C, compute a minimal coast separator Y_C as follows: Compute first the plane graph (C', ψ) obtained from the plane graph $(C, \varphi|_C)$ by adding an extra vertex v^* into the outer face of $(C, \varphi|_C)$ and by inserting edges from v^* to all vertices of C that are part of a crest separator in $\mathcal{S}'$ with a top edge in C. Choose the outer face of ψ such that v^* together with the vertices of C that belong to the coast of φ are the coast of ψ. Then, use the algorithm of Lemma 5.6 to construct a minimal coast separator Y_C for H in (C', ψ). The existence of Y_C is shown in Lemma 5.8. After having found Y_C, determine the inner graph of Y and all crests in $\mathcal{H}$ that are part of the inner graph. Add Y_C to the initial empty set $\mathcal{Y}$.

Step 2: Use Lemma 5.4 to construct a non-overlapping consistent $(k-1)$-bounded $\mathcal{M}$-shortcut set $\mathcal{Z} \subseteq \mathcal{P}$. Initialize E' as an empty set of directed edges. For each crest $H \in \mathcal{H}$ that is not enclosed by a coast separator constructed (possibly for some crest $H' \neq H$) in Step 1, take C as the $(\mathcal{S}, \varphi)$-component containing H and test whether there is a crest separator $X \in \mathcal{S}$ with a top edge in C that has an interior lowpoint and encloses C.

A: If so, let X_C be this crest separator and Y_C be the cycle induced by edges of the essential boundary of X_C.

B: Otherwise, define X_C to be the crest separator with a top edge in C that has a nice pseudo shortcut P in $\mathcal{Z}$. X_C exists since C is not pseudo shortcut free and is unique because $\mathcal{Z}$ is non-overlapping. Take $X_C^{\mathrm{C}P}$ as the crest-separator path of X_C such that P is a pseudo shortcut for $X_C^{\mathrm{C}P}$ and define Y_C as the composed cycle of $(X_C^{\mathrm{C}P}, P)$.

Let C^* be the $(\mathcal{S}, \varphi)$-component that contains the top edge of X_C, but is different to C. Finally add a directed edge (C, C^*) to E' if the following conditions are all satisfied.

Condition 1: C^* has a crest $H \in \mathcal{H}$ that is not enclosed by a coast separator constructed in Step 1.

Condition 2: Either C^* is not enclosed by X_C or X_C has not an interior lowpoint.

Condition 3: the D^*-pseudo shortcut set for X_C is empty where D^* is the (X_C, φ)-component containing C^*.

Intuitively, an edge (C, C^*) indicates the possibility that the coast separator constructed for the crest in C^* also encloses the crest in C (compare also Lemma 5.11).

Step 3: Let $\tilde{F}$ be the subgraph of the mountain connection tree whose vertices consist of all $(\mathcal{S}, \varphi)$-components that have a crest part of $\mathcal{H}$ such that the crest is not enclosed by the coast separators constructed in Step 1. The edges of $\tilde{F}$ are the edge set E' constructed in Step 2. This is a directed forest of intrees since we assigned at most one parent to each $(\mathcal{S}, \varphi)$-component since the unweighted version of the edges are part of a tree, namely the mountain connection tree, and since because of Condition 3, for each pair C_1 and C_2 of $(\mathcal{S}, \varphi)$-components, there exists at most one of the edges (C_1, C_2) and (C_2, C_1). See also Fig. 10 and 11. Then, for each tree $\tilde{T}$ of $\tilde{F}$ run the following substeps: Let $H \in \mathcal{H}$ be the crest contained in the $(\mathcal{S}, \varphi)$-component C being the root of $\tilde{T}$. Put Y_C into $\mathcal{Y}$, determine the inner graph of Y_C and all crests in $\mathcal{H}$ that are part of the inner graph, and remove all nodes from $\tilde{T}$ that are $(\mathcal{S}, \varphi)$-components containing those crests. Then recursively proceed in the same way for each remaining intree being a subgraph of $\tilde{T}$.

Step 4: Return the set $\mathcal{Y}$.

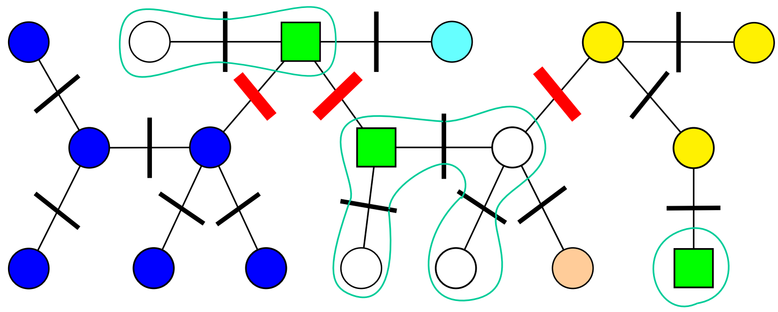

Figure 10: A mountain connection tree T of the mountain structure $(G, \varphi, c, \mathcal{H}, \mathcal{S})$ and the crest separators in $\mathcal{S}$; the latter denoted by straight lines crossing the edges of the tree. Thick straight lines denote crest separators that are pseudo shortcut free. Squared vertices represent pseudo shortcut free components. The curves denote coast separators constructed for the crests in $\mathcal{H}$ that are in the pseudo shortcut free (S, φ)-components. If such a curve in the figure encloses some nodes representing an (S, φ)-component C, this should mean that the corresponding coast separator encloses the crest in $\mathcal{H}$ contained in C. The non-white round vertices are the nodes of the forest $\tilde{F}$.

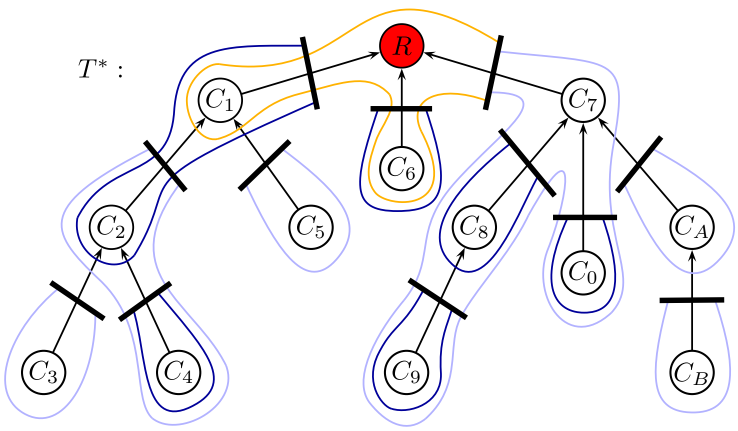

Figure 11: A bold straight line represents a crest separator and a curve together with a straight line represents a cycle: either a composed cycle or an induced cycle (by the edges of the essential boundary of a crest separator). If such a cycle encloses a node being an $(\mathcal{S}, \varphi)$-component C, this should mean that the cycle encloses at least the crest in $\mathcal{H}$ that is contained in C. Running the algorithm from above, the cycles $Y_R, Y_{C_2}, Y_{C_3}, Y_{C_5}, Y_{C_7}, Y_{C_A}$, and Y_{C_B} corresponding to the curves starting in the $(\mathcal{S}, \varphi)$-components R, C_2, C_3, C_5, C_7, C_A, and C_B, respectively, are added to $\mathcal{Y}$ in Step 3. Afterwards, each node of $\tilde{T}$ is enclosed by a cycle in $\mathcal{Y}$.

Analysis of the Algorithm. We now prove properties of the coast separators constructed by the algorithm above.

Observation 5.7. *Each crest in $\mathcal{H}$ is enclosed by a coast separator of $\mathcal{Y}$.*

Lemma 5.8. *For all crests $H \in \mathcal{H}$ that are considered in Step 1, there exists a minimal coast separator in the plane graph (C', ψ) constructed for H.*

Proof. Let C be the $(\mathcal{S}', \varphi)$-component that contains H. Since the vertices of the coast have upper height at most $c_{\max}$ and since H has lower height at least $k + 2c_{\max} \geq k + c_{\max} + 1$, by Theorem 2.4, there is a coast separator for H in G of weighted length at most k. Thus, there is also such a coast separator Y that is minimal. Since C contains vertices of the coast of G, Y and C are not vertex disjoint. Moreover, by Lemma 4.4, Y can not cross a crest separator in $\mathcal{S}'$, i.e, Y is completely contained in C. However, Y may contain vertices of the boundary of C (for some appropriate definition of the boundary), which is the reason for running the algorithm of Lemma 5.6 on the plane graph (C', ψ). □

Lemma 5.9. *For a coast separator $Y \in \mathcal{Y}$, let $\mathcal{C}$ be the (S, φ)-components with the crests in $\mathcal{H}$ that are enclosed by Y. Then, $\mathcal{C}$ induces a connected subtree of the mountain connection tree of $(G, \varphi, c, \mathcal{H}, \mathcal{S})$.*

Proof. The lemma clearly holds if Y is the essential boundary of a crest separator with an interior lowpoint. So from now on, we only consider the remaining kinds of coast separators.

Let Y be a minimum coast separator constructed for a crest H in Step 1. By Lemma 4.4, whenever Y crosses a crest separator X, the part P' of Y contained in the (X, φ)-component not containing H is a pseudo shortcut for a crest-separator path of X contained in the inner graph of Y.

Analogously, let P be a nice pseudo shortcut taken for the construction of a composed cycle Y as a coast separator for a crest H' in Step 2. Since P is nice, whenever P crosses a crest separator X, the part P' of Y contained in the (X, φ)-component not containing H' is a pseudo shortcut.

The lemma now follows from Lemma 5.2 and the fact that each coast separator enclosing two crests in $\mathcal{H}$ contained in different $(\mathcal{S}, \varphi)$-components C_1 and C_2 must cross all crest separators separating two consecutive $(\mathcal{S}, \varphi)$-components on the path from C_1 to C_2 in the mountain connection tree. □

We next want to show that no crest $H \in \mathcal{H}$ is enclosed by more than one coast separator in $\mathcal{Y}$. We start with two auxiliary lemmas.

Lemma 5.10. *Let C^* be an $(\mathcal{S}, \varphi)$-component with a crest $H^* \in \mathcal{H}$. If Y_{C^*} is added to $\mathcal{Y}$ in Step 3 and if Y_{C^*} encloses a crest $H' \in \mathcal{H}$ in an $(\mathcal{S}, \varphi)$-component C', then H' is neither part of a pseudo shortcut free $(\mathcal{S}, \varphi)$-component nor enclosed by a coast separator constructed for a crest in Step 1.*

Proof. The fact that Y_{C^*} encloses H' implies that H' can not be part of a pseudo shortcut free $(\mathcal{S}, \varphi)$-component. Assume now that a coast separator Y_C constructed for a crest $H \in \mathcal{H}$ in a pseudo shortcut free component C encloses H' and assume w.l.o.g. that H' is chosen as a crest enclosed by Y_{C^*} with this property such that the distance between the $(\mathcal{S}, \varphi)$-components C' and C^* is as small as possible. Then, $H^* \neq H'$ since, otherwise, Y_{C^*} would not be added to $\mathcal{Y}$ in Step 3. Moreover, Y_{C^*} can not be the essential boundary of a crest separator with an interior lowpoint. Otherwise, for the subtree T' of T with root C^* that contains C', all its $(\mathcal{S}, \varphi)$-components are enclosed by a crest separator with an interior lowpoint. Since Y_C enclosing H' but not H^* implies that C is in T', we then obtain a contradiction to the fact that C is pseudo shortcut free. Thus, Y_{C^*} is a cycle consisting of some nice pseudo shortcut P' with its crest-separator path. See Fig. 12. Let X be the crest separator with

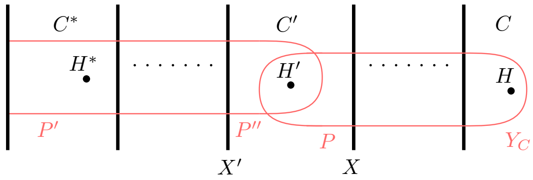

Figure 12: Coast separators constructed in Step 1 and in Step 2 enclosing a crest H'.

a top edge in C' disconnecting H and H', and let D be the (X, φ)-component containing C'. Y_C as a minimal coast separator has a D-pseudo shortcut P of X for a crest-separator path of X contained in the inner graph of Y_C as a subpath (Lemma 4.4). Moreover, let X' be the crest separator with a top edge in C' disconnecting H^* and H', and let D' be the (X', φ)-component containing C'. Note that $X' \neq X$ since we have chosen H' in such a way that C' and C^* have minimal distance. Since P' is nice, a subpath P'' of P' is a D-pseudo shortcut for X' for a crest-separator path of X' contained in the inner graph of Y_{C^*}. The existence of P and P'' is a contradiction to Lemma 5.3. ☐

Lemma 5.11. *Let C and C^* be two $(\mathcal{S}, \varphi)$-components whose crests in $\mathcal{H}$ are not enclosed by a coast separator constructed in Step 1. If the coast separator Y_{C^*} for the crest $H^* \in \mathcal{H}$ in C^* encloses the crest $H \in \mathcal{H}$ in C, then the directed forest $\tilde{F}$ constructed by our algorithm has a directed path from C to C^*.*

Proof. Assume that Y_{C^*} encloses H. Let $C = C_0, C_1, \ldots, C_r = C^*$ be the path in the mountain connection tree connecting C and C^*. For $i \in \{1, \ldots, r\}$, let $X_i \in \mathcal{S}$ be the crest separator with a top edge part of both C_{i-1} and C_i. The situation is sketched in Fig. 9, but this time, ignore the shown pseudo shortcuts. Then, the crest $H_i \in \mathcal{H}$ in C_i $(i = \{0, \ldots, r\})$ is enclosed by Y_{C^*} (Lemma 5.9), but are neither part of a pseudo shortcut free $(\mathcal{S}, \varphi)$-component nor enclosed by a coast separator constructed in Step 1 (Lemma 5.10). Thus, Condition 1 in Step 2 holds for all edges (C_i, C_{i+1}) $(i = \{0, \ldots, r-1\})$. Let D be the $(\mathcal{S}, \varphi)$-component of X_{C^*} containing C^*, where X_{C^*} is the crest separator defined for C^* in Step 2.

We first consider the case where Y_{C^*} is defined in Step 2.A, i.e., Y_{C^*} is the essential boundary of the crest separator X_{C^*}. Then, X_{i+1} $(i \in \{0, \ldots, r-1\})$ has an interior lowpoint and encloses C_i, but not C_{i+1} (Condition 2 holds). Moreover, the D^*-pseudo shortcut set for X_{i+1} is empty by the definition of pseudo shortcuts, where D^* is the (X_{i+1}, φ)-component containing C_{i+1} (Condition 3 holds). As a consequence, our algorithm adds edges (C_i, C_{i+1}) to $\tilde{F}$ for all $i = \{0, \ldots, r-1\}$.

Next, we consider the case where Y_{C^*} is defined in Step 2.B, i.e., Y_{C^*} is the cycle consisting of some nice D-pseudo shortcut P' with its crest-separator path of a crest separator X_{r+1} with a top edge in C_r. Note also that either C_i ($i = \{1, \dots, r\}$) is not enclosed by X_i or X_i has no interior lowpoint (Condition 2 holds) since, otherwise, Y_{C^*} would be also enclosed by X_i and X_r and hence defined in Step 2.A. For each crest separator X_{i+1} ($i = \{1, \dots, r\}$), a subpath of P of P' is a D'-pseudo shortcut where D' is the (X_{i+1}, φ)-component containing C_i. Since P' is a pseudo shortcut in $\mathcal{Z}$ and since $\mathcal{Z}$ is consistent, the subpath of P' starting and ending on X_{i+1} is a pseudo shortcut in $\mathcal{Z}$. Since $\mathcal{Z}$ is also non-overlapping, X_{i+1} is the only crest separator with a top edge in C_i that has pseudo shortcuts, i.e., Condition 3 is satisfied for the edge (C_{i-1}, C_i), i.e., our algorithm adds edges (C_i, C_{i+1}) to $\tilde{F}$ for all $i = \{0, \dots, r-1\}$. □

Lemma 5.12. *No crest $H \in \mathcal{H}$ is enclosed by more than one coast separator in $\mathcal{Y}$.*

Proof. A coast separator constructed in Step 1 can not enclose a crest in $\mathcal{H}$ that is also enclosed by another coast separator added to $\mathcal{Y}$ in Step 1 (Lemma 5.5) or in Step 3 (Lemma 5.10). By Lemma 5.11, a crest in $\mathcal{H}$ can not be enclosed by two coast separators in $\mathcal{Y}$ that are constructed for $(\mathcal{S}, \varphi)$-components part of two different trees of $\tilde{F}$.

By our choice of directing the edges of E' in Step 2, each coast separator Y_C for a crest $H \in \mathcal{H}$ contained in an $(\mathcal{S}, \varphi)$-component C of a tree $\tilde{T}$ of $\tilde{F}$ can only enclose (S, φ)-components below C in $\tilde{T}$. By Lemma 5.9 the $(\mathcal{S}, \varphi)$-components of $\tilde{T}$ with the crests in $\mathcal{H}$ that are enclosed by Y_C induce a connected subtree of $\tilde{T}$. Therefore, two coast separators constructed in Step 2 for the same tree $\tilde{T}$ and added to $\tilde{\mathcal{Y}}$ in Step 3 cannot enclose the same crest. □

Finally, we analyze the running time. Take $G = (V, E)$. Recall that G is $O(k)$-weighted outerplanar and that $(G, \varphi, c, \mathcal{H}, \mathcal{S})$ is a good mountain structure for (G, φ, c).

Lemma 5.13. *The algorithm for computing the set $\mathcal{Y}$ runs in $O(|\mathcal{H}|k^3 + |V|k)$ time.*

Proof. We can construct a D-pseudo shortcut set for all crest separators $X \in \mathcal{S}$ and all (X, φ)-components D in $O(|\mathcal{H}|k^3 + |V|k)$ time (Lemma 4.9). Clearly within the same time, for each (X, φ)-components D we can decide whether the D-pseudo shortcut set is empty and whether the essential boundary of X can be used as a coast separator, i.e., whether X has an interior lowpoint. Hence, again within in the same time we can determine the subset $\mathcal{S}'$ of all pseudo shortcut free crest separators in $\mathcal{S}$, all pseudo shortcut free $(\mathcal{S}, \varphi)$-components as well as all $(\mathcal{S}', \varphi)$-components. A minimal coast separator for a crest $H \in \mathcal{H}$ of a pseudo shortcut free $(\mathcal{S}, \varphi)$-component contained in an $(\mathcal{S}', \varphi)$-component C can be computed in $O(n'k)$ time (Lemma 5.6) where n' is the number of vertices of C. In the same time, we can determine the inner graph of Y_C, all crests in $\mathcal{H}$ that are part of the inner graph, and the set of $(\mathcal{S}, \varphi)$-components containing them. Therefore the running time of Step 1 is bounded by $O(|\mathcal{H}|k^3 + |V|k)$.

To test whether an (X, φ)-component has an interior lowpoint can be done in $O(k)$ time for each of the $O(|V|)$ crest separators. Since the set $\mathcal{Z}$ can be constructed in $O(|\mathcal{H}|k^2)$ time and since $\tilde{F}$ consists of $O(|V|)$ nodes, it is easy to run Step 2 in $O(|V|k + |\mathcal{H}|k^2)$ time.

The construction of the inner graphs of all coast separators Y_C added to $\mathcal{Y}$ in Step 3 as well as the removal of all (S, φ)-components C with crests in $\mathcal{H}$ that are part of such an inner graph can be done by a depth-first search in the inner graph of Y_C and the running time can be bounded by the number of edges of the (S,φ)-components removed. Since each removal of crest separator removes a disjoint set of (S,φ)-components (Lemma 5.12), the running time of Step 3 is bounded by $O(|V|k)$. $\square$

Note that the coast separators constructed in Step 1 have weighted size at most k, whereas the coast separators taken in Step 3 can have a weighted size of at most $3k + 4c_{\max} - 5$ since the down paths of each crest separator $X \in \mathcal{S}$ can have a weighted length of at most $k + 2c_{\max} - 1$, since a pseudo shortcut does not start and end with a vertex of the coast (which reduces the length of a subpath of a down path that can be part of a coast separator by 1), and since the pseudo shortcuts that we use have weighted length $\leq k - 1$.

Let n be the number of vertices of G^+. Since each crest $H \in \mathcal{H}$ of G is obtained from a merge of at least $\lceil k/c_{\max} \rceil$ vertices in G^+, $|\mathcal{H}| \leq n \cdot \min\{1, c_{\max}/k\}$.

For each $H \in \mathcal{H}$, let us choose one crest of height $2k + 2c_{\max}$ in (G^+, φ^+, c^+) that is merged into H. Define $\mathcal{H}^+$ as the set of chosen crests. Note that with $(G, \varphi, c, \mathcal{H}, \mathcal{S})$ being a good mountain structure $(G^+, \varphi^+, c^+, \mathcal{H}^+, \mathcal{S})$ must also be a good mountain structure. Then, the next corollary summarizes the results of the current section, where the function m simply maps each coast separator Y to the $(\mathcal{S}, \varphi)$-components whose crests are enclosed by Y.

Corollary 5.14. *Assume that we are given the integer k, the $(2k + 2c_{\max})$-weighted-outerplanar graph (G^+, φ^+, c^+), and the set $\mathcal{H}^+$ of crests of height $2k + 2c_{\max}$ as defined before. Then, in $O(nk^2 \min\{k, c_{\max}\})$ time, one can construct a good mountain structure $\mathcal{M} = (G^+, \varphi^+, c^+, \mathcal{H}^+, \mathcal{S})$, the mountain connection tree T for $\mathcal{M}$, a set $\mathcal{Y}$ of coast separators in G^+, for each $Y \in \mathcal{Y}$, the inner graph I of Y and the corresponding embedding $\varphi|_I$ such that the properties below hold.*

(i) *For each crest H of height $2k + 2c_{\max}$ in G^+—with H not necessarily being contained in $\mathcal{H}^+$—there is exactly one coast separator $Y \in \mathcal{Y}$ for H of weighted size at most $3k + 4c_{\max} - 5$.*

(ii) *For each pair of crests of height $2k + 2c_{\max}$ in G^+, the crests are either part of the inner graph of one $Y \in \mathcal{Y}$ or there is a crest separator $X \in \mathcal{S}$ strongly going between the crests.*

(iii) *There is a function m mapping each $Y \in \mathcal{Y}$ to a non-empty set of $(\mathcal{S}, \varphi^+)$-components such that*

- *the elements of $m(Y)$ considered as nodes of T induce a connected subgraph of T,*
- *the subgraph of G^+ obtained from the union of all $(\mathcal{S}, \varphi^+)$-components in $m(Y)$ contains the inner graph I_Y of Y as a subgraph,*
- *the set $m(Y)$ does not have any $(\mathcal{S}, \varphi^+)$-component as an element that is also an element in $m(Y')$ for a coast separator $Y' \in \mathcal{Y}$ with $Y' \neq Y$.*

6 A Tree Decompositions for the Components

We first describe an algorithm for constructing a tree decomposition of width $3\ell - 1$ for a weighted almost triangulated ℓ-outerplanar graph. Then we modify the algorithm such that, given a weighted graph (G, φ, c), $\mathcal{S} \subseteq \mathcal{S}(G, \varphi, c)$, and an $(\mathcal{S}, \varphi)$-component C, it constructs a tree decomposition for $\text{ext}(C, \mathcal{S})$ of width at most $3\ell - 1$ with the following property: For each crest separator $X \in \mathcal{S}$ with a top edge in C, there is a bag containing all vertices of X.

It is easier to construct such a tree decomposition if the following *neighborhood property* holds.

(N) For each vertex v, there is at most one vertex u with $u{\downarrow} = v$.

Starting with an almost triangulated weighted ℓ-outerplanar graph $(\tilde{G}, \tilde{\varphi}, \tilde{c})$ as an intermediate goal, we want to transform it in such a way into an ℓ-outerplanar weighted plane graph (G, φ, c) with the neighborhood property (N) that a tree decomposition for $\tilde{G}$ can be easily obtained from a tree decomposition for G. The idea is to compute G as a *reverse minor* of $\tilde{G}$, i.e., $\tilde{G}$ can be obtained from G by iteratively merging adjacent vertices and/or by removing vertices.

In order to guarantee property (N), we start our transformation by splitting each vertex v into a path as it is sketched in Fig. 13. More precisely, iterate over the vertices v with non-increasing lower height. We consider only the more interesting case where $h^-_\varphi(v) \geq 2$. Let $u = v{\downarrow}$. Let $\{v, u\}, \{v, u_1\}, \ldots, \{v, u_d\}$ be the edges incident to v in the clockwise order in which they appear around v. Then v is replaced by a series of vertices $v_1, \ldots, v_d$ with v_1 being incident to u_1 and v_2, whereas v_d is incident to v_{d-1} and u_d, and for $i \in \{2, \ldots, d-1\}$, v_i is incident to v_{i-1}, v_{i+1}, and u_i. We finally connect all vertices $v_1, \ldots, v_d$ to u by an edge. The vertices $v_1, \ldots, v_d$ are then called the *copies of* v. We finally define the weight of the copies of a vertex v as the weight of v. Let (G, φ, c) be the weighted plane graph obtained. Thus, the copies of v have the same height interval as v itself, and G is weighted ℓ-outerplanar. Note that strictly speaking no down vertices and down edges are defined in G since down vertices where only defined for almost triangulated graphs. Thus, if we refer to a down edge $\{u_i, v_j\}$ in G for vertices u_i and v_j being copies of vertices u and v in $\tilde{G}$, we mean that $\{u, v\}$ is a down edge in the original graph. By our construction every vertex u_i in G being a copy of a vertex u in the original graph is then incident to at most one down edge with another endpoint of larger height—so

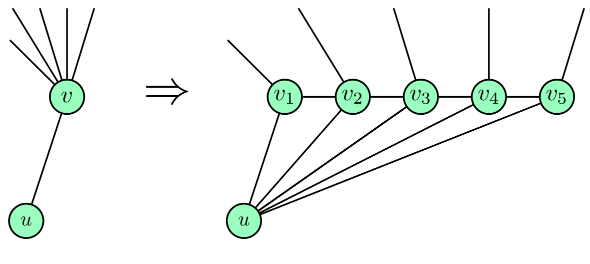

Figure 13: The replacement of a vertex v. Let $u = v{\downarrow}$. The vertices $v_1, \ldots, v_5$ are the copies of v.

that property (N) holds—and exactly one down edge with another endpoint of lower height. The endpoint of the latter edge is then defined to be the down vertex $u_i\!\downarrow$ of u_i, which then is also a copy of $u\!\downarrow$. Similarly, the down path of u_i, defined similarly as for almost triangulated graphs, then consists of copies of the vertices of the down path from u. However, different copies u_i and u_j in G of the same vertex u in $\tilde{G}$ have vertex-disjoint down paths in G.

We next want to bound the number of edges of G with respect to the number of edges of G. Note that, for each edge incident to a vertex v, but not being the down edge of v, the splitting of v introduces a new copy v_i of v and up to two additional edges into G connecting v_i to the down vertex of v_i and to a previous copy v_{i-1} of v_i, respectively. One of them namely, the down edge of v_i will recursively cause further splittings along the down path of v_i so that altogether one edge of the original graph will introduce up to 2ℓ new edges. Consequently, if $\tilde{n}$ is the number of vertices of the original graph, G has $O(\ell\tilde{n})$ vertices and edges.

Lemma 6.1. *For each given weighted almost triangulated ℓ-outerplanar graph $(\tilde{G}, \tilde{\varphi}, \tilde{c})$ with $\tilde{n}$ vertices, a weighted ℓ-outerplanar graph (G, φ, c) can be found in $O(\tilde{n}\ell)$ time such that*

1. *G is a reverse minor of $\tilde{G}$,*
2. *G satisfies the neighborhood property,*
3. *for each each edge $\{u, v\}$, there is an edge connecting a copy u_i of u and a copy v_i of v in G, and*
4. *the down paths of u_i and v_i in G can be obtained from the down paths of u and v, respectively, in $\tilde{G}$ by replacing the vertices of the down paths in $\tilde{G}$ by copies of its vertices in G.*

We now describe an algorithm to compute a tree decomposition for the weighted ℓ-outerplanar graph (G, φ, c) constructed above. Let n be the number of vertices of G. W.l.o.g., G is biconnected; otherwise, one can compute a tree decomposition for each biconnected component independently and finally connect them. Take $S = V \setminus \{u\!\downarrow \mid u \in V \text{ and } h^-_\varphi(u) \geq 2\}$ as illustrated in Fig. 14.

As a first step of our computation, we merge each down path starting at a vertex v of S to one vertex v^* and define the weight of v^* as $h^+_\varphi(v)$. Let (G^+, φ^+, c^+) be the graph obtained. Note that $h^+_\varphi(v) = h^+_{\varphi^+}(v^*)$, and G^+ is weighted ℓ-outerplanar. Since all vertices of (G^+, φ^+, c^+) are incident to the outer face, the unweighted version (G^-, φ^-) of (G^+, φ^+, c^+) is outerplanar and we find a tree decomposition (T, B) for G^- of width 2 as shown by Bodlaender [4], i.e., in each bag we have at most 3 vertices. Since each vertex in G^+ has a weight of at most ℓ, (T, B) is a tree decomposition for (G^+, φ^+, c^+) with bags of size 3ℓ; in other words, (T, B) as a tree decomposition for (G^+, φ^+, c^+) has a width of at most $3\ell - 1$. By replacing each vertex v^* by the down path of v, we obtain a tree decomposition for (G, φ, c) of width $3\ell - 1$ in $O(n\ell)$ time.

As we show in the following, the algorithm from above can be slightly modified to prove the next lemma.

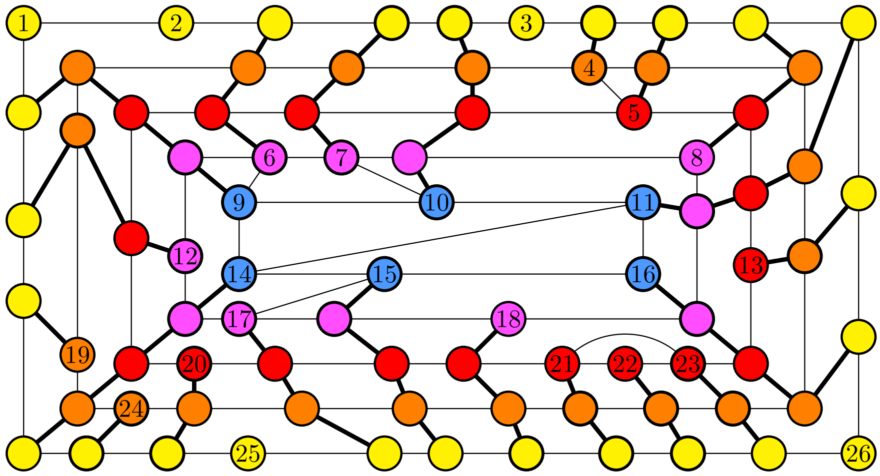

Figure 14: A plane unweighted graph with the neighborhood property. Down edges are drawn bold. Vertices in the set S are numbered.

Lemma 6.2. *Let (G, φ, c) be an almost triangulated weighted ℓ-outerplanar biconnected graph, and $M = (G, \varphi, c, \mathcal{S})$ be a good mountain structure of (G, φ, c). Take C as an $(\mathcal{S}, \varphi)$-component, and $C' = \mathrm{ext}(C, \mathcal{S})$. Then one can construct a tree decomposition (T, B) of width $3\ell - 1$ for C' that, for each crest separator $X \in \mathcal{S}$ with a top edge in C, has a bag containing all vertices of X. Moreover, given C' as well as the height intervals $[h^-_\varphi(v), h^+_\varphi(v)]$ for each vertex v in C', the construction can be done in $O(n'\ell^2)$ time, where n' is the number of vertices in C'.*

Proof. For the time being, let us assume that no crest separator with a top edge in C' has a lowpoint. At the end of our proof we show how to handle crest separators with a lowpoint. The heights of the vertices in C' with respect to $\psi = \varphi|_{C'}$ may differ from the heights of these vertices with respect to φ. In order to avoid this, we insert additional edges into C'. More precisely, for each crest separator $X = (P_1, P_2)$ with a top edge in C, height $r \in \mathbb{N}$, and vertices $u_1, \ldots, u_p$ of P_1 and $v_1, \ldots, v_q$ of P_2 in the order of their appearance on these paths, and for $i \in \{1, \ldots, r\}$, define $u(i)$ as the vertex in $\{u_1, \ldots, u_p\}$ whose height interval contains i. Analogously, we define $v(i)$. We extend C' by the inserting edges $\{u(i), v(i)\}$ for all i with $1 \leq i \leq r$ into the outer face of C'. (To avoid multiple edges, instead of adding an edge, one can also add a path consisting of 2 edges.) By applying the changes above to all crest separators, this results in a weighted plane graph $(\tilde{C}, \tilde{\varphi}, \tilde{c})$ such that $h_{\tilde{\varphi}}(v) = h_\varphi(v)$ holds for all vertices v of C'. We next transform $(\tilde{C}, \tilde{\varphi}, \tilde{c})$ into a graph $(\hat{C}, \hat{\varphi}, \hat{c})$ with the neighbor hood property (N) by applying Lemma 6.1 and construct a tree decomposition of width $3\ell - 1$ for this new graph as it is described after Lemma 6.1.

Let us now analyze what happens to a crest separator $X = (P_1, P_2)$. By Lemma 6.1, the top edge of X is replaced by an edge in $(\hat{C}, \hat{\varphi}, \hat{c})$ connecting copies of the original endpoints. Moreover, the down paths of these endpoint copies consist of copies of the vertices of P_1 and P_2. After merging each down path in $(\hat{C}, \hat{\varphi}, \hat{c})$ to one vertex—let G' be the graph obtained—there is an edge connecting the two vertices that are introduced for P_1 and P_2. Thus, the tree decomposition for G' has a bag containing the two vertices. As a consequence, we obtain a tree decomposition for $\hat{C}$ that contains a bag with copies of all vertices of P_1 and P_2. Since $\hat{C}$ is a reverse minor of $\tilde{C}$, we obtain a tree decomposition for $\tilde{C}$ from a tree decomposition for $\hat{C}$ of at most the same width in a standard way by replacing splitted vertices and edges by the original vertices and edges in $\tilde{C}$ thereby replacing the copies of the vertices of P_1 and P_2 by the original vertices of P_1 and P_2. After removing the vertices $u(i)$ and $v(i)$ outside C', we obtain a tree decomposition of width at most $3\ell - 1$ for C' that, for each crest separator $X \in \mathcal{S}$, has a bag containing all vertices of X.

We next show how to exclude crest separators with lowpoints by modifying the given graphs. For simplicity, our modifications described below do not result in an almost triangulated graph; however the graph can be easily transformed into an almost triangulated graph by adding into each inner face with more than three edges on its boundary edges incident to one vertex of smallest upper height on the boundary, which does not change any height interval. Let us first consider a crest separator $X \in \mathcal{S}$ with a top edge in C that encloses C. Note that in this case every crest separator with a top edge in C contains the lowpoint v of X. Let $i = h^+_\varphi(v)$. Then we remove all vertices u with $h^+_\varphi(u) \leq i$ from G, from C, and from the crest separators contained in $\mathcal{S}$, and additionally we remove from $\mathcal{S}$ all crest separators that afterwards have no vertices anymore. For all vertices u with $h^-_\varphi(u) \leq i + 1 < h^+_\varphi(u)$, we define $c'(u) = h^+_\varphi(u) - i$. For the remaining vertices u, we define $c'(u) = c(u)$. We so obtain a new good mountain structure $(G', \varphi', c', \mathcal{S}')$ from $(G, \varphi, c, \mathcal{S})$, where φ' is a weighted $(\ell - i)$-outerplanar embedding. More precisely, if the graph G' after removing all vertices of upper height at most i is not biconnected, we take a good mountain structure for each biconnected component. The good news is that, for each crest separator $X' = (P'_1, P'_2)$ of $\mathcal{S}$ with a top edge in C, the subpaths P^*_1 and P^*_2 of P'_1 and P'_2, respectively, ending immediately before v are contained in the same biconnected component. To see this, we distinguish between two cases. If we have $h^-_\varphi(u) = h^-_\varphi(v) = i + 1$ for both top vertices u and v of X', P^*_1 and P^*_2 consist only of these two vertices. Since these vertices are connected by an edge, they must be part of the same biconnected component. Otherwise we must have $h^-_\varphi(u) > i + 1$ for at least one top vertex u of X'. Hence we can conclude that there is a cycle of vertices with their height intervals containing $i+1$ that encloses u and all vertices of P^*_1 and P^*_2 of lower height at least $i + 2$ and that contains the vertices of P^*_1 and P^*_2 with lower height $i + 1$. The inner graph of this cycle is biconnected and since it contains the cycle itself it must contain all vertices of P^*_1 and P^*_2. Therefore, we separately construct a tree decomposition for each biconnected component of G' and then connect the tree decompositions of each biconnected component. After the modifications—in particular, the removal of v—no crest separator in $\mathcal{S}'$ encloses the new $(\mathcal{S}', \varphi')$-component C^* obtained from C (Lemma 5.1). The idea is then to use the construction as described for crest separators with no lowpoints to construct a tree decomposition of width $3(\ell - i) - 1$ for each biconnected component such that, for each crest separator $X' \in \mathcal{S}$ with a top edge in C^*, it has a bag containing all vertices u of X' with $h^+_\varphi(u) > i$. Since C is enclosed by X, the remaining vertices of X' are all part of the down path of v in (G, φ, c). We can simply add the vertices of the down path of v into all bags of the tree decomposition to obtain the desired tree decomposition for C' of width $3\ell - 1$.

We next describe how to handle a crest separator $X = (P_1, P_2)$ with a lowpoint that does not enclose C. Define $u_1, \ldots, u_p$ and $v_1, \ldots, v_q$ as the vertices of P_1 and P_2 in the order in which they appear on P_1 and P_2, respectively. Assume that $u_j = v_{j'}$ is the lowpoint of X. Since we want to use a similar construction as for crest separators without any lowpoints, the idea is, for all $i \geq j$, to split u_i in $\tilde{C}$ into two different vertices u'_i and v'_i with the same weight than u_i. Moreover, $u'_j, \ldots, u'_p$ should be connected to the neighbors of P on one side and the vertices $v'_j, \ldots, v'_p$ to the neighbors of P on the other side of P. Additionally, each edge $\{u_i, u_{i+1}\}$ ($i \in \{j, \ldots, p-1\}$) is replaced by the two edges $\{u'_i, u'_{i+1}\}$ and $\{v'_i, v'_{i+1}\}$. Observe that X is *split* into a crest separator without a lowpoint that consists of the two paths $u_1, \ldots, u_{j-1}, u'_j, \ldots, u'_p$ and $v_1, \ldots, v_{j'-1}, v'_j, \ldots, v'_p$. Fig. 15 sketches the construction. If a vertex is a lowpoint of several crest separators, note that we have to do the splitting for each of the crests separators. It is not hard to see that the splitting process for different crest separators does not lead to a non-planar embedding as long as there is no pair of crest separators X_1 and X_2 in $\mathcal{S}$ with top edges in C for which the vertices of the essential boundary of X_1 are enclosed by X_2. Then C' must be enclosed by one of X_1 and X_2, but we have already excluded this case in the last paragraph.

Concerning the running time, it is dominated by the construction of a tree decomposition for $\hat{C}$. This construction takes $O(\hat{n}\ell)$ time, where $\hat{n}$ is the number of vertices of $\hat{C}$. Since the replacement of $\tilde{C}$ by $\hat{C}$ may increase the number of vertices by a factor of $O(\ell)$, the whole running time is $O(n'\ell^2)$. $\square$

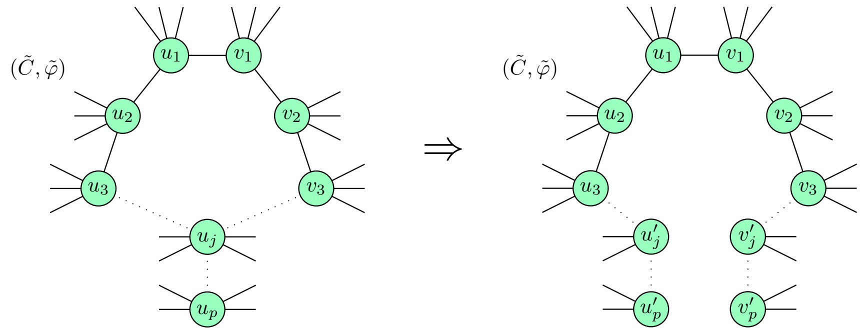

Figure 15: The replacement of a crest separator with a lowpoint by one with no lowpoint.

7 The Main Algorithm

In this section we describe our main algorithm. As mentioned in Section 2, we assume that we are given an almost triangulated weighted graph (G, φ, c) with weighted treewidth k. In the case that no embedding φ is given, we can compute an arbitrary planar embedding in linear time [15]. Recall that $c_{\max}$ denotes the maximum weight over all vertices. Let $\ell = 2k + 2c_{\max}$. Our algorithm starts with cutting off all maximal connected subsets of vertices of lower height at least $\ell - 1$ by coast separators of size $O(k)$. More precisely, to find such coast separators, in a first substep we merge each maximal connected set M of vertices of lower height at least $\ell - 1$ to one vertex v_M and define $c(v_M) = 1$. Therefore the weighted graph (G', φ', c) obtained is weighted ℓ-outerplanar. Given a vertex of the coast, this can be done in a time linear in the number of vertices with a lower height of at most $\ell - 1$. G' is an almost triangulated, biconnected graph since this is true for G. We then can use Corollary 5.14 to construct, for an appropriate subset $\mathcal{H}$ of the vertices of height ℓ, a good mountain structure $(G', \varphi', c, \mathcal{H}, \mathcal{S})$ as well as to find a set of coast separators $\mathcal{Y}$ and a function m that maps the coast separators to $(\mathcal{S}, \varphi')$-components such that the following properties of the lemma hold.

- Each crest of upper height exactly ℓ is enclosed by a coast separator $Y \in \mathcal{Y}$ of weighted size at most $3k + 4c_{\max} - 5$.
- The $(\mathcal{S}, \varphi')$-components in $m(Y)$ induce a connected subgraph of the mountain connection tree.
- For each pair of crests of height ℓ, the crests are either part of the inner graph of one $Y \in \mathcal{Y}$ or there is a crest separator $X \in \mathcal{S}$ strongly going between the crests.
- The inner graph of a coast separator $Y \in \mathcal{Y}$ is a subgraph of the graph obtained from the union of the $(\mathcal{S}, \varphi')$-components in $m(Y)$.
- For each $(\mathcal{S}, \varphi')$-component C, there is at most one $Y \in \mathcal{Y}$ with $C \in m(Y)$.

Since G' is weighted ℓ-outerplanar, we can apply Lemma 6.2 to each $(\mathcal{S}, \varphi')$-component C to compute a tree decomposition (T_C, B_C) of width at most $3\ell - 1$ for $\mathrm{ext}(C, \mathcal{S})$ such that, for each crest separator $X \in \mathcal{S}$ with a top edge in C, (T_C, B_C) has a node whose bag contains all vertices of X. This node is then connected to a node whose bag also contains all vertices of X and that is constructed for $\mathrm{ext}(C', \mathcal{S})$ with C' being the other $(\mathcal{S}, \varphi')$-component C' containing the top edge of X. Since the set of common vertices of $\mathrm{ext}(C, \mathcal{S})$ and $\mathrm{ext}(C', \mathcal{S})$ is a subset of the vertices of X, after also connecting nodes for all other crest separators in $\mathcal{S}$, we obtain a tree decomposition (T^*, B^*) for G'.

Let us next remove from $\mathcal{S}$ all crest separators whose top edge is contained in two $(\mathcal{S}, \varphi')$-components belonging to the same set $m(Y)$ for some $Y \in \mathcal{Y}$. Afterwards, for the new set $\mathcal{S}'$ of crest separators, each cycle $Y \in \mathcal{Y}$ is contained in one $(\mathcal{S}', \varphi')$-component.[1] For each $(\mathcal{S}', \varphi')$-component C', let us call the *flat component* of C' to be the subgraph of $\mathrm{ext}(C', \mathcal{S}')$ obtained by removing the vertices of the inner graph of the cycle $Y \in \mathcal{Y}$ with Y contained in C' if such a cycle Y exists. Otherwise, we define the *flat component* to be $\mathrm{ext}(C', \mathcal{S}')$, which then contains no vertex of upper height larger than ℓ. From the bags in (T^*, B^*), we then remove all vertices that do not belong to a flat component. Afterwards, for each cycle $Y \in \mathcal{Y}$ disconnecting the crests of an $(\mathcal{S}', \varphi')$-component C' from the coast, we put the vertices of Y into all bags of the tree decompositions (T_C, B_C) constructed as part of (T^*, B^*) for the (extended components of the) $(\mathcal{S}, \varphi')$-components C contained in C'. This allows us to connect one of these bags with a bag of a tree decomposition for the inner graph of Y. Indeed, for

[1] We now have found a set of crest separators and coast separators that guarantee (P1) - (P3) from page 8. The set $\mathcal{S}'$ is exactly the set of perfect crest separators.

each $Y \in \mathcal{Y}$, we recursively construct a tree decomposition (T_Y, B_Y) for the inner graph G_Y of Y with the vertices of Y being the coast of G_Y. Into all bags of (T_Y, B_Y) that are not constructed in further recursive calls, we also put the vertices of Y. At the end of the recursions, we obtain a tree decomposition for the whole graph. Note that each bag is of weighted size $O(k)$. More precisely, let us consider a recursive call that constructs a tree decomposition (T_Y, B_Y) for a cycle Y constructed in a previous step. Then the tree decomposition for the flat component considered in the current recursive call puts up to $3\ell - 1$ vertices into each bag. However, we also have to insert the vertices of Y and possibly the vertices of a cycle constructed in the current recursive call into the bags. Since each of these cycles consists of at most $3k+4c_{\max}-5$ vertices, each bag of the final tree decomposition of G contains at most $3\ell+6k+8c_{\max}-11 \leq 12k+14c_{\max}-11$ vertices. Recall that $c_{\max} \leq k$. As mentioned in Section 2, it is possible to replace a non-triangulated weighted graph H of weighted treewidth k by an almost triangulated weighted supergraph G of H and to run our algorithm from above on G such that we can obtain a tree decomposition for H of width $(12 + \epsilon)k + 14c_{\max} + O(1)$.

Concerning the running time, it is easy to see that each recursive call is dominated by the computation of the cycles being used as coast separators. This means that each recursive call runs in $O(\tilde{n}k^2 \min\{k, c_{\max}\})$ time, where $\tilde{n}$ is the number of vertices of the subgraph G' of G considered in this call. Some vertices part of one recursive call are cut off from the current graph and then are also considered in a further recursive call. However, since the coast separators contain no vertex of the coast, the coast is not part of any recursive call. Therefore, each vertex is considered in $O(k)$ recursive calls, and our algorithm finds a tree decomposition for G of width $O(k)$ in $O(|V|k^3 \min\{k, c_{\max}\})$ time. If we do not know k in advance, we can use a binary search to determine a tree decomposition for G of width $O(k)$ in $O(|V|k^3 \min\{k, c_{\max}\} \log k)$ time.

Theorem 7.1. *For a weighted planar graph (G, c) with n vertices and weighted treewidth k and any constant $\epsilon > 0$, a tree decomposition for G of weighted width $(12+\epsilon)k + 14c_{\max} + O(1)$ can be constructed in $O(nk^3 \min\{k, c_{\max}\} \log k)$ time, where $c_{\max}$ denotes the maximum weight of a vertex of G.*

Corollary 7.2. *For a planar graph G with n vertices and treewidth k and any constant $\epsilon > 0$, a tree decomposition for G of width $(12 + \epsilon)k + O(1)$ can be constructed in $O(nk^3 \log k)$ time.*

For a more efficient algorithm, we replace ℓ by $3k+2c_{\max}$. When considering a weighted ℓ-outerplanar graph (G, φ) in one recursive call of the algorihm, we remove all vertices of upper height at most k, reduce the weight of each vertex v with upper height at least $k+1$ and lower height smaller than $k+1$ by k, and search for a coast separator in the resulting weighted $(2k + 2c_{\max})$-outerplanar graph as in our original algorithm. Since now the lower and upper heights of each vertex considered in two consecutive recursive steps differ by at least k, every vertex is now considered in at most $O(1)$ recursive calls. However, we now have to construct a tree decomposition for a weighted $(3k + 2c_{\max})$-outerplanar flat component in each recursive call. Thus, we now construct a tree decomposition with $3\ell + 6k + 8c_{\max} - 11 = 15k + 14c_{\max} - 11$ vertices per bag if G is almost triangulated, and $(15 + \epsilon)k + 14c_{\max} + O(1)$ vertices per bag otherwise.

Theorem 7.3. *For a weighted planar graph (G, c) with n vertices and weighted treewidth k and any constant $\epsilon > 0$, a tree decomposition for G of weighted width $(15+\epsilon)k + 14c_{\max} + O(1)$ can be constructed in $O(nk^2 \min\{k, c_{\max}\} \log k)$ time, where $c_{\max}$ denotes the maximum weight of a vertex of G.*

Corollary 7.4. *For a planar graph G with n vertices and treewidth k and any constant $\epsilon > 0$, a tree decomposition for G of width $(15 + \epsilon)k + O(1)$ can be constructed in $O(nk^2 \log k)$ time.*

It is also interesting to compute a grid minor if a graph has no tree decomposition of size $O(k)$. We can do so, for an unweighted planar graph G', if we abstain from multiplying the weights of the vertices of G' by a factor x during the transformation of G' into an almost triangulated graph G by inserting a vertex into each face and connecting it to all vertices on the boundary. Kloks et al. [18, Theorem 2] showed that the treewidth of G can be bounded by $k = 4k' + 1$ where k' is the treewidth of G'. If, afterwards, the algorithm fails to construct a tree decomposition for G of unweighted treewidth k, then this can only happen when we search for a separator with Theorem 2.4. In this case, we have k internally-vertex-disjoint paths $\mathcal{P}$ that all start and end at two vertices whose heights differ by more than k, i.e., each of these paths "crosses" $k-1$ cycles. If we cut the cycles all between two path of $\mathcal{P}$ and remove the endpoints of the paths, we get a $k \times (k-1)$ minor in the almost triangulated version of G. If we remove every second path of $\mathcal{P}$ and every second cycle, then we can replace the remaining paths and cycles such that no new vertex added into a face is used and the paths (the cycles) are pairwise vertex-disjoint. Thus, G has a grid of size $\lfloor k/2 \rfloor \times \lfloor (k-1)/2 \rfloor$ as minor.

Theorem 7.5. *Given a planar graph G with n vertices and $k \in \mathbb{N}$, there is an algorithm that constructs either a tree decomposition for G of width $O(k)$ or a $\Theta(k) \times \Theta(k)$ grid as a minor of G in $O(nk^2)$ time.*

We finally want to remark that it was not the purpose of this paper to show the smallest possible approximation ratio. Indeed, it is not necessary to turn a given graph into an almost triangulated graph and to compute subsequently a tree decomposition for the almost triangulated graph.

With more sophisticated techniques, Kammer [16] presented an algorithm that can compute a tree decomposition of width $9tw(G) + 9$ for an unweighted planar graph G in $O(nk^4 \log k)$ time.

8 Conclusion

We have shown that a tree decomposition for a planar graph with its width approximating the treewidth by a constant factor can be found in a time linear in the number of vertices of the given graph. Since a tree decomposition for a planar graph with n vertices and treewitdh k can be of size $\Theta(nk)$, an interesting open question is if the running time of Corollary 7.4 can be improved to $O(nk)$.

To obtain a better approximation ratio Kammer [16] has shown how to adapt our algorithm from triangulated planar graphs to general planar graphs. This makes the algorithm much more complicated. Another more promising approach would be to find a linear-time triangulation of a planar (weighted) graph without increasing the treewidth of the graph.

We also want to mention that it is a still an open problem whether the treewidth on planar graphs can be found in polynomial time or whether the problem is NP-hard.

References

[1] Eyal Amir. Approximation algorithms for treewidth. *Algorithmica*, 56(4):448–479, 2010.

[2] S. Arnborg, D. G. Corneil, and A. Proskurowski. Complexity of finding embeddings in a k-tree. *SIAM J. Alg. Disc. Meth.*, 8(2):277–284, 1987.

[3] Stefan Arnborg, Jens Lagergren, and Detlef Seese. Easy problems for tree-decomposable graphs. *J. Algorithms*, 12(2):308–340, 1991.

[4] H. L. Bodlaender. A partial k-arboretum of graphs with bounded treewidth. *Theoret. Comput. Sci.*, 209(1–2):1–45, 1998.

[5] Hans L. Bodlaender. A tourist guide through treewidth. *Acta Cybern.*, 11(1-2):1–21, 1993.

[6] Hans L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM J. Comput.*, 25:1305–1317, 1996.

[7] Hans L. Bodlaender and Ton Kloks. Better algorithms for the pathwidth and treewidth of graphs. In *Proc. 18th International Colloquium on Automata, Languages and Programming (ICALP 1991)*, volume 510 of *LNCS*, pages 544–555. Springer, 1991.

[8] Erik D. Demaine, Fedor V. Fomin, Mohammad Taghi Hajiaghayi, and Dimitrios M. Thilikos. Subexponential parameterized algorithms on bounded-genus graphs and H-minor-free graphs. *J. ACM*, 52(6):866–893, 2005.

[9] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1):269–271, 1959.

[10] Uriel Feige, MohammadTaghi Hajiaghayi, and James R. Lee. Improved approximation algorithms for minimum weight vertex separators. *SIAM J. Comput.*, 38(2):629–657, 2008.

[11] Fedor V. Fomin, Sang-il Oum, and Dimitrios M. Thilikos. Rank-width and tree-width of H-minor-free graphs. *Eur. J. Comb.*, 31(7):1617–1628, 2010.

[12] Qian-Ping Gu and Hisao Tamaki. Optimal branch-decomposition of planar graphs in $O(n^3)$ time. *ACM Transactions on Algorithms*, 4(3), 2008.

[13] Qian-Ping Gu and Hisao Tamaki. Constant-factor approximations of branch-decomposition and largest grid minor of planar graphs in $O(n^{1+\epsilon})$ time. *Theor. Comput. Sci.*, 412(32):4100–4109, 2011.

[14] Qian-Ping Gu and Gengchun Xu. Near-linear time constant-factor approximation algorithm for branch-decomposition of planar graphs. In *Proc. 40th International Workshop on Graph-Theoretic Concepts in Computer Science (WG 2014)*, volume 8747 of *LNCS*, pages 238–249. Springer, 2014.

[15] John E. Hopcroft and Robert Endre Tarjan. Efficient planarity testing. *J. ACM*, 21(4):549–568, 1974.

[16] Frank Kammer. *Treelike and Chordal Graphs: Algorithms and Generalizations*. PhD thesis, University of Augsburg, 2010.

[17] Frank Kammer and Torsten Tholey. Approximate tree decompositions of planar graphs in linear time. In *Proc. 23th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2012)*, pages 683–698. SIAM, 2012.

[18] Ton Kloks, Chuan-Min Lee, and Jiping Liu. New algorithms for k-face cover, k-feedback vertex set, and k-disjoint cycles on plane and planar graphs. In *Proc. 28th International Workshop on Graph-Theoretic Concepts in Computer Science (WG 2002)*, volume 2573 of *LNCS*, pages 282–295. Springer, 2002.

[19] Jens Lagergren and Stefan Arnborg. Finding minimal forbidden minors using a finite congruence. In *Proc. 18th International Colloquium on Automata, Languages and Programming (ICALP 1991)*, volume 510 of *LNCS*, pages 532–543. Springer, 1991.

[20] Sang-il Oum. Rank-width is less than or equal to branch-width. *J. Graph Theory*, 57(3):239–244, 2008.

[21] Bruce A. Reed. Finding approximate separators and computing tree width quickly. In *Proc. 24th Annual ACM Symposium on Theory of Computing (STOC 1992)*, pages 221–228. ACM, 1992.

[22] Neil Robertson and Paul D. Seymour. Graph minors. III. planar tree-width. *J. Comb. Theory, Ser. B*, 36(1):49–64, 1984.

[23] Neil Robertson and Paul D. Seymour. Graph minors. II. algorithmic aspects of tree-width. *J. Algorithms*, 7(3):309–322, 1986.

[24] Neil Robertson and Paul D. Seymour. Graph minors. X. Obstructions to tree-decomposition. *J. Comb. Theory, Ser. B*, 52(2):153–190, 1991.

[25] Neil Robertson and Paul D. Seymour. Graph minors. XIII. The disjoint paths problem. *J. Comb. Theory, Ser. B*, 63(1):65–110, 1995.

[26] Paul D. Seymour and Robin Thomas. Call routing and the ratcatcher. *Combinatorica*, 14(2):217–241, 1994.