Computing Minimum Cycle Bases in Weighted Partial 2-Trees in Linear Time*

Carola Doerr†, G. Ramakrishna‡, Jens M. Schmidt§

Abstract

We present a linear time algorithm for computing an implicit linear space representation of a minimum cycle basis in weighted partial 2-trees (i.e., graphs of treewidth at most two) with non-negative edge-weights. The implicit representation can be made explicit in a running time that is proportional to the size of the minimum cycle basis.

For planar graphs, Borradaile, Sankowski, and Wulff-Nilsen [Min *st*-cut Oracle for Planar Graphs with Near-Linear Preprocessing Time, FOCS 2010] showed how to compute an implicit $O(n \log n)$ space representation of an minimum cycle basis in $O(n \log^5 n)$ time. For the special case of partial 2-trees, our algorithm improves this result to linear time and space. Such an improvement was achieved previously only for outerplanar graphs [Liu and Lu: Minimum Cycle Bases of Weighted Outerplanar Graphs, IPL 110:970–974, 2010].

1 Introduction

A *cycle basis* of a graph G is a minimum-cardinality set $\mathcal{C}$ of cycles in G such that every cycle C in G can be written as the exclusive-or sum of a subset of cycles in $\mathcal{C}$. A *minimum cycle basis* (MCB) of G is a cycle basis that minimizes the total weight of the cycles in the basis. Minimum cycle bases have numerous applications in the analysis of electrical networks, biochemistry, periodic timetabling, surface reconstruction, and public transportation, and have been intensively studied in the computer science literature. We refer the interested reader to [16] for an exhaustive survey. It is—both from a practical and a theoretical viewpoint—an interesting task to compute minimum cycle bases efficiently.

All graphs considered in this work are simple graphs $G = (V, E)$ with a non-negative edge-weight function $w : E \to \mathbb{R}_{\geq 0}$. (Computing MCBs for graphs with cycles of negative weight is an NP-hard problem [16]. In all previous work that we are aware of it is therefore assumed that the edge-weights are non-negative.) Throughout this work, $m = |E|$ denotes the size of the edge set and $n = |V|$ the size of the vertex set of G.

1.1 Previous Work

The first polynomial-time algorithm for computing MCBs was presented by Horton in 1987 [15]. His algorithm has running time $O(m^3 n)$. This was improved subsequently in a series of papers by different authors, cf. [16] or [18] for surveys of the history. The currently fastest algorithms for general graphs are a deterministic $O(m^2 n / \log n)$ algorithm of Amaldi, Iuliano, and Rizzi [2] and a Monte Carlo based algorithm by Amaldi, Iuliano, Jurkiewicz, Mehlhorn, and Rizzi [1] of running time $O(m^\omega)$, where ω is the matrix multiplication constant.

The algorithm from [1] is deterministic on planar graphs, and has a running time of $O(n^2)$. This improved the previously best known bound by Hartvigsen and Mardon [13], which is of order $n^2 \log n$.

*This is the full version of [11] which has appeared in the proceedings of the 39th International Workshop on Graph-Theoretic Concepts in Computer Science (WG 2013).

†Sorbonne Universités, UPMC Univ Paris 06, UMR 7906, LIP6, F-75005 Paris, France. Work initiated while the author was with Université Paris Diderot - Paris 7, LIAFA, Paris, France and the Max Planck Institute for Informatics.

‡Indian Institute of Technology Madras, India. Work initiated while visiting the Max Planck Institute for Informatics during an internship.

§TU Ilmenau, Germany. Work initiated while the author was with the Max Planck Institute for Informatics.

The currently best known algorithm on planar graphs is due to Borradaile, Sankowski, and Wulff-Nilsen [7]. It constructs an $O(n \log n)$ space implicit representation of an MCB in planar graphs in time $O(n \log^5 n)$.[1]

Faster algorithms for planar graphs are known only for the special case of outerplanar graphs. For unweighted outerplanar graphs, Leydold and Stadler [17] presented a linear time algorithm. More recently, Liu and Lu [18] presented a linear time, linear space algorithm to compute an MCB of a weighted outerplanar graph (using an implicit representation). This is optimal both in terms of time and space.

1.2 Our Result

In this contribution, we consider the computation of minimum cycle bases of partial 2-trees. The class of partial 2-trees (also referred to as graphs of treewidth at most two) is a strict superclass of outerplanar graphs. It includes 2-trees and series-parallel graphs (as shown in [5, Theorem 42], a graph G is a partial 2-tree if and only if every 2-connected component of G is a series-parallel graph). Partial 2-trees are planar. They are precisely the graphs that forbid a K_4-subdivision as a subgraph.

Partial 2-trees are extensively studied in the computer science literature, e.g., a deterministic logspace algorithm is presented to canonize and test isomorphism for partial 2-trees [3]; plane embeddings of partial 2-trees are described in [21]; parallel strategies can be used to find the most vital edges [14]; and the oriented chromatic number of partial 2-trees is studied in [20]. Partial 2-trees can be recognized in linear time [4].

Our main result is a linear time algorithm for computing an implicit $O(n)$-space representation of a minimum cycle basis in partial 2-trees. The explicit representation can be obtained in additional time that is proportional to the size of the MCB. Since partial 2-trees are planar graphs, the previously best known algorithm was the one by Borradaile, Sankowski, and Wulff-Nilsen [8]. That is, for the special case of partial 2-trees we are able to improve their running time by a factor of $\Theta(\log^4 n)$.

Our result is achieved by an iterative decomposition of the partial 2-tree into outerplanar graphs to which the recent result of Liu and Lu [18] can be applied. We state our main theorem below. As will be discussed in Section 5 the ideas presented here do not carry over to planar 3-trees. It thus seems that substantially new ideas are required to improve the running time of [8] for graph classes containing graphs of treewidth at least three.

Theorem 1. *Given a partial 2-tree $G = (V, E)$ on n vertices and a non-negative weight function $w : E \to \mathbb{R}_{\geq 0}$, a minimum cycle basis $\mathcal{B}$ of G (implicitly encoded in $O(n)$ space) can be obtained in $O(n)$ time.*

Moreover, $\mathcal{B}$ can be reported explicitly in time $O(\text{size}(\mathcal{B}))$, where $\text{size}(\mathcal{B})$ is the number of edges in $\mathcal{B}$ counted according to their multiplicity.

Note in Theorem 1 that although $\mathcal{B}$ has an *implicit* representation of linear size, the *explicit size* of $\mathcal{B}$ may be quadratic. This is true already for outerplanar graphs, cf. [18] for a simple ladder graph G in which the unique MCB of G contains $\Theta(n^2)$ edges.

For the proof of Theorem 1 it will be crucial that the set of *lex short cycles* (cf. Section 2.3) in any weighted partial 2-tree forms a minimum cycle basis [19]. As lex short cycles are inherently defined by shortest paths, we will need a data structure that reports the distance between two vertices in constant time (e.g., for checking whether an edge is the shortest path between its two endpoints). In outerplanar graphs, such a data structure exists due to Frederickson and Jannardan [12]. For our more general case, we will instead extend a result of Chaudhuri and Zaroliagis [9, Lemma 3.2] on weighted partial k-trees which supports distance queries between two vertices of a common bag in a (fixed) tree decomposition of G in constant time. Using this extension, we can report a shortest path P between any two such vertices in time $O(|E(P)|)$.

2 Graph Preliminaries, Partial 2-Trees, and Lex Shortest Paths

We consider weighted undirected graphs $G = (V, E)$ where V denotes the set of vertices, E the set of edges, and $w : E \to \mathbb{R}_{\geq 0}$ a non-negative weight function. We assume the usual unit-cost RAM as

[1] The arXiv paper [8] announces an improved running time of $O(n \log^4 n)$.

model of computation (real RAM when the input contains reals), but we do not use any low-level bit manipulations in our algorithm to improve the running time. The graphs considered in this paper are sparse; i.e., we have a linear dependence $m = O(n)$.

The weight $w(P)$ of a path P in G is the sum of weights of edges in P; i.e., $w(P) := \sum_{e \in P} w(e)$. A set $X \subset V$ of vertices in G is said to be a *vertex separator* of G if the removal of the vertices X increases the number of connected components. A vertex separator X is *minimal* if no proper subset of X is a vertex separator. For $Y \subseteq V$, we write $G[Y]$ for the subgraph of G that is induced by Y and we write $G - Y$ for the subgraph that is obtained from G by deleting the vertices in Y (and all incident edges). For $k, \ell \in \mathbb{N}$, we denote by K_ℓ the complete graph on ℓ vertices and by $K_{\ell,k}$ we denote the the complete bipartite graph on ℓ and k vertices. For a graph H, an H-subdivision is a graph obtained from H by replacing its edges with pairwise internally vertex-disjoint paths, each containing at least one edge. In this work, we will be mainly concerned with $K_{2,k}$-subdivisions for $k \geq 3$. In such a $K_{2,k}$-subdivision we call the two vertices of degree greater than two the *branch vertices* of the subdivision.

2.1 Minimum Cycle Bases

A cycle C in G is a connected subgraph of G in which every vertex has degree two. Let $C_1, \ldots, C_k$ be cycles in G and let $\oplus$ denote the symmetric difference function. Then the sum $H := C_1 \oplus \ldots \oplus C_k$ is the subgraph of G containing exactly those edges that appear an odd number of times in the multi-set $\{C_1, \ldots, C_k\}$. As is well known, $\mathcal{S}$ is a union of cycles in G.

We say that a set $\mathcal{C} = \{C_1, \ldots, C_k\}$ of cycles of G *spans the cycle space of* G if every cycle C of G can be written as a sum $C_{i_1} \oplus \ldots \oplus C_{i_\ell}$ of elements of $\mathcal{C}$. In this case, we say that $C_{i_1}, \ldots, C_{i_\ell}$ *generate* C. The size *size*($\mathcal{C}$) of $\mathcal{C}$ is the number of edges in $C_{i_1} \cup \ldots \cup C_{i_\ell}$ counted according to their multiplicity.

A *cycle basis* of G is a minimum cardinality set of cycles that spans the cycle space of G. Put differently, a cycle basis is a maximal set of independent cycles, where we consider a set of cycles to be independent if their incidence vectors in $\{0,1\}^m$ are independent over the field GF(2). The cardinality of a cycle basis is sometimes referred to as the *dimension* of the cycle space of G. The dimension of the cycle space of any simple connected graph equals $m - n + 1$ [6].

We are interested in identifying a *minimum cycle basis* (MCB) of G; i.e., a cycle basis $\mathcal{C}$ of minimum total weight $\sum_{C \in \mathcal{C}} w(C)$.[2]

If $G_1, \ldots, G_k$ are the 2-connected components of the graph G and if $\mathcal{C}_1, \ldots, \mathcal{C}_k$ are minimum cycle bases of $G_1, \ldots, G_k$, respectively, then the union $\mathcal{C}_1 \cup \ldots \cup \mathcal{C}_k$ is a minimum cycle basis of G. In what follows, we will therefore assume without loss of generality that G is 2-connected.

2.2 Tree Decompositions and Partial 2-Trees

A *tree decomposition* of a graph G is a pair $(\{X_1, \ldots, X_r\}, T)$ of a set of *bags* $X_1, \ldots, X_r$ and a tree T with vertex set $V(T) = \{X_1, \ldots, X_r\}$ that satisfies the following three properties:

1. $X_1 \cup \ldots \cup X_r = V$,
2. For each edge $\{u, v\} \in E$, there is an index $1 \leq i \leq r$ such that $\{u, v\} \subseteq X_i$, and
3. For each vertex $v \in V$, the bags in T containing v form a subtree of T (*subtree property*).

The *treewidth* of $(\{X_1, \ldots X_r\}, T)$ is $\max\{|X_1|, \ldots |X_r|\} - 1$. The *treewidth* of G is the minimum treewidth over all possible tree decompositions of G. We call a tree decomposition $(\{X_1, \ldots X_r\}, T)$ *optimal* if the treewidth of T is equal to the treewidth of G. To distinguish between the edges of G and T, we refer to the edges of T as *links*.

A *k-tree* is a graph of treewidth k for which the addition of any edge between non-adjacent vertices would increase the treewidth. The following lemma is folklore and characterizes k-trees.

Lemma 2. *A graph G is a k-tree if and only if G can be constructed from a K_{k+1} by iteratively adding new vertices such that the neighborhood of each such vertex is a k-clique.*

[2]Note that some publications define *cycles* as Eulerian subgraphs. If defined this way, it is well known that the elements of an MCB are still connected subgraphs in which every vertex has degree two, i.e., cycles in our sense [15].

A subgraph of a k-tree is called a *partial k-tree*. As mentioned in the introduction, partial 2-trees form a strict superclass of outerplanar graphs. While outerplanar graphs are characterized by forbidding K_4- and $K_{2,3}$-subdivisions as subgraphs, partial 2-trees are exactly the graphs that forbid K_4-subdivisions. Therefore, a partial 2-tree is outerplanar if and only if it does not contain a $K_{2,3}$-subdivision (as a subgraph). The following statement is taken from [19].

Lemma 3 (Lemma 2.4 in [19])**.** *A partial 2-tree G is not outerplanar if and only if G contains a $K_{2,3}$-subdivision. If G contains a $K_{2,k}$-subdivision for $k \geq 3$, each of its k non-branch vertices is contained in a separate connected component of $G - \{u, v\}$; in particular, $G - \{u, v\}$ has k connected components.*

2.3 Lex Shortest Paths and Lex Short Cycles

It is known (Proposition 4.5 in [13]) that for any edge-weighted simple graph G the set of so-called *lex short cycles* contains a minimum cycle basis. For outerplanar graphs [18] and partial 2-trees [19], the whole set of lex short cycles forms a minimum cycle basis. The notion of lex shortest paths and lex short cycles presented here is from [13].

Definition 4 (Lex Shortest Paths)**.** Let $G = (V, E)$ be a graph with weight function $w : E \to \mathbb{R}_{\geq 0}$. Let $\sigma : V \to \{1, 2, \ldots, n\}$ be an arbitrary ordering of the vertices.

A path P between two distinct vertices $u, v \in V$ is called a *lex shortest path* if for any other path P' between u and v either $w(P') > w(P)$ or $\big(w(P') = w(P)$ and $|E(P')| > |E(P)|\big)$ or $\big(w(P') = w(P)$, $|E(P')| = |E(P)|$ and $\min_{y \in V(P') \setminus V(P)} \sigma(y) > \min_{y \in V(P) \setminus V(P')} \sigma(y)\big)$ holds.

It is easily verified that between any two vertices u, v in G there exists exactly one lex shortest path. We refer to this path as $\mathrm{lsp}(u, v)$ (cf. also Proposition 4.1 in [13]). If the dependence of the graph is not clear from the context, we write $\mathrm{lsp}_G(u, v)$. Note that every subpath of a lex shortest path is a lex shortest path.

Definition 5 (Lex Short Cycles)**.** A *lex short cycle* C is a cycle that contains for any two vertices $u, v \in C$ the lex shortest path $\mathrm{lsp}(u, v)$. For an edge-weighted graph G, we denote by $\mathrm{LSC}(G)$ the set of all lex short cycles in G.

Lemma 6 ([13, 19])**.** *For any edge-weighted simple graph G, there is a set $\mathcal{B} \subseteq \mathrm{LSC}(G)$ such that $\mathcal{B}$ is a minimum cycle basis for G. Additionally, the set of lex short cycles $\mathrm{LSC}(G)$ forms a minimum cycle basis if G is a weighted partial 2-tree.*

Abusing notation (since $\mathrm{MCB}(G)$ may not be unique) we write $\mathrm{MCB}(G) \subseteq \mathrm{LSC}(G)$ and $\mathrm{MCB}(G) = \mathrm{LSC}(G)$, respectively, for the two statements in Lemma 6.

By Lemma 6 it would suffice to compute the set of lex short cycles in G for our purposes. This is the approach of Liu and Lu, who showed that for outerplanar graphs an implicit representation of $\mathrm{LSC}(G)$ can be computed in linear time. Before commenting further on our algorithm, we briefly note that in their paper, Liu and Lu assume that the shortest paths in their outerplanar graph are unique. They motivate this assumption by introducing a preprocessing routine that perturbs the input weights accordingly (cf. Lemma 1 in [18]). However, it is not obvious how to run this preprocessing step in linear time (as is misleadingly stated there), since the weight differences that are needed become exponentially small. Therefore, arithmetic operations on these numbers cannot be done in constant time; the unit-cost assumption was never meant to be stretched that far. However, it turns out that this assumption is not needed, as their algorithm relies only on the following fact, which we briefly note and prove here for the sake of completeness.

Lemma 7 (implicitly in [18])**.** *Let $G = (V, E)$ be a weighted outerplanar graph such that all the edges in G are the lex shortest paths between their two endpoints. A cycle C in G is a lex short cycle if and only if C is an induced cycle in G.*

Proof. Let C' be a non-induced cycle in G. Then there exists two non-consecutive vertices x', y' in C' with $\{x', y'\} \in E$. Since all edges in G are the shortest paths between their endpoints, by definition, we have $\mathrm{lsp}(x', y') = \{x', y'\}$. Since x' and y' are non-consecutive in C', the lex shortest path between these two vertices is not in C'. Hence, C' cannot be lex short.

Let us now consider a cycle C in G that is not lex-short. To show that C cannot be an induced cycle, we assume the contrapositive and show that this contradicts the outerplanarity of G. Let x and y be two vertices in C whose lex shortest path $\mathrm{lsp}(x,y) = (x = x_1, \ldots, x_t = y)$ is not contained in C. That is, there at least one index i such that the edge $\{x_i, x_{i+1}\}$ is not in C. Let p be a minimal such index. Since C is an induced cycle, the vertex x_{p+1} cannot lie on the cycle, i.e., $x_{p+1} \notin C$. Let q be the smallest index greater than p such that $x_q \in C$. We observe that x_p and x_q cannot be consecutive in C, for otherwise $x_1, \ldots, x_p, x_q, \ldots, x_t$ would be a path between x and y of length strictly smaller than $\mathrm{lsp}(x,y)$. This shows that there exists a $K_{2,3}$-subdivision with branch vertices x_p and x_q in G. G can thus not be outerplanar by Lemma 3. □ □

From this (and an implementation of a result in [12]) is not too difficult to see that an implicit representation of the set of lex short cycles of an outerplanar graph can be obtained in linear time, cf. [18] for details.

Theorem 8 ([18])**.** *For every weighted outerplanar graph G on n vertices an $O(n)$-space representation of the set $\mathrm{LSC}(G)$ can be computed in $O(n)$ time. From this representation, any cycle $C \in \mathrm{LSC}(G)$ can be computed explicitly in time $O(\mathrm{size}(C))$.*

As mentioned above, if we could compute for every pair of vertices u and v in G the lex shortest path between u and v, we could—like Liu and Lu—resort to computing $\mathrm{LSC}(G)$. However, since we do not know currently how to perturb the weights in linear time such that all lex shortest paths in G are also the unique shortest paths between its two endpoints, it is somewhat more challenging to achieve a linear running time for our generalization. Our algorithm will therefore not necessarily compute the set of lex short cycles. Instead, as we shall describe in the next section, we will do a decomposition of the graph G into outerplanar graphs using arbitrary shortest paths instead of lex shortest paths. The result of Liu and Lu will still be an essential step in our algorithm, as it allows us to handle the outerplanar graphs that result from our decomposition using Theorem 8.

3 High-Level Overview of Our Algorithm and Technical Details

We first describe the high-level idea of our algorithm; most proofs and the algorithmic details are presented in the subsequent sections. From now on we assume that G is a 2-connected weighted partial 2-tree.

3.1 Removal of Long Edges

We call an edge $\{u,v\}$ *tight* if it is a shortest path between u and v, and we call it *long* otherwise. By the observation made in the following lemma, we will treat the long edges in G separately. In fact, the lemma allows us to ignore the set L of all long edges in the main routine of our algorithm. For two vertices u and v in G, let $\mathrm{sp}(u,v)$ be an arbitrary shortest path between u and v.

Lemma 9. *Let $G = (V,E)$ be a weighted partial 2-tree and let L be the set of long edges in G. Then $\mathrm{MCB}(G) = \mathrm{MCB}(G \setminus L) \cup \{\{e\} \cup \mathrm{sp}(u,v) \mid e = \{u,v\} \in L\}$.*

Proof. The independence of $\mathcal{M} := \mathrm{MCB}(G \setminus L) \cup \{\{e\} \cup \mathrm{sp}(u,v) \mid e = \{u,v\} \in L\}$ follows from the independence of $\mathrm{MCB}(G \setminus L)$ and the fact that e is contained in $\mathcal{M}$ only in the one cycle $\{e\} \cup \mathrm{sp}(u,v)$. Since $\mathcal{M}$ contains exactly $m - n + 1$ cycles, it is (cf. the comment in Section 2.1) also a *maximal* set of independent cycles; i.e., a cycle basis. We verify that the total weight of the cycles in $\mathcal{M}$ is minimal: According to Lemma 6, $\mathrm{LSC}(G \setminus L)$ is a minimum cycle basis of $G \setminus L$. Thus, the weight of the cycles in $\mathrm{MCB}(G \setminus L)$ equals the weight of the cycles in $\mathrm{LSC}(G \setminus L)$. In addition, the weight of each cycle $\{e\} \cup \mathrm{sp}(u,v)$ equals the weight of $\{e\} \cup \mathrm{lsp}(u,v)$. It follows directly from the two arguments given above that $\mathrm{MCB}(G \setminus L) \cup \{\{e\} \cup \mathrm{sp}(u,v) \mid e = \{u,v\} \in L\} = \mathrm{LSC}(G \setminus L) \cup \{\{e\} \cup \mathrm{lsp}(u,v) \mid e = \{u,v\} \in L\}$. For every long edge $e = \{u,v\} \in L$, the cycle $\{e\} \cup \mathrm{lsp}(u,v)$ is a lex short cycle in G; it is also the only lex short cycle that contains the edge e. Therefore, $\mathrm{LSC}(G) = \mathrm{LSC}(G \setminus L) \cup \{\{e'\} \cup \mathrm{lsp}(u',v') \mid e' = \{u',v'\} \in L\}$. Since $\mathrm{LSC}(G)$ is a minimum cycle basis of G, $\mathcal{M}$ is of minimum weight. □ □

According to Lemma 9, we can ignore long edges, but need to ensure that we add a short cycle containing e for every edge $e \in L$ to the minimum cycle basis at the very end of the main routine. That long edges can be identified and removed from G in $O(n)$ time using a suitable data structure will be shown in Lemma 18. The removal of the long edges from G does therefore not change the linear runtime of our algorithm.

3.2 High-Level Overview of the Main Algorithm

In a first step of Algorithm 1 we remove the set of long edges L from G (they will be taken care of later on using Lemma 9). The key approach for our algorithm is then to iteratively decompose the graph $G \setminus L$ into outerplanar graphs $\tilde{G}_1, \ldots, \tilde{G}_r$. To these graphs we apply the linear time algorithm of Liu and Lu (Theorem 8). Intuitively, the decomposition is done as follows.

When $G \setminus L$ is not outerplanar, then there exists a $K_{2,3}$-subdivision in $G \setminus L$ with branch vertices u and v such that (i) $\{u, v\}$ is a minimum vertex separator of $G \setminus L$ and (ii) the removal of $\{u, v\}$ disconnects $G \setminus L$ into at least three connected components $H_1, \ldots, H_k$ (cf. Lemma 3). We distinguish two cases. If $\{u, v\} \in E$, we set $G_h := (G \setminus L)[V(H_h) \cup \{u, v\}]$, $1 \leq h \leq k$. Otherwise, let $\mathrm{sp}(u, v)$ be an arbitrary shortest path between u and v in $G \setminus L$ and let $j(u, v) \in \{1, 2, \ldots, k\}$ such that $\mathrm{sp}(u, v) \in (G \setminus L)[V(H_{j(u,v)}) \cup \{u, v\}]$.

We set $G_{j(u,v)} := (G \setminus L)[V(H_{j(u,v)}) \cup \{u, v\}]$, and for all $1 \leq h \neq j(u, v) \leq k$ we set $G_h := (G \setminus L)[V(H_h) \cup \{u, v\}] \cup \mathrm{green}(u, v)$, where $\mathrm{green}(u, v)$ denotes a new "colored" (i.e., marked) edge $\{u, v\}$ that serves as a placeholder for the shortest path $\mathrm{sp}(u, v)$ between u and v (which, by definition, is not contained in G_h). The weight $w(\mathrm{green}(u, v))$ assigned to this new edge is therefore set to the weight $w(\mathrm{sp}(u, v))$ of the shortest path between u and v. Clearly, $w(\mathrm{green}(u, v)) = w(\mathrm{lsp}(u, v))$. Let the operation $\mathrm{decomp}(G \setminus L, u, v)$ decompose $G \setminus L$ into $G_1, \ldots, G_k$ with respect to the vertices u, v. We call each G_h, $1 \leq h \leq k$, a *part* of $\mathrm{decomp}(G \setminus L, u, v)$.

We now iteratively decompose the graphs $G_1, \ldots, G_k$ as described above until we are left with graphs $\tilde{G}_1, \ldots, \tilde{G}_r$ that do not contain any $K_{2,3}$-subdivision, i.e., with outerplanar graphs according to Lemma 3. Since all the edges in $\tilde{G}_h$ are tight, the set of lex short cycles in $\tilde{G}_h$ equals the boundaries of its internal faces. Extracting the internal faces of $\tilde{G}_h$ can be done in linear time, cf. [18] or the comments before and after Lemma 7. We will show in Theorem 20 that the (disjoint) union of $\mathrm{expand}(\mathrm{LSC}(\tilde{G}_1)), \ldots, \mathrm{expand}(\mathrm{LSC}(\tilde{G}_r))$ forms a minimum cycle basis, where, naturally, $\mathrm{expand}(\mathrm{LSC}(G_h))$ replaces the marked edges $\mathrm{green}(u, v)$ in every cycle by the shortest path $\mathrm{sp}(u, v)$.

Finally, we add to this minimum cycle basis the cycles $e \cup \mathrm{sp}(u, v)$ for all long edges $e = \{u, v\} \in L$ in G, where again $\mathrm{sp}(u, v)$ is an arbitrary shortest path between u and v. This can be done either implicitly by storing u, v, and the graph G or explicitly by computing the shortest paths with Lemma 18.

An important part of the algorithm is to find a data structure that allows to identify all $K_{2,3}$-subdivisions and to do the respective decomposition in linear time. To this end, we define *suitable tree decompositions*.

3.3 Suitable Tree Decompositions

We define suitable tree decompositions and we show how they help in efficiently computing our decomposition. To ease readability, we consider *rooted* tree decompositions. We direct all links in the tree decomposition from the root to the leaves, that is, for a link (X, Y) in the decomposition, X has smaller distance to the root than Y. Bag X will then be referred to as the *father*, and bag Y is referred to as the *child*. All links $\ell = (X, Y)$ in the tree decomposition are *labeled* by the intersection $X \cap Y$ of its two endpoints.

Definition 10 (Suitable Tree Decomposition)**.** An optimal rooted tree decomposition of G is *suitable* if it satisfies the following properties:

1. The size of every bag X_i is 3 and every two adjacent bags X_i, X_j in T differ by exactly one vertex; i.e., $|X_i \cap X_j| = 2$ (this property is called *smooth* in [4]).

2. Any two links with the same label have a common father in T; i.e., for any two links (X_1, Y_1) and (X_2, Y_2) with $X_1 \cap Y_1 = X_2 \cap Y_2$ it holds that $X_1 = X_2$.

Observe that for any internal bag in T, the number of children could be arbitrary, but there are at most three different labels associated with the links to its children.

Our algorithm will perform all computations in a suitable tree decomposition of the tight induced subgraph of G. It is therefore important that such a tree decomposition can be computed in linear time.

Lemma 11. *Given a partial 2-tree G, a suitable tree decomposition of linear size can be computed in linear time.*

Proof. The number of bags of a smooth tree decomposition of a partial 2-tree G is $n-2$ (see Lemma 2.5 in [4]). As every bag contains exactly 3 vertices, this gives a linear space representation of the tree decomposition. A rooted tree decomposition $(\{X_1, \ldots, X_r\}, T)$ of G that is additionally smooth can be computed using Bodlaender's algorithm [4]. We make T suitable (i.e., we add Property 10.2) as follows. By the subtree-property of tree decompositions, it suffices to give a smooth tree decomposition T' in which no two links (A, B) and (B, C) have the same label.

Traverse T in any order that starts on the root and in which father bags precede their children. Whenever a bag B with father A and child C is visited such that the links (A, B) and (B, C) have the same labels, we modify T by attaching the subtree of T that is rooted on C to A; i.e., after the modification C is a sibling of B. This causes B and C to have the same father. It is straight-forward to see that the modified tree is still a smooth tree decomposition. After the traversal is finished, we have a suitable tree decomposition. Clearly, all modifications can be computed in constant time per step, leading to a total running time of $O(n)$. □ □

One of the key observations of our algorithm is the fact that for all $K_{2,3}$-subdivisions the two branch vertices must be contained in at least *three common bags* of a suitable tree decomposition. This is shown using the following results (cf. Corollary 15).

Lemma 12 (Lemma 12.3.4 in [10])**.** *Let $W \subseteq V(G)$ and let T be a tree decomposition of G. Then T contains either a bag that contains W or a link (X_1, X_2) such that two vertices of W are separated by $X_1 \cap X_2$ in G.*

Lemma 13 ($K_{2,3}$-Subdivisions in Partial 2-Trees)**.** *Let u and v be the branch vertices of a $K_{2,3}$-subdivision H in a partial 2-tree G. For every optimal tree decomposition T of G without bag duplicates (in particular for suitable tree decompositions), $\{u, v\}$ is contained in at least one bag of T.*

Proof. We show the claim by applying Lemma 12 with $W = \{u, v\}$. If u and v are not contained in a bag, there must be a link (X_1, X_2) in T such that u and v are separated by $X_1 \cap X_2$ in G. Since H is a $K_{2,3}$-subdivision, at least three vertices need to be removed in order to separate u and v. Since T is optimal, $X_1 \cap X_2$ can only contain more than two vertices when X_1 and X_2 consist of the same three vertices. This contradicts that there are no bag duplicates in T. □ □

Now we can prove the desired Corollary 15 with the following lemma.

Lemma 14. *Let T be a suitable tree decomposition of a 2-connected partial 2-tree G and let $u, v \in V$. Then T contains at least three bags that contain both u and v if and only if $G - \{u, v\}$ has at least three connected components. If T contains at least $k \geq 3$ such bags, the number of connected components in $G - \{u, v\}$ is exactly k; in particular, G contains then a $K_{2,k}$-subdivision.*

Proof. In the following, let $Y_1, \ldots, Y_k$ be the bags in T containing both u and v. By the subtree property of tree decompositions, $Y_1, \ldots, Y_k$ induce a connected subgraph in T. Since T is suitable, we can assume that $Y_2, \ldots, Y_k$ are children of Y_1. Let F be the forest obtained from T by deleting the links $(Y_1, Y_2), \ldots, (Y_1, Y_k)$. For each $1 \leq i \leq k$, let T_i be the subtree in F containing Y_i and let $V_i = \{x \in V \mid \text{ there is a bag containing } x \text{ in } T_i\}$.

Let T contain $k \geq 3$ bags that contain both u and v. We prove that $G - \{u, v\}$ has at least k connected components. Let $i, j \in \{1, \ldots, k\}$ with $i \neq j$. By the subtree property of tree decompositions and $Y_i \neq Y_j$, we have $V_i \cap V_j = \{u, v\}$. Since all bags in T contain exactly three vertices, the sets $V_i - V_j$ and $V_j - V_i$ are non-empty. We need to show that for all $x \in V_i - V_j$ and all $y \in V_j - V_i$ there is no edge xy in G. Assume to the contrary that such an edge xy exists. Then there is a bag B in T that contains both x and y. Since $x \in V_i - \{u, v\}$ and by the subtree property, T_i is the only tree with a bag

containing x. Similarly, T_j is the only tree with a bag containing y. This contradicts the existence of B. Thus, every of the $V_i - \{u, v\}$ is the vertex set of a connected component of $G - \{u, v\}$.

Let $G - \{u, v\}$ have $\ell \geq 3$ connected components. We prove that T contains at least ℓ bags that contain both u and v. It is well-known that every connected component that is obtained by deleting a minimal vertex separator is adjacent to all vertices of this separator. Since G is 2-connected, $\{u, v\}$ is a minimal vertex separator of G. It follows that G contains a $K_{2,\ell}$-subdivision H with branch-vertices u and v and non-branch-vertices $x_1, \ldots, x_\ell$ different from u and v. According to Lemma 13, at least one bag of T contains both u and v. Thus, there is at least one Y_i, T_i and V_i defined as above (note that $T_1 = T$ and $V_1 = V$ if Y_i is the only bag containing u and v). We prove that no set V_i contains two vertices x_a and x_b with $1 \leq a < b \leq \ell$. This gives the claim, as it implies that there are at least l trees T_i and thus, at least l bags Y_i, each of which contains u and v. By construction of T_i, T_i has exactly one bag C that contains u and v (note that the uniqueness of C exploits the fact that T is suitable). Let c be the vertex $C \setminus \{u, v\}$. Assume to the contrary that T_i contains a bag A containing x_a and a bag B containing x_b ($A = B$ is possible). Let L be the least common ancestor of A and B in T_i. We know from the existence of H that there are two independent paths from x_a to u and v, respectively, that share only the vertices u and v; similarly, there are two such independent paths from x_b to u and v, respectively. It follows that $L \neq C$ and, in particular $c \notin \{x_a, x_b\}$, since otherwise two of the four independent paths from $\{x_a, x_b\}$ to $\{u, v\}$ would intersect in $c \notin \{u, v\}$. However, if $L \neq C$, two of these independent paths must intersect in c as well in order to reach u and v, which gives the desired contradiction. □ □

From Lemmata 3 and 14, we obtain the following corollary.

Corollary 15. Let T be a suitable tree decomposition of a 2-connected partial 2-tree G. Then G is outerplanar if and only if for every two nodes $u, v \in V$ at most two bags of T contain u and v.

Corollary 15 allows us to efficiently find all $K_{2,k}$-subdivisions for $k \geq 3$ in a 2-connected partial 2-tree by finding k pairwise adjacent bags in a suitable tree decomposition that share the same two vertices u and v.

3.4 Suitable Data Structures for Finding the Lex Shortest Paths

Another useful tool in our algorithm will be the following data structure. It supports the query for an intermediate vertex that lies on a shortest path between two nodes. The following lemma is along the lines of [12].

Lemma 16. *Let T be an unrooted tree decomposition of G. There is a linear space data-structure with $O(n)$ preprocessing time that supports the following query: Given a bag $A \in T$ and a vertex v in G that is not in A, find the link incident to A that leads to some bag containing v. The query time is $O(\log d)$, where d is the degree of A in T.*

Proof. Note that the desired link is unique, as T is a tree decomposition. For building the data structure, we perform a depth first search (dfs) on T, starting at an arbitrary artificial root, and label every bag X with a dfs-number. We label each vertex of a bag X with the dfs-number of X. For any bag X and the subtree $T(X)$ of T that is rooted at X, the bags in $T(X)$ get consecutive dfs-numbers. Hence, these numbers form an interval, which we can store in constant space at X during the depth first search; i.e., in linear total time. Similarly, the bags not in $T(X)$ get dfs-numbers that are consecutive in the cyclic order of dfs-numbers; we store the corresponding interval at the father bag of X (if exists). The desired answer for the query is then obtained by performing a binary search on the neighboring bags of A (performed in the same order as the dfs) that stops at the bag having an interval that contains the label of v. This takes time $O(\log d)$. □ □

We are finally ready to show that for any long edge $\{u, v\}$ a shortest path $\mathrm{sp}(u, v)$ between u and v can be computed in time $O(|E(\mathrm{sp}(u, v))|)$. The following lemmata will be useful also to identify the subtree of the tree decomposition that contains $\mathrm{sp}(u, v)$ for two branch vertices u and v with $\{u, v\} \notin E$.

Lemma 17 (Lemma 3.2 in [9])**.** *Given a partial k-tree G and an optimal tree decomposition T of G, there is an algorithm with running time $O(k^3 n)$ that outputs the distances of all vertex pairs that are contained in common bags and that, for each such vertex pair, outputs some intermediate vertex of a shortest path between the vertices.*

Lemma 17 is originally stated for directed graphs in [9]. However, representing each undirected edge with two edges oriented in opposite directions gives the above undirected variant.

We extend Lemma 17 by giving the following data structure.

Lemma 18. *Given a connected partial 2-tree G and a suitable tree decomposition T of G, there is an $O(n)$-space data structure requiring $O(n)$ preprocessing time that supports the following queries, given two vertices u and v and a bag $X \in T$ that contains u and v:*

- *Compute in time $O(1)$ the length of a shortest path between u and v* (distance query)*.*
- *Compute in time $O(1)$ an intermediate vertex w of some shortest path between u and v, and a bag $Y \in T$ such that $Y = \{u, v, w\}$, providing that any shortest path between u and v has at least two edges* (intermediate vertex query)*.*
- *Compute in time $O(|E(P)|)$ a shortest path P between u and v* (shortest path extraction)*.*

Since a tree decomposition maintains for every edge the bag that contains it, the queries of Lemma 18 can in particular be performed when—instead of the bag X—an edge $\{u, v\} \in G$ is given.

of Lemma 18. We apply the algorithm of Lemma 17 and store the distance of every vertex pair $\{u, v\}$ that is contained in a common bag, say in X, in a table linked to X. Since T contains only linearly many bags, this takes $O(n)$ space. The table supports distance queries in constant time, as there are only constantly many vertex pairs in each bag.

Assume for the moment that we know how to support the intermediate vertex query. Then we can easily support the shortest path extraction by first applying an intermediate vertex query, which gives Y, and subsequently recursing on the two intermediate vertex queries $\{u, w\}$ and $\{w, v\}$, both in Y, until each shortest path is just an edge. This allows to extract a shortest path between u and v in time proportional to its length.

It remains to show how to support intermediate vertex queries. We initialize the data structure D of Lemma 16 for the tree decomposition T in time $O(n)$ and apply the algorithm of Lemma 17 in time $O(n)$. Let X be a bag containing u and v. By Lemma 17, we have already found an intermediate vertex z between u and v, but want to find an intermediate vertex w that is in a common bag Y with u and v. If z does not exist, there is a shortest path that is just an edge, in which case we just set w to be non-existent as well. If $z \in X$, we set $w = z$ and $Y = X$ and are done.

Otherwise, we query D with (X, z) and get a link (X, A) such that z is contained in the subtree of T that is separated by (X, A) and does not contain X (note that A may be the father of X in T). According to Lemma 16, this query takes time proportional to at most the degree of X in T.

We now distinguish two cases. In the case that A contains u and v, we iterate this procedure on A instead on X. In this iteration, this case cannot happen more than a constant number of times, as T is suitable, so any path in the subtree of T consisting of bags containing $\{u, v\}$ has length at most 2.

Otherwise, A contains exactly one vertex of $\{u, v\}$, say u. Consider $X = \{u, v, r\}$ and the subtree T_1 of T that is separated by the link (X, A) and contains A. By the subtree property, T_1 cannot contain a bag with v, as then v would also be contained in A. Since T_1 contains a part of a shortest path between u and v, but has only u and r in common with X, r must be an intermediate vertex. Since X contains u, v, and r, we set $w = r$ and $Y = X$.

We investigate the preprocessing time of the data structure, i.e., the time spent computing for all vertex pairs (u, v) the intermediate vertex w and the bag containing all three vertices $\{u, v, w\}$. In every bag X, there are only constantly many vertex pairs. For each such vertex pair, we could find w in time $O(\deg(X))$, where $\deg(X)$ is the degree of X in T. Hence, the preprocessing time sums up to a linear total. □ □

3.5 Obtaining $\mathrm{MCB}(G)$ from $\mathrm{LSC}(\tilde{G}_1), \ldots, \mathrm{LSC}(\tilde{G}_r)$

As a last technicality, we show that—as claimed in the high-level overview of our algorithm—the disjoint union $\mathrm{expand}(\mathrm{LSC}(\tilde{G}_1)) \uplus \ldots \uplus \mathrm{expand}(\mathrm{LSC}(\tilde{G}_r))$ forms a minimum cycle basis of $G \setminus L$.

Recall that $G_{j(u,v)}$ is the part of $\mathrm{decomp}(G \setminus L, u, v)$ containing the shortest path $\mathrm{sp}(u, v)$ between u and v along which we have decomposed $G \setminus L$.

Definition 19. For any cycle C of $G \setminus L$, let $\text{expand}(C)$ be the cycle obtained from C by replacing the green edges $\text{green}(u,v)$ in C (if exist) by the shortest path $\text{sp}(u,v)$ between u and v in the part $G_{j(u,v)}$. For a set of cycles $\mathcal{C}$, let $\text{expand}(\mathcal{C}) := \{\text{expand}(C) \mid C \in \mathcal{C}\}$.

Theorem 20. $\text{MCB}(G \setminus L) = \text{expand}(\text{LSC}(\tilde{G}_1)) \uplus \ldots \uplus \text{expand}(\text{LSC}(\tilde{G}_r))$.

Theorem 20 follows from iteratively applying the following lemma.

Lemma 21. *Let G be a graph in which every edge is tight. Let u and v be the two branch vertices of a $K_{2,3}$-subdivision in G. Let $G_1, \ldots, G_k$ be the subgraphs resulting from the decomposition* $\text{decomp}(G,u,v)$. *For each $1 \leq h \leq k$, let $\mathcal{B}_h$ be a minimum cycle basis of the graph G_h. Then $\mathcal{E} := \text{expand}(\mathcal{B}_1) \cup \ldots \cup \text{expand}(\mathcal{B}_k)$ is a minimum cycle basis of G.*

To prove Lemma 21 we first introduce an alternative decomposition, $\text{decomp}^*(G,u,v)$, which decomposes a non-outerplanar graph with respect to the *lex short path* between the two branch vertices—as opposed to the decomposition $\text{decomp}(G,u,v)$ which decomposes G along an arbitrary shortest path.

Similarly to the decomposition $\text{decomp}(G,u,v)$ described in Section 3.2 let G be a graph that is not outerplanar and let $u,v \in V$ be the branch vertices of a $K_{2,3}$ subdivision in G. Let $H_1, \ldots, H_k$ be the connected components of $G - \{u,v\}$.

Case 1: If $\{u,v\} \in E$, set $G_h^* := G_h = G[V(H_h) \cup \{u,v\}]$, $1 \leq h \leq k$.

Case 2: If $\{u,v\} \notin E$, let $i(u,v) \in \{1,2,\ldots,k\}$ such that $\text{lsp}(u,v) \in G[V(H_{i(u,v)}) \cup \{u,v\}]$. Set $G^*_{i(u,v)} := G[V(H_{i(u,v)}) \cup \{u,v\}]$, and for all $1 \leq h \neq i(u,v) \leq k$ we set $G_h^* := G[V(H_h) \cup \{u,v\}] \cup \text{blue}(u,v)$, where $\text{blue}(u,v)$ is a marked edge $\{u,v\}$ that serves as a placeholder for the lex shortest path $\text{lsp}(u,v)$ between u and v (which is not contained in G_h^*). Set $w(\text{blue}(u,v)) = w(\text{lsp}(u,v))$ For any cycle C of G, let $\text{expand}^*(C)$ be the cycle obtained from C by replacing the blue edges $\text{blue}(u,v)$ in C (if exist) by the lex shortest path $\text{lsp}(u,v)$. For a set of cycles $\mathcal{C}$, let $\text{expand}^*(\mathcal{C}) := \{\text{expand}^*(C) \mid C \in \mathcal{C}\}$.

The proof of Lemma 21 is based on the following result, which we believe to be of independent interest.

Lemma 22. *In the setting of Lemma 21 let $G_1^*, \ldots, G_k^*$ be the subgraphs resulting from the decomposition* $\text{decomp}^*(G,u,v)$. *Then* $\text{LSC}(G) = \text{expand}^*(\text{LSC}(G_1^*)) \uplus \ldots \uplus \text{expand}^*(\text{LSC}(G_k^*))$.

Lemma 22 can be proven using the following observation.

Observation 23 (Lemma 2.5 and Corollary 2.8 in [19])**.** *Let G be a weighted graph and let G' be a subgraph of G. Let P be a path in G'. If P is lex shortest in G, it is lex shortest in G'.*

Furthermore, for G, k, and $G_1^*, \ldots, G_k^*$ as in Lemma 22, we have $\text{expand}^*(\text{LSC}(G_h^*)) \subseteq \text{LSC}(G)$, $1 \leq h \leq k$.

of Lemma 22. The disjointness of the sets follows immediately from the facts that only cycles are contained in these sets and that the subgraphs $H_1, \ldots, H_k$ in $G - \{u,v\}$ are disjoint. The inclusion $\bigcup_{h=1}^k \text{expand}^*(\text{LSC}(G_h^*)) \subseteq \text{LSC}(G)$ follows from Observation 23.

It thus remains to show $\text{LSC}(G) \subseteq \bigcup_{h=1}^k \text{expand}^*(\text{LSC}(G_h^*))$. To this end, let $C \in \text{LSC}(G)$. We need to show that $E(C) \setminus E(\text{lsp}_G(u,v))$ is contained in one of the G_h^*; Observation 23 implies that for any such cycle either C itself or the cycle $\text{shrink}(C)$ with the lex shortest path $\text{lsp}_G(u,v)$ replaced by the edge $\text{blue}(u,v)$ must be contained in $\text{LSC}(G_h^*)$.

Since the decomposition is done along the vertices u and v, there is nothing to show in case $|C \cap \{u,v\}| \leq 1$. Indeed, any such C or its short version $\text{shrink}(C)$ is contained in exactly one connected component G_h^*.

Let us therefore assume that both vertices u and v are contained in C. Since C is a lex short cycle in G, it must contain the lex shortest path $\text{lsp}(u,v)$ between u and v. The cycle C is complemented by another path P from u to v; i.e., there exists a path P from u to v such that $C = \text{lsp}(u,v) \cup P$ and $V(P) \cap V(\text{lsp}(u,v)) = \{u,v\}$. By the structure of our decomposition, this path P is certainly contained in one connected component G_h^*. Since G_h^* contains also either $\text{lsp}(u,v)$ itself or the placeholder edge $\text{blue}(u,v)$ we have that either C or $\text{shrink}(C)$ is contained in G_h^*. As mentioned above, by Observation 23 it follows that $C \in \text{expand}^*(\text{LSC}(G_h^*))$. □ □

Before we are finally ready to prove Lemma 21, we observe that from Lemma 3 it follows that in the parts $G_{j(u,v)}$ and $G^*_{i(u,v)}$ there are no two vertex-disjoint paths between u and v.

Observation 24. *If a partial 2-tree G contains a $K_{2,3}$-subdivision with branch vertices $\{u, v\}$, the subgraph $G_{j(u,v)}$ defined by* $\mathrm{decomp}(G, u, v)$ *and the subgraph $G^*_{i(u,v)}$ defined by* $\mathrm{decomp}^*(G, u, v)$ *do not contain a cycle that contains both vertices u and v.*

of Lemma 21. For $1 \leq h \leq k$ let m_h be the number edges in the graph G_h and let n_h be the number of vertices in G_h. By the observation made in Section 2.1 the number of cycles in $\mathcal{B}_h$ equals $m_h - n_h + 1$. The number of cycles in $\mathcal{E}$ therefore equals

$$\sum_{h=1}^{k} (m_h - n_h + 1) = m + k - 1 - (n + 2(k-1)) + k = m - n + 1.$$

We therefore need to show that the cycles in $\mathcal{E}$ are independent and that they are of minimum total weight.

As for the independence assume that there exist cycles $C_1, \ldots, C_\ell$ in $\mathcal{B}_1 \cup \ldots \cup \mathcal{B}_k$ with $\mathrm{expand}(C_1) \oplus \ldots \oplus \mathrm{expand}(C_\ell) = 0$. For each h let R_h be the set of indices r such that $C_r \in G_h$. Since the only vertices that appear in more than one subgraph G_h are the two vertices u and v and since the sum of cylces forms a disjoint union of cycles, $\sum_{r \in R_h} C_r = 0$ must hold. This implies $R_h = \emptyset$ for all h and shows the independence of the cycles in $\mathcal{E}$.

Since the expansion of a cycle $C \in \mathcal{B}_h$ does not change its total cost, the total weight of the cycles in $\mathcal{E}$ equals the total weight of the cycles in $\mathcal{B}_1 \cup \ldots \cup \mathcal{B}_k$. We need to show that there is no cycle basis of G that has strictly smaller cost. By Lemma 22 we know that, if we decompose G along the lex shortest path $\mathrm{lsp}_G(u, v)$ between u and v, i.e., if we apply $\mathrm{decomp}^*(G, u, v)$, then the set $\mathcal{E}^*$ as defined in Lemma 22 forms a minimum cycle basis. We show that the cycles in $\mathcal{E}^*$ are of the same total weight as the cycles in $\mathcal{E}$. If $\{u, v\} \in E$, this is trivially true as the decompositions $\mathrm{decomp}^*(G, u, v)$ and $\mathrm{decomp}(G, u, v)$ are identical. Let $i = i(u, v)$ and $j = j(u, v)$.

We first consider the case when $(u, v) \notin E(G)$, $i = j$, and $\mathrm{sp}(u, v) \neq \mathrm{lsp}(u, v)$. For each $1 \leq h \neq i \leq k$, G_h is same as G^*_h except that (u, v) is a green edge in G_h and (u, v) is a blue edge in G^*_h. Furthermore, G_i is same as G^*_i. We know that $\mathrm{green}(u, v)$ and $\mathrm{blue}(u, v)$ are placeholders for $\mathrm{sp}(u, v)$ and $\mathrm{lsp}(u, v)$, respectively. Also the weights of $\mathrm{sp}(u, v)$ and $\mathrm{lsp}(u, v)$ are same. Thus total weight of the cycles in $\mathcal{E}^*$ is same as the total weight of the cycles in $\mathcal{E}$.

Consider the remaining case where $(u, v) \notin E(G)$, $i \neq j$, and $\mathrm{sp}(u, v) \neq \mathrm{lsp}(u, v)$. We define a bijective mapping

$$\psi : \mathrm{LSC}(G^*_1) \cup \ldots \cup \mathrm{LSC}(G^*_k) \to \mathrm{LSC}(G_1) \cup \ldots \cup \mathrm{LSC}(G_k)$$

such that $w(\psi(C)) = w(C)$ for all $C \in \mathrm{LSC}(G^*_1) \cup \ldots \cup \mathrm{LSC}(G^*_k)$. From this we get the statement by evoking again Lemma 6 which tells us that for each h the sum of the weights of the cycles in $\mathrm{LSC}(G_h)$ equals the sum of the weights of the cycles in $\mathcal{B}_h$. For $h \notin \{i, j\}$, $G_h = G^*_h \setminus \{\mathrm{blue}(u, v)\} \cup \mathrm{green}(u, v)$ holds. For $C \in \mathrm{LSC}(G^*_h)$ we can therefore set $\psi(C) := C \setminus \{\mathrm{blue}(u, v)\} \cup \mathrm{green}(u, v)$ if $\mathrm{blue}(u, v) \in C$, and $\psi(C) := C$ otherwise. As we have mentioned in Observation 24, there is no cycle in G^*_i that contains both u and v. It is thus easy to see that $\mathrm{LSC}(G^*_i) \subseteq \mathrm{LSC}(G_i)$. Set $\psi(C) := C$ for all $C \in \mathrm{LSC}(G^*_i)$. Since there is one more edge in G_i as there is in G^*_i, the number of cycles in $\mathrm{LSC}(G_i)$ is by one larger than the number of cycles in $\mathrm{LSC}(G^*_i)$. Note that $D := \{\mathrm{green}(u, v)\} \cup \mathrm{lsp}_G(u, v)$ is a lex short cycle in G_i that is not in the image $\{\psi(C) \mid C \in \mathrm{LSC}(G^*_i)\}$. The cost of D is $2w(\mathrm{lsp}_G(u, v))$. Note that the situation is symmetric for G^*_j and G_j. We thus set $\psi(D^*) := D$ for $D^* := \{\mathrm{blue}(u, v)\} \cup \mathrm{sp}(u, v)$ and we set $\psi(C) := C \setminus \{\mathrm{blue}(u, v)\} \cup \mathrm{sp}(u, v)$ for all other cycles in $\mathrm{LSC}(G^*_j)$. As the cost $w(D^*)$ equals $2w(\mathrm{lsp}_G(u, v))$ as well, the claim follows. □ □

4 Computing an MCB in Weighted Partial 2-Trees

As we now have all the technical tools at hand, we can finally give a detailed description of our algorithm, whose pseudo-code can be found in Algorithm 1.

We first compute a suitable tree decomposition T of the 2-connected weighted partial 2-tree G. According to Lemma 11, this can be done in linear time. We then compute the set L of long edges, i.e., the edges whose length is greater than the length of a shortest path between their two endpoints (cf. Lemma 9). By using distance queries between the endpoints of every edge, the data structure of

Algorithm 1: A linear time algorithm to compute a minimum cycle basis of a weighted 2-connected partial 2-tree G

```
1  Compute a suitable tree decomposition T of G;
2  Find the set L of long edges in G;
3  E ← E \ L;
4  for each internal bag Y_1 ∈ T (in any order) and every u, v ∈ Y_1 do
5      Let Y_2, ... Y_k be the children of Y_1 such that for 2 ≤ i ≤ k, Y_1 ∩ Y_i = {u, v};
6      if k ≥ 3 then
7          for 2 ≤ i ≤ k do delete the link (Y_1, Y_i);
8          if {u, v} ∉ E then
9              Compute the weight w(P) of a shortest path P between u and v;
10             Find an intermediate vertex y of P and a bag B containing y;
11             Compute j such that either j ∈ {2, ..., k} and the subtree rooted at Y_j contains B or
               j = 1 otherwise (indicating that B is in the subtree containing Y_1);
12             for 1 ≤ h ≠ j ≤ k do
13                 Add the new edge green(u, v) to Y_h and assign to it weight w(P);
14 Let T̃_1, ..., T̃_r be the connected components of T;
15 Obtain the outerplanar graphs G̃_1, ..., G̃_r that correspond to T̃_1, ..., T̃_r;
16 Compute LSC(G̃_1), ..., LSC(G̃_r) using [18];
17 Output (in an implicit or explicit representation):
   MCB(G) = expand(LSC(G̃_1)) ⊎ ··· ⊎ expand(LSC(G̃_r)) ⊎ {{e} ∪ sp(u, v) | e = {u, v} ∈ L};
```

Lemma 18 allows to do this in linear time. We delete the edges in L and consider thus $G \setminus L$ until Line 15 of Algorithm 1 is reached; clearly, T is still a suitable tree decomposition of $G \setminus L$.

As described in the high-level overview, we decompose the graph iteratively into outerplanar graphs along $K_{2,3}$-subdivisions (cf. Section 3.2). Algorithmically, Corollary 15 allows us to detect efficiently whether the current graph contains a $K_{2,3}$-subdivision: We just have to check whether T contains at least three bags each of which contains the same two vertices u and v. Since T is suitable, this can be done by fixing every inner vertex Y_1 of T (in any order) and counting the number k of children of Y_1 whose links are labeled identically, say with $\{u, v\}$ (cf. the first three lines of the main loop of Algorithm 1). If $k \geq 3$, we have identified a $K_{2,k}$-subdivision by Lemma 14.

If $\{u, v\} \in E$ (the existence of such an edge can be efficiently looked up by a table of size $O(n)$ with Lemma 17), we can simply decompose the graph into the parts defined by decomp by deleting k links in T (cf. Line 7). Otherwise, we additionally fix a shortest path P between u and v and augment all parts except the one containing P with a green edge that replaces P. For this purpose we have to find the weight of P and the part that contains P. The first can be done in constant time by using a distance query of Lemma 18 (cf. Line 9). The latter is computed by identifying the subtree of the tree decomposition (the one after deleting the links) that contains an intermediate vertex y of P (cf. Line 11). Such a vertex can be computed in constant time, using once more the data structure of Lemma 18.

Finally, we end up with several components $\tilde{T}_1, \ldots, \tilde{T}_r$ of the original tree decomposition T; these are easy to find in linear time, e.g., by depth-first search. We can compute the graphs $\tilde{G}_1, \ldots, \tilde{G}_r$ that are represented by these tree decompositions in linear total time by simply collecting the vertices and edges in all bags. Note that the total number of edges in $\tilde{G}_1, \ldots, \tilde{G}_r$ is still in $O(n)$, as we add at most $\deg(V_1)$ new green edges for each bag V_1, where $\deg(V_1)$ is the degree of bag V_1 in T. Every G_i is outerplanar; thus, we can compute the LSC of every G_i in linear time (cf. Theorem 8). According to Lemmata 9 and 20, the output in Line 17 is then a minimum cycle basis of G.

This concludes the first part of our main result, Theorem 1. It remains to clarify how $\mathrm{MCB}(G)$ is represented in the output. For an implicit representation, we store G, L, $\mathrm{LSC}(\tilde{G}_1), \ldots, \mathrm{LSC}(\tilde{G}_r)$ and a trace of the main loop of Algorithm 1. Clearly the space consumption is in $O(n)$. For every long edge $e = \{u, v\}$ in L, we can compute an arbitrary shortest path between u and v in G in time proportional to its length; as the choice of this path does not matter due to Lemma 9, this will complete every long edge to a cycle of an $\mathrm{MCB}(G)$. The trace stores every decision that was made in the decomposition

decomp($G \setminus L$). It thus allows to reconstruct the whole decomposition in linear time, as Algorithm 1 takes linear time.

In particular, we can identify for every decomposition step that is performed on a graph H, the part $H_{j(u,v)}$ of decomp(H, u, v) in which we had chosen the shortest path sp(u, v) between u and v, whose length we computed. An explicit representation of expand(LSC($\tilde{G}_1$)) $\uplus \cdots \uplus$ expand(LSC($\tilde{G}_r$)) is then computed by constructing the shortest path sp(u, v) for every decomposition step explicitly via Lemma 18 (in contrast to computing only its length and an intermediate vertex, as in the original decomposition).

According to Lemma 18, computing these shortest paths takes time proportional to the number of edges in these paths. This gives the desired running time of $O(\text{size}(\text{MCB}(G)))$, where size(MCB($G$)) is the number of edges in MCB(G) counted according to their multiplicity.

5 Discussion

We have shown that an implicit representation of a minimum cycle basis of a weighted partial 2-tree can be computed in linear time. It remains a challenging question if our result can be extended to partial k-trees for $k > 2$. We remark that it was noted in [19] that already for partial 3-trees the set of lex short cycles do not necessarily form a minimum cycle basis. Since in particular the proof of Theorem 20 is based on this, extending our result to partial 3-trees may therefore require substantially new ideas.

Acknowledgments.

We would like to thank Geevarghese Philip for pointing us to Lemma 12, which significantly simplified our proof of the runtime bound. We also thank the anonymous reviewers for providing their feedback which has helped us to improve the presentation of our work.

Carola Doerr gratefully acknowledges support from the Alexander von Humboldt Foundation and the Agence Nationale de la Recherche (project ANR-09-JCJC-0067-01).

G. Ramakrishna would like to thank his adviser N.S. Narayanaswamy for fruitful discussions during the early stages of this work. He would also like to thank Kurt Mehlhorn for providing him the opportunity to do an internship at the Max Planck Institute for Informatics.

References

[1] E. Amaldi, C. Iuliano, T. Jurkiewicz, K. Mehlhorn, and R. Rizzi. Breaking the $O(m^2n)$ barrier for minimum cycle bases. In *Proc. of the 17th Annual European Symposium on Algorithms (ESA'09)*, volume 5757 of *Lecture Notes in Computer Science*, pages 301–312. Springer, 2009.

[2] E. Amaldi, C. Iuliano, and R. Rizzi. Efficient deterministic algorithms for finding a minimum cycle basis in undirected graphs. *Proc. of the 14th International Conference on Integer Programming and Combinatorial Optimization (IPCO'10)*, 6080:397–410, 2010.

[3] V. Arvind, B. Das, and J. Köbler. A logspace algorithm for partial 2-tree canonization. In *CSR*, pages 40–51, 2008.

[4] H. L. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM Journal of Computing*, 25(6):1305–1317, 1996.

[5] H. L. Bodlaender. A partial k-arboretum of graphs with bounded treewidth. *Theoretical Computer Science*, 209(1-2):1–45, 1998.

[6] J. A. Bondy and U. S. R. Murty. *Graph Theory*. Springer, 2008.

[7] G. Borradaile, P. Sankowski, and C. Wulff-Nilsen. Min st-cut oracle for planar graphs with near-linear preprocessing time. In *Proc. of the 51th Annual IEEE Symposium on Foundations of Computer Science (FOCS'10)*, pages 601–610. IEEE Computer Society, 2010.

[8] G. Borradaile, P. Sankowski, and C. Wulff-Nilsen. Min st-cut oracle for planar graphs with near-linear preprocessing time. *CoRR*, abs/1003.1320, 2013. To appear in ACM Transactions of Algorithms. Available online at `http://arxiv.org/abs/1003.1320`.

[9] S. Chaudhuri and C. D. Zaroliagis. Shortest paths in digraphs of small treewidth. Part I: Sequential algorithms. *Algorithmica*, 27(3):212–226, 2000.

[10] R. Diestel. *Graph Theory*. Springer, fourth edition, 2010.

[11] C. Doerr, G. Ramakrishna, and J. M. Schmidt. Computing minimum cycle bases in weighted partial 2-trees in linear time. In *Proc. of the 39th International Workshop on Graph-Theoretic Concepts in Computer Science (WG'13)*, volume 8165 of *Lecture Notes in Computer Science*, pages 225–236. Springer, 2013.

[12] G. N. Frederickson and R. Janardan. Designing networks with compact routing tables. *Algorithmica*, 3:171–190, 1988.

[13] D. Hartvigsen and R. Mardon. The all-pairs min cut problem and the minimum cycle basis problem on planar graphs. *SIAM Journal on Discrete Mathematics*, 7:403–418, 1994.

[14] C.-W. Ho, S.-Y. Hsieh, and G.-H. Chen. An efficient parallel strategy for computing k-terminal reliability and finding most vital edges in 2-trees and partial 2-trees. *Journal of Parallel and Distributed Computing*, 51(2):89 – 113, 1998.

[15] J. D. Horton. A polynomial-time algorithm to find the shortest cycle basis of a graph. *SIAM Journal of Computing*, 16:358–366, 1987.

[16] T. Kavitha, C. Liebchen, K. Mehlhorn, D. Michail, R. Rizzi, T. Ueckerdt, and K. A. Zweig. Cycle bases in graphs characterization, algorithms, complexity, and applications. *Computer Science Review*, 3(4):199–243, 2009.

[17] J. Leydold and P. F. Stadler. Minimal cycle bases of outerplanar graphs. *Electronic Journal of Combinatorics*, 5, 1998.

[18] T. Liu and H. Lu. Minimum cycle bases of weighted outerplanar graphs. *Information Processing Letters*, 110:970–974, 2010. A preliminary report appeared in Proc. of the 20th International Symposium on Algorithms and Computation (ISAAC'09).

[19] N. S. Narayanaswamy and G. Ramakrishna. Characterization of minimum cycle bases in weighted partial 2-trees. In *Proc. of the 11th Cologne-Twente Workshop on Graphs and Combinatorial Optimization (CTW'12)*, pages 193–196, 2012. Available online at `http://arxiv.org/abs/1302.5889`. All references in this paper refer to this arXiv version.

[20] P. Ochem and A. Pinlou. Oriented colorings of partial 2-trees. *Information Processing Letters*, 108(2):82 – 86, 2008.

[21] A. Proskurowski, M. M. Sysło, and P. Winter. Plane embeddings of 2-trees and biconnected partial 2-trees. *SIAM Journal on Discrete Mathematics*, 9(4):577–596, 1996.