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

Carola Doerr[1,2], G. Ramakrishna[3⋆], Jens M. Schmidt[2]

[1] Université Paris Diderot - Paris 7, LIAFA, Paris, France
[2] Max Planck Institute for Informatics, Saarbrücken, Germany
[3] Indian Institute of Technology Madras, India

Abstract. We present a linear time algorithm for computing an implicit linear space representation of a minimum cycle basis (MCB) in weighted partial 2-trees, i.e., graphs of treewidth two. The implicit representation can be made explicit in a running time that is proportional to the size of the MCB.
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 MCB 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 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, cf. [12] for an exhaustive survey. It is therefore—both from a practical and a theoretical viewpoint—an interesting task to compute them 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 [12]. In all previous work that we are aware of it is therefore assumed that the edge-weights are non-negative.)

1.1 Previous Work

The first polynomial time algorithm for computing MCBs was presented by Horton [11] in 1987. His algorithm has running time $O(m^3 n)$. This was improved subsequently in a series of papers by different authors, cf. [12] or [14] 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 [10], which is of order $n^2 \log n$. The currently best known algorithm on planar graphs is due to Borradaile, Sankowski, and Wulff-Nilsen [6]. It constructs an $O(n \log n)$ space implicit representation of an MCB in planar graphs in time $O(n \log^5 n)$.

Faster algorithms for planar graphs are known only for the special case of outerplanar graphs. For unweighted outerplanar graphs, Leydold, and Stadler [13] presented a linear time algorithm. More recently, Liu and Lu [14] presented a linear time, linear space algorithm to compute an MCB of a weighted outerplanar graph (using an implicit representation). This is optimal in terms of both time and space.

⋆ Work initiated while visiting the Max Planck Institute for Informatics during an internship.

1.2 Our Result

In this contribution, we present a linear time algorithm for computing an implicit $O(n)$-space representation of a minimum cycle basis in partial 2-trees (graphs of treewidth two). 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. That is, for the special case of partial 2-trees we are able to improve their running time by a factor of $\Theta(\log^5 n)$.

The class of partial 2-trees subsumes, in particular, the class of outerplanar graphs.

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 [14] can be applied. We state our main theorem.

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

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

Note in Theorem 1 that, although B has an *implicit* representation of linear size, the *explicit size* of B may be quadratic. This is true already for outerplanar graphs, cf. [14] for a simple grid 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 [15]. 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 endpoints). In outerplanar graphs, such a data structure exists due to Frederickson and Jannardan [9]. For our more general case, we will instead extend a result of Chaudhuri and Zaroliagis [7, Lemma 3.2] on weighted partial k-trees, which is able to give the distance of every two vertices in constant time, as long as both are contained in one bag of a tree decomposition. Using this extension, we can report the 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. Throughout this work, we set $n = |V|$ and $m = |E|$. All graph classes 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 in X disconnects the graph G. 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. 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 non-empty and pairwise vertex-disjoint paths. In this work, we will be mainly concerned with $K_{2,3}$-subdivisions. In such a $K_{2,3}$-subdivision, we call the 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 $\mathcal{S} := C_1 \oplus \ldots \oplus C_k$ is the set of edges appearing an odd number of times in the multi-set $\{C_1, \ldots, C_k\}$. It is well known that $\mathcal{S}$ is a union of cycles in G.

Let a set $\mathcal{C} = \{C_1, \ldots, C_k\}$ of cycles of G *span 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 $\text{size}(\mathcal{C})$ of $\mathcal{C}$ is the number of edges in $\mathcal{C}$ 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. For any simple weighted graph the dimension equals $m - n + 1$ [5].

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

A minimum cycle basis of a graph G is equal to the disjoint union of the minimum cycle basis of the 2-connected components of G. Therefore, throughout this paper, we 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. Any subgraph of a k-tree is called a *partial k-tree*. Partial 2-trees are also known as graphs in which every biconnected component is a *series-parallel* graph [4]. The partial 2-trees form a strict superclass of outerplanar graphs, as outerplanar graphs are characterized by the forbidden minor set $\{K_4, K_{2,3}\}$, while partial 2-trees have the forbidden minor set $\{K_4\}$. Equivalently, a partial 2-tree is outerplanar if and only if it does not contain a $K_{2,3}$-subdivision (as a subgraph). We will need the following somewhat stronger statement.

Lemma 2 (Lemma 2.4 in [15]). *Let G be a weighted partial 2-tree. G is not outerplanar if and only G contains a $K_{2,3}$-subdivision with branch vertices u and v such that $G - \{u, v\}$ has at least 3 connected components.*

2.3 Lex Shortest Paths and Lex Short Cycles

It is known (Proposition 4.5 in [10]) that for any edge-weighted simple graph G the set of so-called *lex short cycles* contains a minimum cycle basis. For outerplanar graphs [14] and partial 2-trees [15], the whole set of lex short cycles forms a minimum cycle basis.

Definition 3 (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 $(w(P') = w(P)$ and $|E(P')| > |E(P)|)$ or $(w(P') = w(P)$, $|E(P')| = |E(P)|$ and $\min_{y \in V(P') \setminus V(P)} \sigma(y) > \min_{y \in V(P) \setminus V(P')} \sigma(y))$ holds.*

It is easily verified (cf. Proposition 4.1 in [10]) that between any two vertices u, v in G there is exactly one lex shortest path. We refer to this path as $\mathrm{lsp}(u, v)$. 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 4 (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 5 ([10, 15]). *For any edge-weighted simple graph G, there is a set $B \subseteq \mathrm{LSC}(G)$ such that 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 5. This lemma allows us to restrict ourselves to compute the set of lex short cycles. For outerplanar graphs, Liu and Lu showed that an implicit representation of $\mathrm{LSC}(G)$ can be computed in linear time.

Theorem 6 ([14]). *For any weighted outerplanar graph G an $O(n)$-space representation of the set $\mathrm{LSC}(G)$ can be computed in $O(n)$ time. From this representation, we can compute a cycle $C \in \mathrm{LSC}(G)$ explicitly in time $O(\mathrm{size}(C))$.*

3 High-Level Overview of Our Algorithm and Technical Details

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

3.1 Preprocessing Steps

We introduce a few preprocessing steps besides 2-connectivity that will simplify the description of our algorithm. As a first step, we construct an alternative to the weight function w for which every lex shortest path is the unique shortest path. That this can be done in linear time has been shown in [10].

Lemma 7 (Proposition 4.3 in [10]). *If G is a simple graph with edge weight function w, then there exists a perturbation $\hat{w}$ of w such that every lex shortest path $\mathrm{lsp}(u, v)$ under w is the unique shortest $u - v$ path under $\hat{w}$.*

We call an edge $\{u, v\}$ *tight* if it is the unique shortest path between u and v, and we call it *long* otherwise. The long edges in G will be treated separately. This is based on the following lemma.

Lemma 8. *Let $G = (V, E)$ be an edge-weighted graph. Let L be the set of long edges in G. Then $\mathrm{MCB}(G) = \mathrm{MCB}(G \setminus L) \cup \{\{e\} \cup \mathrm{lsp}(u, v) \mid e = \{u, v\} \in L\}$.*

Proof. For every $e = \{u, v\} \in L$ the cycle $\{e\} \cup \mathrm{lsp}(u, v)$ is a lex short cycle in G. By definition, it is the only lex short cycle that contains e. Therefore, $\mathrm{LSC}(G) = \mathrm{LSC}(G \setminus L) \cup \{e' \cup \mathrm{lsp}(u', v') \mid e' = \{u', v'\} \in L\}$. By Lemma 5 we know that $\mathrm{MCB}(G) \subseteq \mathrm{LSC}(G)$. The independence of $\mathcal{M} := \mathrm{MCB}(G \setminus L) \cup \{e' \cup \mathrm{lsp}(u', v') \mid e' = \{u', v'\} \in L\}$ follows again from the fact that e is contained in $\mathcal{M}$ only in this one cycle $\{e\} \cup \mathrm{lsp}(u, v)$. □

As we shall see in Lemma 15 below, the set L in Lemma 8 can be identified and removed from G in $O(n)$ time using a suitable data structure. Using these two lemmata, computing the contribution of the removed long edges to the MCB of G takes at most linear time.

3.2 High-Level Overview of the Main Algorithm

By Lemma 5 it suffices to compute the set of lex short cycles in G. This set forms a minimum cycle basis. For simplicity, we assume that the simplifications described above have been performed in a preprocessing step. In particular, all shortest paths in G are unique and G contains only tight edges. As shown above, the preprocessing steps can be done in time $O(n)$.

The key approach for our algorithm is to iteratively decompose the graph G into outerplanar subgraphs $G_1, \ldots, G_r$. To these subgraphs we apply the linear time algorithm of Liu and Lu (Theorem 6). Intuitively, the decomposition is done as follows.

When G is not outerplanar, then there exists a $K_{2,3}$-subdivision in G with branch vertices u and v such that (i) $\{u, v\}$ is a minimum vertex separator of G and (ii) the removal of $\{u, v\}$ disconnects G into at least three connected components $H_1, \ldots, H_k$ (cf. Lemma 2). We distinguish two cases. If $\{u, v\} \in E$, we set $G_i := G[V(H_i) \cup \{u\} \cup \{v\}]$. Otherwise, let $j \in \{1, 2, \ldots, k\}$ such that $\mathrm{lsp}(u, v) \in G[V(H_j) \cup \{u\} \cup \{v\}]$. We set $G_j := G[V(H_j) \cup \{u\} \cup \{v\}]$, and for all $1 \leq i \neq j \leq k$ we set $G_i := G[V(H_i) \cup \{u\} \cup \{v\}] \cup \mathrm{blue}(\{u, v\})$, where $\mathrm{blue}(\{u, v\})$ denotes a new "colored" (i.e., marked) edge $\{u, v\}$. This *blue* edge serves as a placeholder for the lex shortest path between u and v, as this is not contained in G_i. The weight

$w(\text{blue}(\{u,v\}))$ assigned to this new edge is therefore $w(\text{lsp}(u,v))$. Let the operation $\text{decomp}(G,u,v)$ decompose G into $G_1,\dots,G_k$ with respect to the vertices u,v.

We now iteratively decompose the graphs $G_1,\dots,G_k$ as described above until we are left with graphs $G_1,\dots,G_r$ that do not contain any $K_{2,3}$-subdivision, i.e., with outerplanar graphs according to Lemma 2. Since all the edges in G_i are tight, the set of lex short cycles in G_i equals the boundaries of its internal faces. Extracting the internal faces of G_i can be done in linear time, cf. [14]. We will show in Lemma 17 that $\text{LSC}(G)$ equals the disjoint union of $\text{expand}(\text{LSC}(G_1)),\dots,\text{expand}(\text{LSC}(G_r))$, where $\text{expand}(\text{LSC}(G_i))$ replaces the blue edges $\text{blue}(\{u,v\})$ in every cycle by the lex shortest path $\text{lsp}(u,v)$.

The challenging part of the algorithm is to find a data structure that allows to identify all the $K_{2,3}$-subdivisions and to do the respective decompositions 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. Let the *label* of a link $\ell = (X,Y)$ in a tree decomposition be $X \cap Y$.

Definition 9 (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 [3]).*
2. *Any two links with the same label have a common parent in T.*

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 10. *Given a weighted partial 2-tree G, a suitable tree decomposition can be computed in linear time and has linear space.*

Proof. The number of bags of a smooth tree decomposition of a partial 2-tree G is $n-2$ (see Lemma 2.5 in [3]). As every bag contains exactly 3 vertices, this gives a linear space representation of the tree decomposition. A rooted tree decomposition $(\{X_1,\dots,X_r\},T)$ of G that is additionally smooth can be computed using Bodlaender's algorithm [3]. We make T suitable (i.e., we add Property 9.2) 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 $\{B,C\}$ that have the same label satisfy that A is a parent of B and B is a parent of C.

Traverse T in any order that starts on the root and in which parents precede their children. Whenever a bag B with parent 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 parent. 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 underlying key observations of our algorithm is the fact that for all $K_{2,3}$-subdivisions the two branch vertices must be contained in a common bag of a suitable tree decomposition. This is shown using the following result.

Lemma 11 (Lemma 12.3.4 in [8]). *Let $W \subseteq V(G)$ and 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 12 ($K_{2,3}$-subdivisions in partial 2-trees). *For every $K_{2,3}$-subdivision H in a partial 2-tree G and every smooth tree decomposition T of G, the two branch vertices u and v of H are contained in a common bag of T.*

Proof. We apply Lemma 11 with $W = \{u,v\}$. If u and v are not contained in a common bag, there must be a link $\{X_1,X_2\}$ such that u and v are separated by $X_1 \cap X_2$ in G. Since H is a $K_{2,3}$-subdivision, $|X_1 \cap X_2| \geq 3$, contradicting the smoothness of T. □

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 the lex shortest path between two nodes. The following lemma is along the lines of [9].

Lemma 13. *Let T be a rooted 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 the subtree of T containing a bag with vertex 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 its root, and label every bag X and its vertices with a dfs-number. 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. These numbers form an interval (taking constant space), which we can store at X during the depth first search, i.e., in linear total time. The desired answer for the query is then obtained by performing a binary search on the neighbors of A (in the same order as the dfs) that searches for the neighbor with the interval containing the label of v. This takes time $O(\log d)$. □

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

Lemma 14 (Lemma 3.2 in [7]). *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 the shortest path between the vertices.*

Lemma 14 is originally stated for directed graphs in [7]. However, representing each undirected edge with two edges oriented in opposite directions gives the above undirected variant.

We extend Lemma 14 by giving the following data structure.

Lemma 15. *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 distance of the shortest path between u and v (*distance query*).*
- *Compute in time $O(1)$ an intermediate vertex w of the shortest path between u or v (if exists) and a bag $Y \in T$ such that $Y = \{u, v, w\}$ (*intermediate vertex query*)*
- *Compute in time $O(|E(P)|)$ the shortest path P between u and v (*shortest path extraction*)*

Proof. We perform the algorithm of Lemma 14 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 the shortest path 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 13 for the tree decomposition T in time $O(n)$ and apply the algorithm of Lemma 14 in time $O(n)$. Let X be a bag containing u and v. By Lemma 14, 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, the shortest path 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 parent of X in T). According to Lemma 13, 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 further 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 the 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. □

Since a tree decomposition maintains for every edge the bag that contains it, the query of Lemma 15 can also be performed when no bag is given but an edge $\{u, v\}$.

3.5 Obtaining LSC(G) from LSC(G_1), ..., LSC(G_r)

As a last technicality, we show that—as claimed in the high-level overview of our algorithm—the set of lex short cycles in G equals the disjoint union $\text{expand}(\text{LSC}(G_1)) \uplus \ldots \uplus \text{expand}(\text{LSC}(G_r))$. This follows from iteratively applying Lemma 17 below.

Definition 16. *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* $\text{expand}(\mathcal{C}) := \{\text{expand}(C) \mid C \in \mathcal{C}\}$.

As mentioned above, we obtain a minimum cycle basis of G by expanding the cycles in the MCBs of the subgraphs.

Lemma 17. *Let G be a weighted partial 2-tree in which every edge is tight. Let u and v be the two branch vertices of a $K_{2,3}$-subdivision in G. Let $k \geq 3$ and let $G_1, \ldots, G_k$ be the subgraphs resulting from the decomposition* $\text{decomp}(G, u, v)$*. Then* $\text{LSC}(G) = \text{expand}(\text{LSC}(G_1)) \uplus \ldots \uplus \text{expand}(\text{LSC}(G_k))$.

The proof of Lemma 17 is based on the following observation.

Lemma 18 (Lemma 2.5 and Corollary 2.8 in [15]). *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 17, we have $\text{expand}(\text{LSC}(G_i)) \subseteq \text{LSC}(G)$*, $1 \leq i \leq k$.*

Proof (of Lemma 17). 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 $\bigcup_{i=1}^{k} \text{expand}(\text{LSC}(G_i)) \subseteq \text{LSC}(G)$ follows from Lemma 18.

It thus remains to show $\text{LSC}(G) \subseteq \bigcup_{i=1}^{k} \text{expand}(\text{LSC}(G_i))$. 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_i; Lemma 18 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 $\text{LSC}(G_i)$.

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_i.

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_i. Since G_i 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_i. As mentioned above, by Lemma 18 it follows that $C \in \text{expand}(\text{LSC}(G_i))$. □

4 Computing an MCB in Weighted Partial 2-Trees

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

```
1  Do the graph preprocessing steps described in Section 3.1;
2  Compute a suitable tree decomposition of G;
3  for each internal bag Y_1 ∈ V(T) and every u, v ∈ Y_1 do
4      Let Y_2, ... Y_k be the children of Y_1 such that for 2 ≤ i ≤ k, Y_1 ∩ Y_i = {u, v};
5      if k ≥ 3 then
6          for 2 ≤ i ≤ k do remove the link {Y_1, Y_i};
7          if {u, v} ∉ E then
8              Find an intermediate vertex y in lsp(u, v) and compute w(lsp(u, v)) ;
9              if there exists a j ∈ {2, ..., k} such that the subtree rooted at Y_j has a bag that
               contains the vertex y then let j be that index ; else j = 1;
10             for 1 ≤ i ≠ j ≤ k do
11                 Add the new blue edge blue({u, v}) to Y_i and assign to it weight w(lsp(u, v));
12 Let T_1, ..., T_r be the connected components of T;
13 Obtain the graphs G_1, ..., G_r that correspond to the tree decompositions T_1, ..., T_r, respectively;
14 Compute LSC(G_1), ..., LSC(G_r) using [14] ; // LSC(G_i) equals the internal faces of G_i
15 Store (L, LSC(G_1), ..., LSC(G_r)) ; // L is the set of long edges
```

Our algorithm, Algorithm 1, can now be described as follows.

Preprocessing. Given a 2-connected weighted partial 2-tree $G = (V, E)$ with edge-weight function $\hat{w}$, we first compute the perturbation w of $\hat{w}$ such that every lex shortest path is the unique shortest path (cf. Lemma 7). We then identify the set L of long edges; i.e., the set of edges that are not the shortest paths between their two endpoints (cf. Lemma 8), and, for the moment, we remove these edges from G.

For the graph obtained from this preprocessing step, we compute a suitable tree decomposition.

Main Procedure. In the main loop of Algorithm 1, we check for every vertex pair $\{u, v\}$ in every bag if $\{u, v\}$ is a vertex separator that decomposes G into at least three different components. By the Lemmata 2 and 12 and the fact that G is assumed to be 2-connected, this identifies a $K_{2,3}$-subdivision if it exists. If so, we decompose along u and v by deleting the appropriate links in T in line 6. Lines 8 and 9 are needed to identify the subtree of T that contains $\text{lsp}(u, v)$. In all other subtrees, the edge $\{u, v\}$ is marked in blue, and the weight associated to this edge is $w(\text{lsp}(u, v))$. By maintaining the data structure of Lemma 15, the intermediate vertex can be found efficiently.

When Algorithm 1 stops, we compute the connected components $T_1, \ldots, T_r$ of T in linear time by performing a standard graph traversal routine. We can compute the graphs $G_1, \ldots, G_r$ that are represented by these tree decompositions in linear total time by just collecting the vertices and edges in all bags. Note that the total number of edges in $G_1, \ldots, G_r$ is linear, as we add at most $\deg(V_1)$ new blue edges for each bag V_1, where $\deg(V_1)$ is the degree of bag V_1 in T. Every G_i is outerplanar and the LSCs of all these outerplanar graphs can be computed in linear total time. (As mentioned in the high-level overview, one can simply apply the result of Liu and Lu (Theorem 6) as a black-box. Or one observes that since all edges in G_i are tight, the set of lex short cycles in G_i equals the boundaries of its internal faces. Extracting the internal faces of G_i can be done in linear time.)

The set L of long edges together with (the implicit representations) of $\text{LSC}(G_1), \ldots, \text{LSC}(G_r)$ forms the implicit representation of our minimum cycle basis.

This concludes the presentation of our main result, the first part of Theorem 1. In order to get the explicit representation of the minimum cycle basis, we apply Lemma 8. It thus suffices to augment every edge (u, v) in the set L of long edges of the implicit representation by $lsp(u, v)$, which can be done in time $O(|C|)$ for each constructed cycle C, according to Lemma 15.

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 [15] that already for partial 3-trees the set of lex short cycles do not necessarily form a minimum cycle basis. Extending our result to partial 3-trees may therefore require substantially new ideas.

Acknowledgments. Carola Doerr is supported by a Feodor Lynen postdoctoral research fellowship of the Alexander von Humboldt Foundation and by the Agence Nationale de la Recherche under the 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.

We would like to thank Geevarghese Philip for pointing us to Lemma 11, which greatly simplifies our proof of the runtime bound.

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. H. Bodlaender. A linear-time algorithm for finding tree-decompositions of small treewidth. *SIAM Journal of Computing*, 25(6):1305–1317, 1996.
4. H. L. Bodlaender. A partial k-arboretum of graphs with bounded treewidth. *Theoretical Computer Science*, 209(1-2):1–45, 1998.
5. J. A. Bondy and U. S. R. Murty. *Graph Theory*. Springer, 2008.
6. 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.
7. S. Chaudhuri and C. D. Zaroliagis. Shortest paths in digraphs of small treewidth. Part I: Sequential algorithms. *Algorithmica*, 27(3):212–226, 2000.
8. R. Diestel. *Graph Theory*. Springer, fourth edition, 2010.
9. G. N. Frederickson and R. Janardan. Designing networks with compact routing tables. *Algorithmica*, 3:171–190, 1988.
10. 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.
11. J. D. Horton. A polynomial-time algorithm to find the shortest cycle basis of a graph. *SIAM Journal of Computing*, 16:358–366, 1987.
12. 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.
13. J. Leydold and P. F. Stadler. Minimal cycle bases of outerplanar graphs. *Electronic Journal of Combinatorics*, 5, 1998.
14. 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).
15. 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.