

A LINEAR-TIME ALGORITHM FOR PRODUCT STRUCTURE IN PLANAR GRAPHS[‡]

Prosenjit Bose^δ Kaya Gouin^δ Pat Morin^δ Saeed Odak^σ

ABSTRACT. The *Product Structure Theorem* for planar graphs (Dujmović et al. *JACM*, **67**(4):22) states that any planar graph is contained in the strong product of a planar 3-tree, a path, and a 3-cycle. We give a simple linear-time algorithm for finding this partition as well as several related partitions. This improves on the previous $O(n \log n)$ time algorithm for finding this partition (Morin. *Algorithmica*, **85**(5):1544–1558). The algorithm is practical; we provide open-source C and C++ implementations that, even on modest hardware, can process graphs with millions of vertices in a matter of seconds.

1 Introduction

For two graphs G and X , the notation $G \subseteq X$ denotes that G is isomorphic to some subgraph of X . The following *planar product structure theorems* have recently been used as a key tool in resolving a number of longstanding open problems on planar graphs, including queue number [10], nonrepetitive chromatic number [13], adjacency labelling [12], universal graphs [15], p -centered colouring [14], and vertex ranking [7].^{1,2}

Theorem 1 (Dujmović et al. [10], Ueckerdt, Wood, and Yi [26]). *For any planar graph G , there exists:*

- (a) *a planar graph H of treewidth at most 3 and a path P such that $G \subseteq H \boxtimes P \boxtimes K_3$ [10];*
- (b) *a planar graph H of treewidth at most 4 and a path P such that $G \subseteq H \boxtimes P \boxtimes K_2$; and*
- (c) *a planar graph H of treewidth at most 6 and a path P such that $G \subseteq H \boxtimes P$ [26].*

In the applications of [Theorem 1](#), the proofs are constructive and lead to algorithms whose running time is dominated by the time required to compute the relevant partition. Indeed, the proofs of each part of [Theorem 1](#) lead to $O(n^2)$ time algorithms, as observed

[‡]This research was partly funded by NSERC. This paper is the journal-length version of the conference paper [8]. This version provides a much more detailed description of the algorithm that covers a number of technicalities and implementation details that were previously omitted. It also reports on open-source C/C++ implementations of the algorithm.

^δSchool of Computer Science, Carleton University

^σDepartment of Computer Science, Aalto University

¹For graphs G and X , an X -decomposition of G is a collection $\mathcal{X} := (B_x : x \in V(X))$ of subsets of $V(G)$ called *bags* indexed by the vertices of X and such that (i) for each $v \in V(G)$, $X[\{x \in V(X) : v \in B_x\}]$ is connected; and (ii) for each $vw \in E(G)$, there exists some $x \in V(X)$ such that $\{v, w\} \subseteq B_x$. The *width* of $\mathcal{X}$ is $\max\{|B_x| : x \in V(X)\} - 1$. In the special case where X is a tree, $\mathcal{X}$ is called a *tree decomposition* of G . The *treewidth* $\text{tw}(G)$ of G is the minimum width of any tree decomposition of G .

²For two graphs G_1 and G_2 , the *strong graph product* of G_1 and G_2 , denoted $G_1 \boxtimes G_2$, is a graph whose vertex set is $V(G_1 \boxtimes G_2) := V(G_1) \times V(G_2)$ and that contains an edge between distinct vertices $v = (v_1, v_2)$ and $w = (w_1, w_2)$ if and only if (i) $v_1 = w_1$ and $v_2 w_2 \in E(G_2)$; (ii) $v_2 = w_2$ and $v_1 w_1 \in E(G_1)$; or (iii) $v_1 w_1 \in E(G_1)$ and $v_2 w_2 \in E(G_2)$.

already by Dujmović et al. [10]. Morin [22] later showed that there exists an $O(n \log n)$ time algorithm to find the partition in [Theorem 1.a](#). In the current note, we show that there exists a linear time algorithm for finding each of the three partitions guaranteed by [Theorem 1](#). This immediately gives an $O(n)$ -time algorithm for each of the following problems on any n -vertex planar graph G :

- computing an $O(1)$ -queue layout of G [10];
- nonrepetitively vertex-colouring G with $O(1)$ colours [13];
- assigning $(1 + o(1)) \log n$ -bit labels to the vertices of G so that one can determine from the labels of vertices v and w whether or not v and w are adjacent in G [12];
- mapping the vertices of G into a universal graph U_n that has $n^{1+o(1)}$ vertices and edges so that any pair of vertices that are adjacent in G maps to a pair of vertices that are adjacent in U_n [15];
- colouring the vertices of G with $O(p^3 \log p)$ colours so that each connected subgraph H of G contains a vertex whose colour is unique in H or contains vertices of at least $p + 1$ different colours [14]; and
- colouring the vertices of G with $O(\log n / \log \log \log n)$ integers so that the maximum colour that appears on any path P of length at most ℓ appears at exactly one vertex of P (for any fixed $\ell \geq 2$) [7].

In addition to planar graphs, product structure theorems similar to [Theorem 1](#) exist for k -planar graphs, h -framed graphs, bounded-genus graphs, and graphs from apex-minor-free families [3, 10, 11]. The existence of each of these partitions relies at some point on [Theorem 1](#). Thus, the algorithm presented here can also serve as an optimal subroutine in the computation of product structure partitions of graphs from these classes. However, these larger graph classes also require additional machinery that (at the time of writing) makes finding these partitions largely impractical.³

The remainder of this paper is organized as follows: [Section 2](#) presents some necessary background and notation. [Section 3](#) reviews the proof of [Theorem 1.a](#). [Section 4](#) presents the linear time algorithm for finding the partition in [Theorem 1.a](#). [Section 5](#) describes the algorithms for finding the partitions in [Theorem 1.b](#) and [Theorem 1.c](#). [Section 6](#) reports on the performance of an Open-Source C++ implementation of the algorithm.

2 Preliminaries

Throughout this paper we use standard graph theory terminology as used in the textbook by Diestel [9]. All graphs discussed here are simple and finite. For a graph G , $V(G)$ and $E(G)$ denote the vertex and edge sets of G , respectively. We use the terms *vertex* and *node* interchangeably, though we typically refer to the vertices of some primary graph G of interest and refer to the nodes of some auxiliary graph (such as a spanning tree) related to G . We say that a subgraph G' of a graph G *spans* a set $S \subseteq V(G)$ if $S \subseteq V(G')$.

Quotient Graphs. Given a graph G and a partition $\mathcal{P}$ of $V(G)$, the *quotient graph* $G/\mathcal{P}$ is the graph with vertex set $V(G/\mathcal{P}) := \mathcal{P}$ and in which two nodes $X, Y \in V(G/\mathcal{P})$ are adjacent

³Possible exceptions here are cases in which a bounded-genus graph, k -planar graph, or h -framed graph is given along with its embedding.

if G contains at least one edge xy with $x \in X$ and $y \in Y$.

Embeddings, Planar Graphs, and (Near-)Triangulations. An *embedding* ψ of a graph G associates each vertex v of G with a point $\psi(v) \in \mathbb{R}^2$ and each edge vw of G with a simple open curve $\psi(vw) : (0, 1) \rightarrow \mathbb{R}^2$ whose endpoints⁴ are $\psi(v)$ and $\psi(w)$. We do not distinguish between such a curve $\psi(vw)$ and the point set $\{\psi(vw)(t) : 0 < t < 1\}$. We let $\psi(V(G)) := \{\psi(v) : v \in V(G)\}$, $\psi(E(G)) := \bigcup_{vw \in E(G)} \psi(vw)$, and $\psi(G) := \psi(V(G)) \cup \psi(E(G))$. An embedding ψ of G is *plane* if $\psi(vw) \cap \psi(V(G)) = \emptyset$ and $\psi(vw) \cap \psi(xy) = \emptyset$ for each distinct pair of edges $vw, xy \in E(G)$. A graph G is *planar* if it has a plane embedding. A *triangulation* is an edge-maximal planar graph.

If ψ is a plane embedding of a planar graph G , then we call the pair (G, ψ) an *embedded graph* and we will not distinguish between a vertex v of G and the point $\psi(v)$ or between an edge vw of G and the curve $\psi(vw)$. Similarly, we will not distinguish between G and the point set $\psi(G)$. Any cycle in an embedded graph defines a Jordan curve. For such a cycle C , $\mathbb{R}^2 \setminus C$ has two components, one bounded and the other unbounded. We will refer to the bounded component as the *interior* of C and the unbounded component as the *exterior* of C . If G is an embedded triangulation and C is an arbitrary cycle in G , then the subgraph of G consisting of all edges and vertices of G contained in the closure of the interior of C is called a *near-triangulation*.

Each component of $\mathbb{R}^2 \setminus G$ is a *face* of G and we let $F(G)$ denote the set of faces of G . If G is 2-connected then, for any face $f \in F(G)$, the set of vertices and edges of G contained in the boundary of f forms a cycle. We may therefore treat a face f of a 2-connected graph G as a component of $\mathbb{R}^2 \setminus G$ or as the cycle of G on the boundary of f , relying on context to distinguish between the two usages. Note that every embedded graph contains exactly one face—the *outer face*—that is unbounded.

Duals and Cotrees. The *dual* $G^\star$ of an embedded graph G is the graph with vertex set $V(G^\star) := F(G)$ and edge set $E(G^\star) := \{fg \in \binom{F(G)}{2} : E(f) \cap E(g) \neq \emptyset\}$.⁵ If T is a spanning tree of G then the *cotree* $\bar{T}$ of (G, T) is the graph with vertex set $V(\bar{T}) := V(G^\star)$ and edge set $E(\bar{T}) := \{ab \in E(G^\star) : E(a) \cap E(b) \setminus E(T) \neq \emptyset\}$. It is well known that, if G is connected and T is a spanning tree of G then $\bar{T}$ is a spanning tree of $G^\star$ [9, Chapter 4, Exercise 42].

For our purposes, a *binary tree* is a rooted tree of maximum degree 3 whose root has degree at most 2 and in which each child v of a node u is either the unique *left child* or the unique *right child* of u . If G is a triangulation and we root $\bar{T}$ at any face $f_0 \in F(G)$ that contains an edge of T , then $\bar{T}$ is a binary tree, with the classification of left and right children determined by the embedding of G .⁶

Paths and Distances. A *path* in G is a (possibly empty) sequence of distinct vertices $v_0, \dots, v_r$ with the property that $v_{i-1}v_i \in E(G)$, for each $i \in \{1, \dots, r\}$. The *endpoints* of a path $v_0, \dots, v_r$ are the vertices v_0 and v_r . The *length* of a non-empty path $v_0, \dots, v_r$ is the number,

⁴The *endpoints* of an open curve $\psi : (0, 1) \rightarrow \mathbb{R}^2$ are the two points $\lim_{\epsilon \downarrow 0} \psi(\epsilon)$ and $\lim_{\epsilon \downarrow 0} \psi(1 - \epsilon)$.

⁵For a set S , $\binom{S}{2}$ denotes the $\binom{|S|}{2}$ -element set $\binom{S}{2} := \{\{x, y\} : x, y \in S, x \neq y\}$.

⁶There is a small ambiguity here when T contains two edges of f_0 , in which case the unique child of f_0 in $\bar{T}$ can be treated as the left or right child of f_0 .

r , of edges in the path.

Trees, Depth, Ancestors, and Descendants. Let T be a tree rooted at a vertex $v_0 \in V(T)$. For each $w \in V(T)$, $P_T(w)$ denotes the path in T from w to v_0 . For each $w_0 \in V(T)$, any prefix $w_0, \dots, w_r$ of $P_T(w_0)$ is called an *upward path* in T ; w_0 is the *lower endpoint* of this path and w_r is the *upper endpoint*. The T -depth of a node $w \in V(T)$ is the length of the path $P_T(w)$. The second node in $P_T(v)$ (if any) is the T -parent of v . A vertex $a \in V(T)$ is a T -ancestor of $w \in V(T)$ if $a \in V(P_T(w))$. If a is a T -ancestor of w then w is a T -descendant of a .

Lowest Common Ancestors. For any two vertices $v, w \in V(T)$, the *lowest common ancestor* $\text{lca}_T(v, w)$ of v and w is the node a in $P_T(v) \cap P_T(w)$ having maximum T -depth. The *lowest common ancestor problem* is a well-studied data structuring problem that asks to preprocess a given n -vertex rooted tree so that one can quickly return $\text{lca}_T(v, w)$ for any two nodes $v, w \in V(T)$. A number of optimal solutions to this problem exist that, after $O(n)$ time preprocessing using $O(n)$ space, can answer queries in $O(1)$ time [1, 4, 5, 16, 20, 25]. The most recent work in this area includes simple and practical data structures that achieve this optimal performance [1, 4, 16].

Reconstructing Binary Tree Models. Let T be a binary tree and $S \subseteq V(T)$. An *upward path* $v_0, \dots, v_r$ in a binary tree T is S -non-branching if v_i has degree 2 and $v_i \notin S$ for each $i \in \{1, \dots, r-1\}$. For any binary tree T and set $S \subseteq V(T)$, the *model* T' of T with respect to S is the binary tree obtained by replacing each maximal S -non-branching path $v_0, \dots, v_r$ with the edge $v_0 v_r$; if v_{r-1} is the left (respectively, right) child of v_r then v_0 becomes the left (respectively, right) child of v_r . See Figure 1.

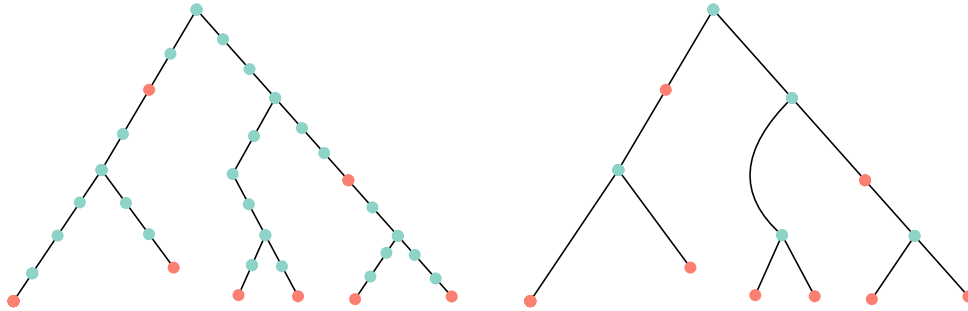


Figure 1: A binary tree T with set $S \subseteq V(T)$ depicted in red and the model of T with respect to S .

The following lemma shows that, given a constant time lowest common ancestor data structure for a rooted tree T and a set S of d vertices in T , we can reconstruct the model T'_0 of a minimal subtree T_0 of T that spans S in time that depends only on d , and not on the size of T or T_0 . In our algorithms we will use values of d ranging from 1 to 5.

Lemma 2. Let T be a binary tree, let $S = \{x_1, \dots, x_d\} \subseteq V(T)$, and let T_0 be the minimal subtree of T that spans S . Then there exists an algorithm that, given an $O(1)$ -query time lowest common ancestor data structure for T , computes the model T'_0 of T_0 with respect to S in $O(d^2)$ time.

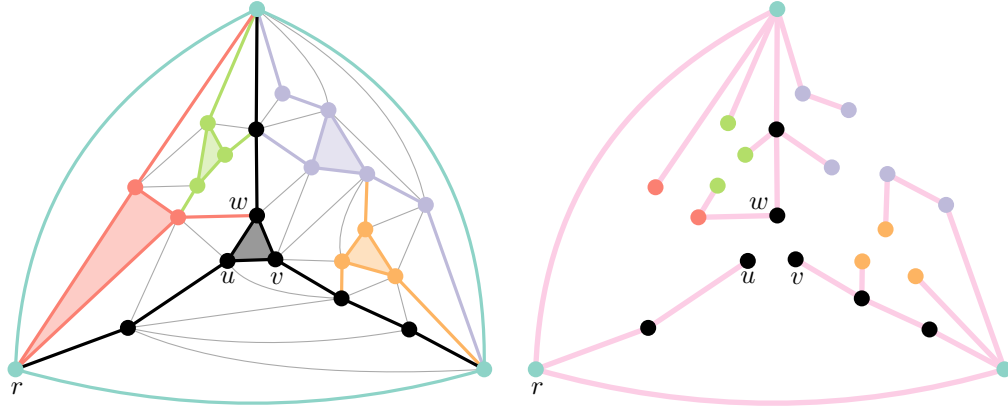


Figure 2: A (G, T) -tripod partition of a triangulation G (and the underlying spanning tree T .)

Proof. The proof is by induction on $|S|$. The base case $|S| = 1$ is trivial, since then $T'_0 = T_0$ is the tree with one node, which is the unique element in S .

If $|S| \geq 2$, then the first step is to determine the root r of T_0 , which must also be the root of T'_0 . This is easily done by first setting $r := x_1$ and then repeatedly setting $r := \text{lca}_T(r, x_i)$ for each $i \in \{2, \dots, d\}$. This step takes $O(d)$ time.

If r has no left child in T , then we can immediately apply induction on $S \setminus \{r\}$ and make the right child of r in T'_0 the root of the model obtained by induction. The case in which r has no right child can be handled similarly. If r has both a left child r_1 and a right child r_2 , then the next step is to partition $S \setminus \{r\}$ into a set S_1 of descendants of r_1 and a set S_2 of descendants of r_2 . For each $x \in S \setminus \{r\}$ there are only two possibilities for $\text{lca}_T(r_1, x)$:

1. If $\text{lca}_T(r_1, x) = r_1$ then $x \in S_1$.
2. If $\text{lca}_T(r_1, x) = r$ then $x \in S_2$.

Therefore, using $O(d)$ lowest common ancestor queries, we can determine the root r of T' and partition $S \setminus \{r\}$ into sets S_1 and S_2 that define the left and right subtrees of r . We can now recurse on S_1 to obtain a tree with root r'_1 and recurse on S_2 to obtain a tree with root r'_2 . We make r'_1 the left child of r and r'_2 the right child of r to obtain the model T'_0 of T_0 . The running time of this algorithm obeys the recurrence $T(d) \leq O(d) + T(d_1) + T(d_2)$, where $d_1 + d_2 \leq d$ and $d_1, d_2 \leq d - 1$. This recurrence resolves to $T(d) \in O(d^2)$. $\square$

3 Tripod Partitions

Refer to Figure 2. Let G be an n -vertex triangulation and let T be a spanning tree of G rooted at a vertex $r \in V(G)$. For a face uvw of G , a (G, T) -tripod Y with crotch uvw is the vertex set of three disjoint (and each possibly empty) upward paths in T (the legs of Y) whose lower endpoints are u , v , and w . A (G, T) -tripod partition is a partition of $V(G)$ into (G, T) -tripods. Dujmović et al. [10] proved the following result:

Theorem 3. *Let G be a triangulation and T be a spanning tree of G . Then there exists a (G, T) -tripod partition $\mathcal{Y}$ such that $G/\mathcal{Y}$ is planar and has treewidth at most 3.*

A *breadth-first spanning (BFS) tree* of a connected graph G is a subgraph of G that is a rooted tree and is rooted at a vertex r of G with the property that the path from r to v in T is a shortest path from r to v in G , for each vertex $v \in V(G)$. It is straightforward to verify that Theorem 3 implies Theorem 1.a by first triangulating the given graph and then taking T to be a BFS tree of the resulting triangulated graph [10, Observation 35]. Briefly, the graph H is defined as $H := G/\mathcal{P}$; the length r of the path $P := p_0, \dots, p_r$ is the height of the BFS tree T ; and each tripod Y in $\mathcal{P}$ maps to a subgraph of $u_Y \boxtimes P \boxtimes K_3$, where u_Y denotes the vertex of H corresponding to Y . More specifically, a vertex $v \in Y$ that has depth d in T , maps to the vertex (u_Y, p_d, i_v) where $i_v \in \{1, 2, 3\}$ indicates the leg of Y that contains v .

3.1 Tripod Partitions from Face Orderings

We now describe how a (G, T) -tripod partition can be obtained from a sequence of distinct faces of G . Throughout this section (and for the remainder of the paper):

- G is an embedded triangulation with outer face f_0 and
- T is a spanning-tree of G rooted at a vertex $v_0 \in V(f_0)$.

For any subgraph f of G , we define $Y_T(f) := f \cup \bigcup_{v \in V(f)} P_T(v)$.⁷ In words, $Y_T(f)$ is the subgraph of G that includes all the vertices and edges of f and all the vertices and edges of each path from each vertex of f to the root of T .

Let $\mathcal{F} := f_0, \dots, f_r$ be a sequence of distinct faces of G whose first element is the outer face f_0 . Let G_{-1} denote the graph with no vertices and, for each $i \in \{0, \dots, r\}$, define the graph $G_i := \bigcup_{j=0}^i Y_T(f_j)$ and let $Y_i := V(G_i) \setminus V(G_{i-1})$.

Informally, we require that each of the legs of each tripod Y_i have a *foot*⁸ on a different vertex of G_{i-1} and that the tripods $Y_1, \dots, Y_r$ cover all the vertices and edges of G . Formally, we say that the sequence $\mathcal{F}$ is *proper* if, for each $i \in \{1, \dots, r\}$, and each distinct $v, w \in V(f_i)$, $V(P_T(v) \cap G_{i-1}) \neq V(P_T(w) \cap G_{i-1})$. The sequence $\mathcal{F}$ is *complete* for G if $G_r = G$. Note that, if $\mathcal{F}$ is complete, then $\{Y_0, \dots, Y_r\}$ is a tripod partition of G .

From the preceding definitions it follows that, if $\mathcal{F}$ is proper, then G_i is 2-connected (since each face of G_i is a cycle) for each $i \in \{0, \dots, r\}$. For any $i \in \{0, \dots, r\}$, consider any face f of G_i that we now treat as a cycle in G . An easy proof by induction shows that, for any $j \in \{0, \dots, i\}$, the induced graph $f[Y_j]$ is connected. We are interested in keeping the number of tripods in $Y_0, \dots, Y_i$ that contribute to $V(f)$ as small as possible, which motivates our next definition.

The sequence $\mathcal{F}$ is *good* if the resulting sequence of graphs $\mathcal{G}_{\mathcal{F}} := G_0, \dots, G_r$ and tripods $\mathcal{Y}_{\mathcal{F}} := Y_0, \dots, Y_r$ satisfy the following condition: For each $i \in \{0, \dots, r\}$ and each face f of G_i ,

$$|\{\ell \in \{0, \dots, i\} : V(f) \cap Y_\ell \neq \emptyset\}| \leq 3.$$

⁷In all of our examples, the subgraph f will always be a single edge or single face of G .

⁸Slightly more formally, the *foot* of a leg $u_0, \dots, u_\ell$ of a (G, T) -tripod is the parent of u_ℓ in T . An empty leg of a tripod has no foot, so a tripod may have zero, one, two, or three feet.

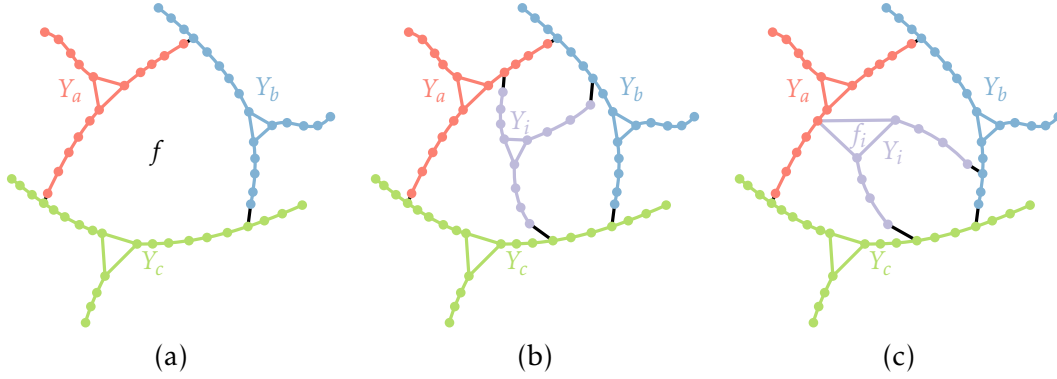


Figure 3: Each face f in G_{i-1} is bounded by at most three tripods Y_{a_f} , Y_{b_f} , and Y_{c_f} and the tripod Y_i is chosen so that it connects each of these.

In words, each face f of each graph G_i has vertices from at most three tripods of $Y_0, \dots, Y_i$ on its boundary. Even more, the vertices of f can be partitioned into at most three paths where the vertices of each path belong to a single tripod. Dujmović et al. [10] prove [Theorem 3](#) by proving the next lemma.

Lemma 4. *Let G be a triangulation with a vertex v_0 on its outer face f_0 and let T be a spanning tree of G rooted at v_0 . Then there exists a sequence $\mathcal{F} := f_0, \dots, f_r$ of distinct faces of G that is proper, good, and complete.*

Remark 5. [Lemma 4](#) is stated in terms of sequences only for convenience and could be rephrased in terms of partial orders. Indeed, consider the partial order $<$ defined as follows: For each $i \in \{1, \dots, r\}$ let f'_i be the face of G_{i-1} that contains f_i ; then $f_\ell < f_i$ for each $\ell \in \{0, \dots, i-1\}$ such that $V(f'_i) \cap Y_\ell \neq \emptyset$. It is straightforward to check that any linearization of this partial order will result in the same tripod partition $\mathcal{Y}_{\mathcal{F}} := \{Y_0, \dots, Y_r\}$.

The key tool used by Dujmović et al. [10] (and the earlier work by Pilipczuk and Siebertz [24]) is Sperner’s Lemma for 3-coloured near-triangulations. Dujmović et al. [10] prove [Lemma 4](#) by giving a recursive algorithm that constructs the face sequence $\mathcal{F}$. For a face f of G_i , define the set $I_f := \{\ell \in \{0, \dots, i\} : V(f) \cap Y_\ell \neq \emptyset\}$. They begin with the outer face f_0 of G . To find the face f_i , $i > 0$, they consider some face $f \notin \{f_0, \dots, f_{i-1}\}$ of G_{i-1} and use Sperner’s Lemma to show that there is an appropriate face f_i of G (called a *Sperner triangle*) that is contained in f . In particular, f_i is chosen so that the three upward paths in $Y_T(f_i)$ lead back to each of the (at most 3) tripods in $\{Y_j : j \in I_f\}$. See [Figure 3](#).

This proof leads to a divide-and-conquer algorithm: After finding f_i , the algorithm recursively decomposes each of the near-triangulations that are bounded by the at most three new faces in $S_i := F(G_i) \setminus F(G_{i-1}) \setminus \{f_i\}$. The Sperner triangle f_i can easily be found in time proportional to the number of faces of G in the interior of f . However, because the resulting recursion is not necessarily balanced, a straightforward implementation of this yields an algorithm with $\Theta(n^2)$ worst-case running time.

Morin [22] later showed that, using an appropriate data structure for T , this approach can be implemented in such a way that the resulting algorithm runs in $O(n \log n)$ time.

Essentially, Morin's algorithm works by finding the Sperner triangle f_i in time proportional to the minimum number of faces of G contained in any of the faces in S_i . In the next section, we will show that, by using a lowest common ancestor data structure for the cotree $\overline{T}$ along with [Lemma 2](#), the Sperner triangle f_i can be found in constant time, yielding an $O(n)$ time algorithm.

By now, our presentation of this material differs somewhat from that in [10, 26]. Therefore, we now pause to explain how [Lemma 4](#) implies [Theorem 3](#). A graph H is *chordal* if there exists an ordering $v_1, \dots, v_h$ of the vertices of H such that, for each $i \in \{2, \dots, h\}$, $N_G(v_i) \cap \{v_1, \dots, v_{i-1}\}$ is a clique in the induced graph $G[\{v_1, \dots, v_{i-1}\}]$. Let G be a triangulation, let T be a spanning tree of G , let $\mathcal{F} := f_0, \dots, f_r$ be the proper good face sequence guaranteed by [Lemma 4](#), and let $\mathcal{Y}_{\mathcal{F}} := \{Y_0, \dots, Y_r\}$ be the resulting tripod partition. We now show that there exists a chordal graph H whose largest clique has size at most 4 and that contains $G/\mathcal{Y}_{\mathcal{F}}$. We construct the graph H so that for each $i \in \{0, \dots, r\}$ and each face f of G_i , H contains a clique on $\{Y_j : j \in I_f\}$. To accomplish this, for each $i \in \{1, \dots, r\}$ let f be the face of G_{i-1} that contains f_i and form a clique on $\{Y_i\} \cup \{Y_j : j \in I_f\}$. Inductively, the elements of $\{Y_j : j \in I_f\}$ already form a clique, so this operation is equivalent to attaching Y_i to all the vertices of an existing clique of size at most 3. Therefore, this results in a chordal graph H whose largest clique has size at most 4 and therefore H has treewidth at most 3 [17].

4 An $O(n)$ -Time Algorithm

Throughout the rest of this paper, we will assign colours to vertices of graphs. We say that an edge is *monochromatic* if both its endpoints have the same colour and *bichromatic* if its endpoints have different colours. A face is *monochromatic*, *bichromatic*, or *trichromatic* if it contains vertices of one, two, or three distinct colours, respectively. (The definition of good sequences will prevent the appearance of any faces with vertices of more than three different colours.)

Refer to [Figure 4](#) for an illustration of the following (probably well-known) lemma, which is closely related to Sperner's Lemma:

Lemma 6. *Let N be a near-triangulation with outer face $v_0, \dots, v_r$ and colour each vertex of N red or blue in such a way that $v_0, \dots, v_\ell$ are coloured red for some $\ell \in \{0, \dots, r-1\}$ and $v_{\ell+1}, \dots, v_r$ are coloured blue. Then there exists a path $w_0, \dots, w_k$ in the dual $N^\star$ of N such that*

1. w_0 is the inner face of N with $v_0 v_r$ on its boundary;
2. w_k is the inner face of N with $v_\ell v_{\ell+1}$ on its boundary; and
3. for each $i \in \{1, \dots, k\}$, the single edge in $E(w_{i-1}) \cap E(w_i)$ has an endpoint of each colour.

Proof. If $w_0 = w_k$, the lemma is immediately true, so assume $w_0 \neq w_k$. The outer face f_0 of N has exactly two bichromatic edges $v_0 v_r$ and $v_\ell v_{\ell+1}$ and any inner face of N has either zero or two bichromatic edges. Consider the subgraph H of $N^\star$ obtained by removing each edge $fg \in E(N^\star)$ such that the edge in $E(f) \cap E(g)$ is monochromatic. Every vertex in H has degree 0 or 2, so each connected component of H is either an isolated vertex or a cycle. The face f_0 has degree 2 so it is contained in a cycle C of H . The two neighbours of f_0 in H are w_0 and w_k . Therefore C contains a path $w_0, \dots, w_k$ that satisfies the conditions of the

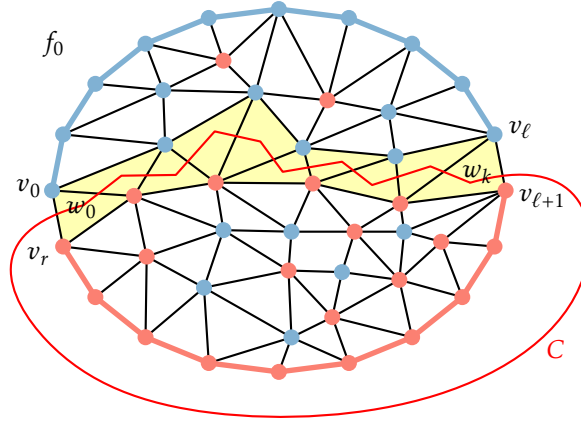


Figure 4: Lemma 6

lemma. □

The next lemma, which is the main new insight in this paper, allows us to use Lemma 2 (which relies on a lowest common ancestor data structure) to find Sperner triangles in constant time. In particular, in the critical case described in this lemma, the Sperner triangle can be found by constructing the model $\bar{T}'_0$ of the minimal subtree $\bar{T}_0$ of $\bar{T}$ that spans the set $S := \{g_1, g_2, g_3\}$. Specifically, the Sperner triangle τ is the unique node of $\bar{T}'_0$ such that each component of $\bar{T}'_0 - \tau$ contains at most one of g_1, g_2 or g_3 .

Lemma 7. *Let G be a triangulation with a vertex v_0 on its outer face f_0 ; let T be a spanning tree of G rooted at v_0 ; let $\bar{T}$ be the cotree of (G, T) rooted at f_0 ; let $f_0, \dots, f_{i-1}$ be a good proper sequence of faces of G that yields a sequence $\mathcal{G}_{\mathcal{F}} := G_0, \dots, G_{i-1}$ of graphs and a sequence $\mathcal{Y}_{\mathcal{F}} := Y_0, \dots, Y_{i-1}$ of tripods; let $f \notin \{f_0, \dots, f_{i-1}\}$ be a face of G_{i-1} , and let S be a set of three distinct faces of G such that, for each $g \in S$,*

- (i) *g is contained in the interior of f ;*
- (ii) *g contains an edge $vw \in E(f)$ with $v \in Y_a$ and $w \in Y_b$ for some distinct $a, b \in I_f = \{\ell \in \{0, \dots, i\} : V(f) \cap Y_\ell \neq \emptyset\}$.*

Let $\bar{T}_0$ be the minimal subtree of $\bar{T}$ that spans S . If $f_i \in V(\bar{T}_0)$ is such that each component of $\bar{T}_0 - f_i$ contains at most one element of S , then $f_0, \dots, f_i$ is good.

Proof. Let N be the near-triangulation consisting of all vertices and edges of G contained in the closure of the interior of f . Since $f_0, \dots, f_{i-1}$ is good, $|I_f| \leq 3$. Since $|S| = 3$, $|I_f| = 3$. For each $j \in I_f$, colour each vertex v of N with the colour j if the first vertex of $P_T(v)$ in $V(f)$ is contained in Y_j .

$E(f)$ contains exactly three bichromatic edges, each of which is on the boundary of exactly one face in S . Let X be the subgraph of N^* that contains an edge $fg \in E(N^*)$ if and only if f and g are inner faces of N and the edge in $E(f) \cap E(g)$ is bichromatic. We claim that X is a subgraph of $\bar{T}$. In order to show this, we need only argue that each edge

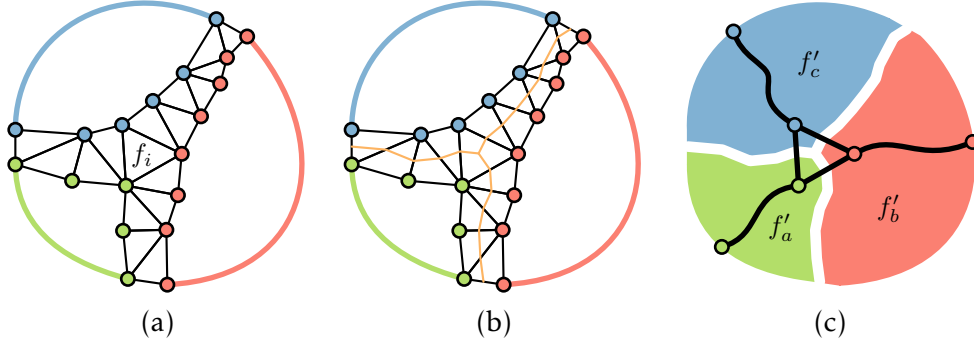


Figure 5: The proof of Lemma 7

uv of T in the interior of f is monochromatic. Consider any $uv \in E(N) \setminus E(f)$ where u is the T -parent of v . If $v \notin V(f)$ then, by definition, v has the same colour as u , so uv is monochromatic. The case where $v \in V(f)$ and $u \notin V(f)$ cannot occur since $v \in V(f)$ implies that $P_T(v) \subseteq G_{i-1}$, but $u \notin V(G_{i-1})$. Similarly, the case in which $u \in V(f)$ and $v \in V(f)$ cannot occur since this implies that $P_T(v) \subseteq G_{i-1}$, but $uv \notin E(G_{i-1})$.

Next we claim that all the elements of S are in a single connected component of X . To see this, let $\{a, b, c\} := I_f$ and consider a pair $g_1, g_2 \in S$ where (without loss of generality) g_1 contains a bichromatic edge of f with colours a and b , and g_2 contains a bichromatic edge of f with colours b and c . By treating a and c as a single colour, we apply Lemma 6 to conclude that g_1 and g_2 are in the same component of X .

Refer to Figure 5(a). Therefore X is a subgraph of $\bar{T}$ that has a component containing all the elements of S . Therefore X contains $\bar{T}_0$. By choice, $\bar{T}_0$ contains a path from f_i to each $g \in S$ and these paths are pairwise disjoint except for their shared starting point f_i .

Refer to Figure 5(b). Now, consider the embedded graph X_0 obtained as follows: For each $g \in V(\bar{T}_0)$, place a vertex on the center of each bichromatic edge of g and, if g is trichromatic, then place a vertex in the center of g . Next,

1. add an edge joining the center of each trichromatic triangle to each of the centers of its bichromatic edges; and
2. add an edge (embedded as a straight line segment) joining the centers of each pair of bichromatic edges that are on a common bichromatic face $g \in V(\bar{T}_0)$.

The graph X_0 is a tree of maximum-degree 3 that has three leaves. (Each leaf in X_0 is the center of a bichromatic edge of $E(f)$). Therefore X_0 has exactly one degree-3 vertex. We claim that this degree-3 vertex is in the interior of f_i . Indeed, if this were not the case, then $\bar{T}_0 - f_i$ would necessarily have a component Z such that $\bigcup_{g \in V(Z)} E(g)$ contains at least two bichromatic edges of $E(f)$. But this would imply that Z contains two elements of S , contradicting the definition of f_i .

Refer to Figure 5(c). Now treat X_0 as a point set and consider the point set f' obtained by removing X_0 from the closure of f . Now f' has $|I_f| = 3$ connected components and each vertex of f_i is in a different component. Each of the components of f' contains vertices of Y_j for exactly one $j \in I_f$; call this component f'_j . Since no edge of T crosses X_0 , the colour

of each vertex v in f_j' is equal to j , for each $j \in I_f$. Thus, each vertex of f_i has a distinct colour.

To see that $f_0, \dots, f_i$ is good, first observe that we need only be concerned with the at most three faces in $F(G_i) \setminus F(G_{i-1}) \setminus \{f_i\}$. Let $g \in F(G_i) \setminus F(G_{i-1}) \setminus \{f_i\}$. Then g has exactly two vertices v and w of f_i on its boundary. Without loss of generality v and w have colours a and b with $a \neq b$. Then all vertices of g are contained in $f_a' \cup f_b'$. Therefore, all vertices of $V(g)$ have colour a or b . Since $V(g) \subseteq V(f) \cup Y_i \subseteq Y_a \cup Y_b \cup Y_c \cup Y_i$, we have $V(g) \subseteq Y_a \cup Y_b \cup Y_i$. Therefore $|I_g| = |\{\ell \in \{0, \dots, i\} : V(g) \cap Y_\ell \neq \emptyset\}| \leq |\{a, b, i\}| = 3$. Since this is true for each $g \in F(G_i) \setminus F(G_{i-1}) \setminus \{f_i\}$, $f_0, \dots, f_i$ is good. $\square$

We now pause to explain the representation of our triangulation. For concreteness, we describe our algorithms using a half-edge data structure, variants of which are described in [2, 19, 21, 27]. The important property of this structure is that the following operations are supported in constant time: For every directed half-edge vw , we can find the reversal wv of vw , the next directed half-edge vw' that appears in counterclockwise order around v , and the face to the left of vw . For every face f , we can find the three neighbouring faces of f , the three incident vertices of f , and the three half-edges having f to their left.

Theorem 8. *There exists an $O(n)$ time algorithm that, given any n -vertex triangulation G and any rooted spanning tree T of G , produces a (G, T) -tripod partition $\mathcal{Y}$ such that $\text{tw}(G/\mathcal{Y}) \leq 3$.*

Proof. Let v_0 be the root of T and let f_0 be a face of G incident to v_0 that contains an edge of T incident to v_0 . In a preprocessing step, we compute the cotree $\bar{T}$ of (G, T) and construct a lowest common ancestor data structure for $\bar{T}$ in $O(n)$ time that allows us to compute $\text{lca}_{\bar{T}}(f, g)$ for any two faces $f, g \in F(G)$ in $O(1)$ time.

After this preprocessing, we construct the good sequence $f_0, \dots, f_r$ recursively. Conceptually, during any recursive invocation, the input is a near-triangulation N bounded by a cycle C in G whose vertices belong to at most three tripods computed in previous steps. Each vertex of G starts initially *uncoloured* and we *colour* a vertex v in a tripod Y_i with the colour i when tripod Y_i is created. Likewise, each face of G starts initially uncoloured and we colour a face once we have added it to the good sequence. Initially, only the three vertices of f_0 are coloured and each is assigned the colour 0 of the tripod Y_0 . There are three types of recursive instances. The initial recursive call is a monochromatic instance, described below.

In the following, we first describe the precise input to each type of recursive instance and the resulting Sperner triangle f_i arising from this type of instance. Then, for each type of instance, we describe how to construct the recursive subproblems (corresponding to faces of G_i that are not present in G_{i-1}) that are generated by the instance. The precise input to a recursive invocation is defined as follows:

Trichromatic instances. If C intersects three tripods, then the input consists of the three bichromatic half-edges e_1 , e_2 , and e_3 of C (arranged in counterclockwise order around C) that have N to their left. For each $i \in \{1, 2, 3\}$, let g_i be the inner face of N to the left of e_i .

If the faces g_1 , g_2 , and g_3 are all distinct, then [Lemma 7](#) characterizes the face f_i in

terms of the minimum subtree $\bar{T}_0$ of $\bar{T}$ that contains g_1 , g_2 , and g_3 . Indeed, f_i is either the unique degree-3 node of $\bar{T}_0$ (if g_1 , g_2 , and g_3 are all leaves of $\bar{T}_0$) or f_i is the unique node among g_1 , g_2 , and g_3 that has degree 2. By [Lemma 2](#) we can construct the model $\bar{T}'_0$ of $\bar{T}_0$ in constant time and find the node f_i . If the faces g_1 , g_2 and g_3 are not all distinct because, for instance $g_1 = g_2$, then the face g_1 has at least two edges (e_1 and e_2) that belong to C , and the three vertices of g_1 belong to three different tripods. In this case we choose $f_i = g_1$.

Bichromatic instances. If C intersects two tripods, then the input consists of the two half-edges e_1 and e_2 of C . By extension, g_1 and g_2 are the two inner faces of N to the left of these half-edges. In this case, we let $f_i := g_1$ or $f_i = g_2$, either choice satisfies our requirements.

Monochromatic instances. If C intersects only one tripod, then the input consists of some half-edge e_1 of C . By extension, g_1 is the inner face of N to the left of this half-edge. In this case $f_i := g_1$ satisfies our requirements.

Constructing tripods and identifying subproblems. Once we have found f_i , we can compute the tripod Y_i and mark its vertices by following the path in T from each vertex of f_i to its nearest marked ancestor in T . This takes $O(1 + |Y_i|)$ time. Once we have done this, we have also found the at most three bichromatic edges of G_i that are needed to perform the at most three recursive invocations on the near-triangulations whose outer faces coincide with each of the new faces in $F(G_i) \setminus F(G_{i-1}) \setminus \{f_i\}$. To identify these recursive invocations in constant time, we need to efficiently:

1. recognize some candidate near-triangulation as being non-empty, and
2. determine the bichromatic half-edges needed to perform said recursive invocation.

We begin with the first issue. Let uv be a half-edge of f_i with f_i to the left of uv . There are three situations that may cause the subproblem to the right of uv to be empty:

1. uv is an edge of C . To detect this we distinguish between two cases. If uv is bichromatic then uv belongs to C if and only if $uv \in \{e_1, e_2, e_3\}$. If uv is monochromatic then uv belongs to C if and only if u is the T -parent of v (or vice-versa) or uv is an edge of f_j for some $j < i$. The former situation can easily be checked directly. The latter situation can be identified by checking the colour (if any) of the face to the right of the half-edge uv .
2. $u \in Y_i$ and v is the T -parent of u .
3. $v \in Y_i$ and u is the T -parent of v .

Note that points 1 and 2 are mutually exclusive, since if uv is an edge of C then $u \notin Y_i$. Similarly, points 1 and 3 are also mutually exclusive. The precise subproblem identification process is broken down as follows:

Recursive subproblems from monochromatic instances. Monochromatic instances induce at most two recursive invocations. We must use points 1, 2, and 3 above to identify subproblems. Let uvw be the vertices of f_i , with $vw = e_1$. Then, there exists a subproblem to the right of uv , and a subproblem to the right of wu . We describe how to treat the subproblem to the right of uv using points 1 and 2. There are two cases to consider:

-
1. If u is a vertex of C , then this subproblem is empty if and only if uv is an edge of C (point 1). If this subproblem is non-empty, then it is a monochromatic subproblem whose input is the half-edge $e'_1 = vu$.
 2. If u is not a vertex of C , then $u \in Y_i$ and the subproblem to the right of uv is empty if and only if v is the T -parent of u (point 2). If this subproblem is non-empty, then it is a bichromatic subproblem whose input is the half-edge $e'_1 = vu$ and the half-edge $e'_2 = u'u''$, where u' is the last vertex in the leg of Y_i that contains u and u'' is the T -parent of u' .

The subproblem to the right of wu is treated similarly using points 1 and 3.

Recursive subproblems from bichromatic instances. Similar to monochromatic instances, bichromatic instances induce at most two recursive invocations. We must use points 1, 2, and 3 above to identify subproblems. Let uvw be the vertices of f_i , with $vw = e_1$. Then, there exists a subproblem to the right of uv , and a subproblem to the right of wu . We describe how to treat the subproblem to the right of uv using points 1 and 2. There are two cases to consider:

1. If u is a vertex of C , then this subproblem is empty if and only if uv is an edge of C (point 1). If this subproblem is non-empty, then it is either a monochromatic subproblem or a bichromatic subproblem.
 - (a) A non-empty subproblem is monochromatic if u and v are coloured identically (that is, uv is monochromatic). The input to a monochromatic subproblem is the half-edge $e'_1 = vu$.
 - (b) A non-empty subproblem is bichromatic if u and w are coloured identically (that is, uw is bichromatic). The input to a bichromatic subproblem is the half-edge $e'_1 = vu$ and the half-edge $e'_2 = e_2$.
2. If u is not a vertex of C , then $u \in Y_i$. Then the subproblem to the right of uv is empty if and only if v is the T -parent of u (point 2). Let u' denote the last vertex in the leg of Y_i that contains u and let u'' be the T -parent of u' . If the subproblem to the right of uv is non-empty, then it is either a bichromatic subproblem or a trichromatic subproblem.
 - (a) A non-empty subproblem is bichromatic if v and u'' are coloured identically. The input to a bichromatic subproblem is the half-edge $e'_1 = vu$ and the half-edge $e'_2 = u'u''$.
 - (b) A non-empty subproblem is trichromatic if w and u'' are coloured identically. The input to a trichromatic subproblem is the half-edge $e'_1 = vu$, the half-edge $e'_2 = u'u''$, and the half-edge $e'_3 = e_2$.

The subproblem to the right of wu is treated similarly using points 1 and 3.

Recursive subproblems from trichromatic instances. Trichromatic instances induce at most three recursive invocations. We must use point 1 above to identify subproblems. Let uvw be the vertices of f_i . Then, there exists a subproblem to the right of uv , a subproblem to the right of vw , and a subproblem to the right of wu . Without loss of generality, suppose that the subproblem to the right of uv contains e_1 , the subproblem to the right of vw contains e_2 , and the subproblem to the right of wu contains e_3 .

We describe how to treat the subproblem to the right of uv . There are four cases to consider:

1. If u is a vertex of C and v is a vertex of C , then this subproblem is empty if and only if uv is an edge of C (point 1). If this subproblem is non-empty, then it is a bichromatic subproblem whose input is the half-edge $e'_1 = vu$ and the half-edge $e'_2 = e_1$.
2. If u is a vertex of C and v is not a vertex of C , then this subproblem is always a non-empty trichromatic subproblem. Let v' denote the last vertex in the leg of Y_i that contains v and let v'' be the T -parent of v' . Then the input to this subproblem is the half-edge $e'_1 = v''v'$, the half-edge $e'_2 = vu$, and the half-edge $e'_3 = e_1$.
3. If u is not a vertex of C and v is a vertex of C , then this subproblem is always a non-empty trichromatic subproblem. Let u' denote the last vertex in the leg of Y_i that contains u and let u'' be the T -parent of u' . Then the input to this subproblem is the half-edge $e'_1 = u'u''$, the half-edge $e'_2 = e_1$, and the half-edge $e'_3 = vu$.
4. If u is not a vertex of C and v is not a vertex of C , then this subproblem is always a non-empty trichromatic subproblem. Let u' denote the last vertex in the leg of Y_i that contains u , let u'' be the T -parent of u' , let v' denote the last vertex in the leg of Y_i that contains v , and let v'' be the T -parent of v' . Then the input to this subproblem is the half-edge $e'_1 = v''v'$, the half-edge $e'_2 = u'u''$, and the half-edge $e'_3 = e_1$.

The subproblem to the right of vw and the subproblem to the right of wu are treated similarly using point 1.

As stated above, the initial recursive call is a monochromatic instance. Its input is any of the three half-edges adjacent to the outer face f_0 that have $G \setminus f_0$ to their left. Each recursive invocation adds a new face f_i to the good face sequence $f_0, \dots, f_r$ and takes $O(1 + |Y_i|)$ time. Since $Y_0, \dots, Y_r$ is a partition of $V(G)$, the running time of this algorithm is therefore $\sum_{i=0}^r O(1 + |Y_i|) = O(n)$. $\square$

5 Variations

In this section we show that there are $O(n)$ time algorithms for computing the partitions in [Theorem 1.b](#) and [Theorem 1.c](#). In the same way that [Theorem 1.a](#) follows from the tripod partition of [Theorem 3](#), [Theorem 1.b](#) follows from a bipod partition given by [Theorem 10](#), and [Theorem 1.c](#) follows from a monopod partition given by [Theorem 11](#).

5.1 Bipod Partitions

We begin with the partition in [Theorem 1.b](#), which was communicated to us by Vida Dujmović, and has not appeared before. This partition is obtained by selecting a proper sequence $\mathcal{E} := e_0, \dots, e_r$ of distinct edges of G , which define a sequence of graphs $\mathcal{G}_{\mathcal{E}} := G_0, \dots, G_r$ where $G_i := \bigcup_{j=0}^i Y_T(e_j)$ and a sequence of *bipods* $\mathcal{I}_{\mathcal{E}} := \Lambda_0, \dots, \Lambda_r$ where $\Lambda_i = V(G_i) \setminus V(G_{i-1})$. We call $\mathcal{E}$ *good* if, for each $i \in \{0, \dots, r\}$ and each face $f \in F(G_i)$, $V(f)$ has a non-empty intersection with at most 4 bipods in $\Lambda_0, \dots, \Lambda_i$.

Exactly the same argument used in [Section 3.1](#) to show that $G/\mathcal{V}_{\mathcal{F}}$ is contained in a chordal graph of maximum clique size 4 also shows that if $\mathcal{E}$ is a good edge sequence that produces a bipod partition $\mathcal{I}_{\mathcal{E}}$ of $V(G)$, then $G/\mathcal{I}_{\mathcal{E}}$ is contained in a chordal graph of

maximum clique size 5, so $G/\mathcal{I}_\mathcal{E}$ has treewidth at most 4.

We now explain why a good edge sequence $e_0, \dots, e_r$ exists.⁹ As before, we set f_0 to be any face of G such that $E(f_0)$ contains an edge of T incident to the root v_0 of T . The edge e_0 is any edge of $E(f_0) \setminus E(T)$. Next we take special care to ensure that G_i is biconnected for $i \geq 1$. In particular, if G_0 contains only two edges of f_0 , then we take e_1 to be the edge of f_0 that does not appear in G_0 . Otherwise, we choose e_1 using the general strategy for choosing e_i , described next.

Refer to Figure 6. Now we may assume that G_{i-1} is biconnected. To choose the edge e_i , we consider any face $f \in F(G_{i-1}) \setminus F(G)$. Inductively, $V(f)$ contains vertices from at most four bipods in $\Lambda_0, \dots, \Lambda_{i-1}$. Let $I_f := \{j \in \{0, \dots, i-1\} : \Lambda_j \cap V(f) \neq \emptyset\}$. If $|I_f| < 4$ then we can select e_i to be any edge in the interior f . Therefore, we focus on the case $|I_f| = 4$. As before we colour vertices in the near-triangulation N using colours in the set I_f ; we let S be the set of inner faces in N that contain a bichromatic edge in $E(f)$; and let $\bar{T}_0$ be the minimal subtree of $\bar{T}$ that spans S . The same argument in the proof of Lemma 7 shows that every node of $\bar{T}_0$ is contained in f .

Claim 9, below, shows that $\bar{T}_0$ contains an edge xy such that each component of $\bar{T}_0 - xy$ contains at most two elements of S . It is straightforward to verify that, if we choose e_i to be the edge in $E(x) \cap E(y)$ then we obtain a graph G_i in which each of the two new faces containing vertices from Λ_i contains vertices from at most three bipods in $\{\Lambda_j : j \in I_f\}$, as required.

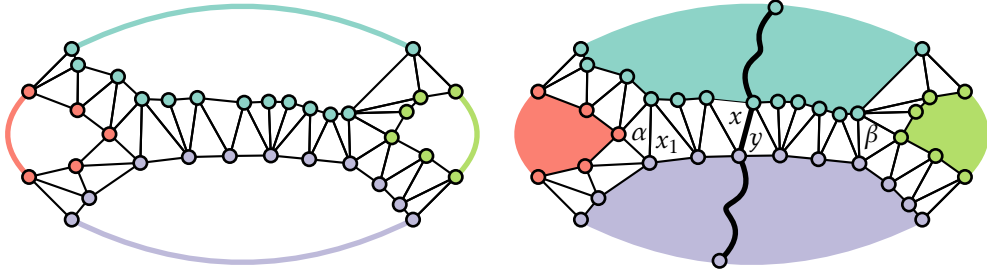


Figure 6: Choosing the next edge e_i in a good edge sequence.

Claim 9. $\bar{T}_0$ contains an edge xy such that each component of $\bar{T}_0 - xy$ contains at most two nodes of S .

Proof. Direct each edge xy of $\bar{T}_0$ in the direction $\overrightarrow{xy}$ if the component of $\bar{T}_0 - xy$ that contains y contains three or more nodes of S . It is sufficient to show that this process leaves some edge xy of $\bar{T}_0$ undirected. Assume for the sake of contradiction that every edge of $\bar{T}_0$ is directed. Then some node x of $\bar{T}_0$ has only incoming edges. Certainly x does not have degree 1 in $\bar{T}_0$.

⁹The existence of this edge sequence is more easily proven using Sperner's Lemma, but we want a proof that lends itself to a linear time algorithm.

If x has degree 2 in $\bar{T}_0$ then $\bar{T}_0$ contains two subtrees T_1 and T_2 that have only the node x in common and such that $|V(T_1) \cap S| \geq 3$ and $|V(T_2) \cap S| \geq 3$, which implies that $|S| \geq 3 + 3 - 1 > 4$, a contradiction.

Suppose therefore that x has degree 3 in $\bar{T}_0$. Each face in S contains an edge in $E(f)$, so each face in S has degree at most 2 in $\bar{T}_0$. Therefore $x \notin S$. Therefore $\bar{T}_0 - x$ contains three components T_1, T_2, T_3 such that each pair of components contains at least 3 elements of S . But this implies that $|S| \geq (3 \times 3)/2 > 4$, a contradiction. $\square$

Algorithmically, using [Lemma 2](#), we can construct the model $\bar{T}'_0$ of $\bar{T}_0$ in constant time given the set S . The model $\bar{T}'_0$ will also contain an edge $\alpha\beta$ such that each component of $\bar{T}'_0 - \alpha\beta$ contains at most two nodes of S . We claim that $E(\alpha)$ contains an edge that makes a suitable choice for e_i , and this edge can be found in constant time. Indeed, the edge $\alpha\beta$ in $\bar{T}'_0$ corresponds to a path $\alpha, x_1, \dots, x_k, \beta$ in $\bar{T}_0$ and the unique edge in $E(\alpha) \cap E(x_1)$ is a suitable choice for e_i .

The rest of the details of the algorithm are similar to those given in the proof of [Theorem 8](#): Each subproblem is a near-triangulation N bounded by a cycle C and the input that defines the subproblem consists of the (at most four) bichromatic half-edges of C . (In the degenerate case where C has no bichromatic edges, the input is any half-edge of C .) Similar issues arise when determining which subproblems are non-empty and determining the half-edges used as input for each non-empty subproblem. The interested reader can find these details in the code [\[18, 23\]](#).

Theorem 10. *There exists an $O(n)$ time algorithm that, given any n -vertex triangulation G and any rooted spanning tree T of G , produces a (G, T) -bipod partition $\mathcal{I}$ such that $\text{tw}(G/\mathcal{I}) \leq 4$.*

5.2 Monopod Partitions

Finally we consider the partition described in [Theorem 1.c](#). This partition is obtained from a tripod partition $\mathcal{Y} := Y_0, \dots, Y_r$, obtained by a sequence $\mathcal{F} := f_0, \dots, f_r$ of faces of G in the same manner described in [Section 3.1](#). However in this setting, the sequence $f_0, \dots, f_r$ is *good* if, for each $i \in \{0, \dots, r\}$ and each face f of $G_i := \bigcup_{j=0}^i Y_T(f_j)$, $V(f)$ contains vertices from at most 5 legs of tripods in $Y_0, \dots, Y_i$. Under these conditions, Ueckerdt et al. [\[26\]](#) are able to show that the *monopod partition* $\mathcal{I}$ obtained by splitting each tripod Y_i into three upward paths yields a quotient graph $G/\mathcal{I}$ of treewidth at most 6.

As before we focus on the extreme case when $V(f)$ contains vertices from exactly 5 legs of monopods. Refer to [Figure 7](#). Following the same strategy used for the previous two partitions, the set S in this case has size at most 5 and the face f_i corresponds to a node of $\bar{T}_0$ such that each component of $\bar{T}_0 - f_i$ contains at most 2 nodes of S . (This is always possible because $\lfloor 5/2 \rfloor = 2$.) Again, a suitable choice for f_i can be found in the model $\bar{T}'_0$ of $\bar{T}_0$ in constant time.

The details of determining which subproblems are non-empty and which half-edges are used as input for each subproblem are similar to the previous two partitions. Details can be found in the source code [\[18, 23\]](#).

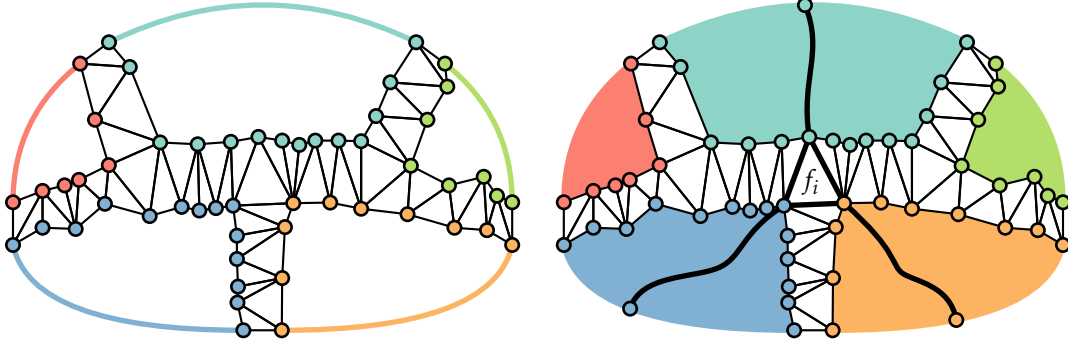


Figure 7: The selection of a tripod by Ueckerdt et al. [26]

Theorem 11. *There exists an $O(n)$ time algorithm that, given any n -vertex triangulation G and any rooted spanning tree T of G , produces a (G, T) -monopod partition $\mathcal{I}$ such that $\text{tw}(G/\mathcal{I}) \leq 6$.*

6 Implementation

The algorithms described in this paper have been implemented in C++ [23] and in C [18]. The C++ implementation includes the algorithms for tripod, bipod, and monopod partitions. The C implementation implements only the tripod decomposition (which, at the time of writing, gives the tightest results in theoretical applications).

As discussed in the proof of [Theorem 8](#), the implementation requires some care when deciding which subproblems are non-empty, but the resulting code is not overly long. The entire C++ implementation of all three algorithms, including the code for the lowest common ancestor data structure, has 924 lines of code.¹⁰ The resulting code is also quite efficient. [Figure 8](#) shows the total running time of the implementation on n -vertex triangulations for n ranging from 10,000 to 1,990,000 in increments of 10,000. Each measurement is the average of 20 runs on 20 different triangulations.

[Figure 8](#) breaks down the total running time of the algorithms into the various stages, including the construction of the BFS tree T , the construction of the cotree $\bar{T}$, and the tripod partition algorithm proper. [Figure 8](#) does not include the time required to read the input file that contains the triangulation G and convert it into the triangle adjacency structure described before the proof of [Section 4](#). The upper part of [Figure 8](#) shows the total running time, in seconds. The lower part shows the total running time divided by the number of vertices. For even the largest triangulations tested, the time per vertex is not more than 1.3×10^{-6} seconds (1.3 microseconds).

The time per vertex appears to increase slowly with n in the lower part of [Figure 8](#). This is a caching effect. The CPU in this machine has three levels of cache, the largest of which has 6MB. As the size of the input triangulation increases, the fraction of memory references that are available in the cache decreases, resulting in (slower) accesses to RAM. In the limit, every memory access would be to RAM and, on this machine this would result

¹⁰A line of code is any line that contains a semicolon.

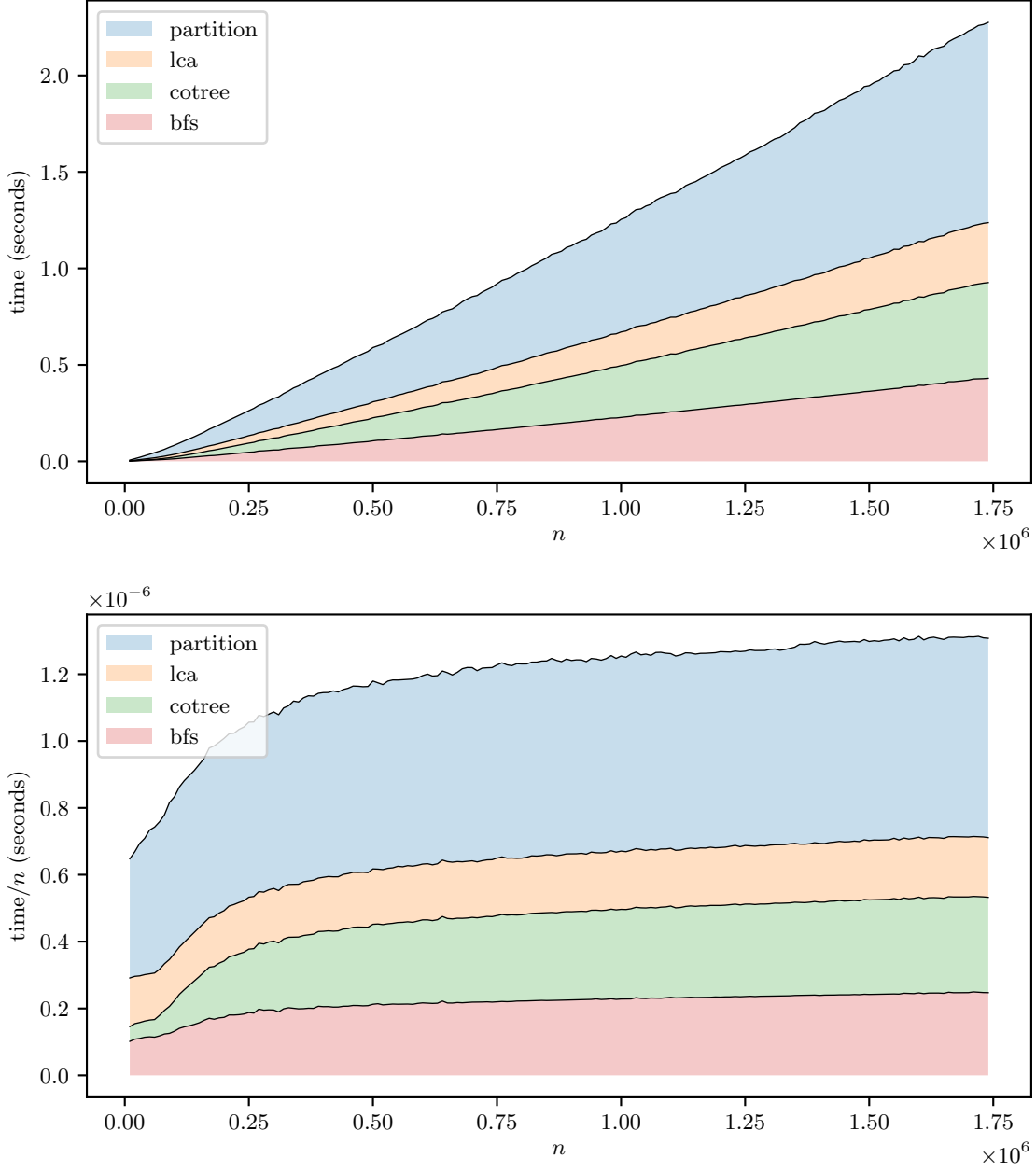


Figure 8: Running time of the tripod partition algorithm C++ implementation in [23]

in a processing time of approximately 1.4 microseconds per vertex.

The input triangulations used to generate [Figure 8](#) were obtained by computing Delaunay triangulations of points uniformly distributed in a circle, with three additional points added to ensure that the outer face is a triangle. The machine used to run the experiments is the third author's office desktop computer. The CPU in this machine is an Intel(R) Core(TM) i5-4670K CPU @ 3.40GHz. The machine has 16GB of DDR3 memory in four 4GB modules configured at a speed of 1600MT/s. (This machine is nothing special. It was purchased in 2013 and is a standard desktop computer.)

The short summary of these results is that the implementation is fast in real terms on basic hardware. For comparison, the time required to make a 1-million vertex triangulation using the (very fast) delaunator algorithm [6] is approximately 8 seconds. The time required to compute a tripod partition of this triangulation is approximately 1 second.

Acknowledgement

This research was initiated at the BIRS 21w5235 Workshop on Graph Product Structure Theory, held November 21–26, 2021 at the Banff International Research Station. The authors are grateful to the workshop organizers and participants for providing a stimulating research environment. We are especially grateful to Vida Dujmović for sharing [Theorem 1.b](#) with us.

References

- [1] Stephen Alstrup, Cyril Gavoille, Haim Kaplan, and Theis Rauhe. Nearest common ancestors: A survey and a new algorithm for a distributed environment. *Theory Comput. Syst.*, 37(3):441–456, 2004. doi:10.1007/s00224-004-1155-5.
- [2] Bruce G. Baumgart. A polyhedron representation for computer vision. In *Proceedings of the May 19-22, 1975, National Computer Conference and Exposition, AFIPS '75*, page 589–596. Association for Computing Machinery, New York, NY, USA, 1975. doi:10.1145/1499949.1500071.
- [3] Michael A. Bekos, Giordano Da Lozzo, Petr Hliněný, and Michael Kaufmann. Graph product structure for h -framed graphs. *The Electronic Journal of Combinatorics*, 31(4):P4.56, 2024. doi:10.37236/12123.
- [4] Michael A. Bender and Martin Farach-Colton. The LCA problem revisited. In Gaston H. Gonnet, Daniel Panario, and Alfredo Viola, editors, *LATIN 2000: Theoretical Informatics, 4th Latin American Symposium, Punta del Este, Uruguay, April 10-14, 2000, Proceedings*, volume 1776 of *Lecture Notes in Computer Science*, pages 88–94. Springer, 2000. doi:10.1007/10719839_9.
- [5] Omer Berkman and Uzi Vishkin. Recursive star-tree parallel data structure. *SIAM J. Comput.*, 22(2):221–242, 1993. doi:10.1137/0222017.
- [6] Vladi Bilonenko. delaunator-cpp: A really fast C++ library for delaunay triangulation of 2d points. <https://github.com/delfrrr/delaunator-cpp/>, 2018. GitHub repository.

-
- [7] Prosenjit Bose, Vida Dujmović, Mehrnoosh Javarsineh, and Pat Morin. Asymptotically optimal vertex ranking of planar graphs. *CoRR*, abs/2007.06455, 2020. [2007.06455](#).
- [8] Prosenjit Bose, Pat Morin, and Saeed Odak. An optimal algorithm for product structure in planar graphs. In Artur Czumaj and Qin Xin, editors, *18th Scandinavian Symposium and Workshops on Algorithm Theory, SWAT 2022, Tórshavn, Faroe Islands, June 27-29, 2022*, volume 227 of *LIPICs*, pages 19:1–19:14. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022. [doi:10.4230/LIPICS.SWAT.2022.19](#).
- [9] Reinhard Diestel. *Graph Theory, Fifth Edition*, volume 173 of *Graduate texts in mathematics*. Springer, 2017. [doi:10.1007/978-3-662-53622-3](#).
- [10] Vida Dujmović, Gwenaël Joret, Piotr Micek, Pat Morin, Torsten Ueckerdt, and David R. Wood. Planar graphs have bounded queue-number. *J. ACM*, 67(4):22:1–22:38, 2020. [doi:10.1145/3385731](#).
- [11] Vida Dujmović, Pat Morin, and David R. Wood. Graph product structure for non-minor-closed classes. *Journal of Combinatorial Theory, Series B*, 162:34–67, 2023. [doi:10.1016/J.JCTB.2023.03.006](#).
- [12] Vida Dujmović, Louis Esperet, Cyril Gavoille, Gwenaël Joret, Piotr Micek, and Pat Morin. Adjacency labelling for planar graphs (and beyond). *J. ACM*, 68(6):42:1–42:33, 2021. [doi:10.1145/3477542](#).
- [13] Vida Dujmović, Louis Esperet, Gwenaël Joret, Bartosz Walczak, and David R. Wood. Planar graphs have bounded nonrepetitive chromatic number. *CoRR*, abs/1904.05269, 2019. [1904.05269](#).
- [14] Michał Dębski, Stefan Felsner, Piotr Micek, and Felix Schröder. Improved bounds for centered colorings. In Shuchi Chawla, editor, *Proceedings of the 2020 ACM-SIAM Symposium on Discrete Algorithms, SODA 2020, Salt Lake City, UT, USA, January 5-8, 2020*, pages 2212–2226. SIAM, 2020. [doi:10.1137/1.9781611975994.136](#).
- [15] Louis Esperet, Gwenaël Joret, and Pat Morin. Sparse universal graphs for planarity. *Journal of the London Mathematical Society*, 108(4):1333–1357, 2023. [doi:10.1112/jlms.12781](#).
- [16] Johannes Fischer and Volker Heun. Theoretical and practical improvements on the rmq-problem, with applications to LCA and LCE. In Moshe Lewenstein and Gabriel Valiente, editors, *Combinatorial Pattern Matching, 17th Annual Symposium, CPM 2006, Barcelona, Spain, July 5-7, 2006, Proceedings*, volume 4009 of *Lecture Notes in Computer Science*, pages 36–48. Springer, 2006. [doi:10.1007/11780441_5](#).
- [17] Fănică Gavril. The intersection graphs of subtrees in trees are exactly the chordal graphs. *Journal of Combinatorial Theory, Series B*, 16:47–56, 1974. [doi:doi:10.1016/0095-8956\(74\)90094-X](#).
- [18] K. Gouin. `planar_prod_struct_theorem`. https://github.com/kgouin/planar_prod_struct_theorem, 2026. GitHub repository.
-

-
- [19] Leonidas Guibas and Jorge Stolfi. Primitives for the manipulation of general subdivisions and the computation of voronoi. *ACM Trans. Graph.*, 4(2):74–123, 1985. doi: [10.1145/282918.282923](https://doi.org/10.1145/282918.282923).
- [20] Dov Harel and Robert Endre Tarjan. Fast algorithms for finding nearest common ancestors. *SIAM J. Comput.*, 13(2):338–355, 1984. doi: [10.1137/0213024](https://doi.org/10.1137/0213024).
- [21] Lutz Kettner. *Software design in computational geometry and contour-edge based polyhedron visualization*. Ph.D. thesis, ETH Zürich, 1999.
- [22] Pat Morin. A fast algorithm for the product structure of planar graphs. *Algorithmica*, 83(5):1544–1558, 2021. doi: [10.1007/s00453-020-00793-5](https://doi.org/10.1007/s00453-020-00793-5).
- [23] Pat Morin. fatripair: Linear-time tripod partitions of triangulations. <https://github.com/patmorin/fatripair>, 2026. GitHub repository.
- [24] Michal Pilipczuk and Sebastian Siebertz. Polynomial bounds for centered colorings on proper minor-closed graph classes. *J. Comb. Theory B*, 151:111–147, 2021. doi: [10.1016/J.JCTB.2021.06.002](https://doi.org/10.1016/J.JCTB.2021.06.002).
- [25] Baruch Schieber and Uzi Vishkin. On finding lowest common ancestors: Simplification and parallelization. *SIAM J. Comput.*, 17(6):1253–1262, 1988. doi: [10.1137/0217079](https://doi.org/10.1137/0217079).
- [26] Torsten Ueckerdt, David R. Wood, and Wendy Yi. An improved planar graph product structure theorem. *The Electronic Journal of Combinatorics*, 29(2):P2.51, 2022. doi: [10.37236/10614](https://doi.org/10.37236/10614).
- [27] Kevin Weiler. Edge-based data structures for solid modeling in curved-surface environments. *IEEE Computer Graphics and Applications*, 5(1):21–40, 1985. doi: [10.1109/MCG.1985.276271](https://doi.org/10.1109/MCG.1985.276271).