Deriving Embeddings and Triconnectivity from the Haeupler-Tarjan Planarity Test

Bachelor Thesis of

Tim-Florian Feulner

At the Department of Informatics and Mathematics
Chair of Theoretical Computer Science

Reviewer:	Prof. Dr. Ignaz Rutter
Advisor:	Simon Fink, M.Sc.

Time Period: 16st May 2023 – 21st August 2023

Eigenständigkeitserklärung

Hiermit versichere ich, Tim-Florian Feulner,

1. dass ich die vorliegende Arbeit selbstständig und ohne unzulässige Hilfe verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt, sowie die wörtlich und sinngemäß übernommenen Passagen aus anderen Werken kenntlich gemacht habe.
2. Außerdem erkläre ich, dass ich der Universität ein einfaches Nutzungsrecht zum Zwecke der Überprüfung mittels einer Plagiatssoftware in anonymisierter Form einräume.

Passau, 21.08.2023

Abstract

A planarity test is an algorithm that determines whether a graph is planar, i.e. whether it can be drawn in the two-dimensional plane such that no two edges cross each other. These algorithms may be extended to embedding algorithms, which compute a planar embedding of the input graph, i.e. the cyclic order of edges at each vertex in a planar drawing of that graph. Multiple linear-time planarity tests have been proposed, for instance the "vertex-addition" test by Lempel, Even, and Cederbaum in 1967 [LEC67], which has been extended for computing an embedding by Chiba et al. in 1985 [CNAO85]. In 2008, Haeupler and Tarjan proposed a conceptually simpler version of the "vertex-addition" planarity test [HT08], but the previous extension to obtain an embedding algorithm does not apply anymore. This paper proposes an extension to the Haeupler-Tarjan planarity test for computing a planar embedding. Our embedding algorithm is motivated, formally proven for correctness and linear runtime, and its implementation is evaluated in practice. Furthermore, we propose an additional extension of the Haeupler-Tarjan planarity test for computing SPQR-trees of biconnected input graphs. An SPQR-tree is a decomposition of a graph at its separation pairs, which has multiple practical applications. To our knowledge, this yields the first algorithm capable of simultaneously testing for planarity and computing a planar embedding as well as the SPQR-tree if the graph is planar.

Deutsche Zusammenfassung

Ein Planaritätstest ist ein Algorithmus, welcher auf Planarität eines Graphen testet, also ob dieser in die zweidimensionale Ebene gezeichnet werden kann, ohne dass sich Kanten überschneiden. Diese Algorithmen können zu Einbettungsalgorithmen erweitert werden, welche eine planare Einbettung des Eingabegraphen berechnen, also die zyklische Ordnung von Kanten um jeden Knoten in einer planaren Zeichnung des Graphen. Es wurden mehrere Linearzeit-Planaritätstests veröffentlicht, beispiels-weise der "vertex addition"-Test von Lempel, Even und Cederbaum in 1967 [LEC67], welcher im Jahr 1985 zu einem Einbettungsalgorithmus durch Chiba et al. erweitert wurde [CNAO85]. In 2008 wurde eine konzeptionell einfachere Version des "vertex addition"-Planaritätstests durch Haeupler und Tarjan veröffentlicht [HT08], aller-dings kann die vorige Erweiterung zu einem Einbettungsalgorithmus hier nicht mehr angewandt werden. In dieser Arbeit wird eine Erweiterung des Haeupler-Tarjan-Planaritätstests zur Einbettungsberechnung präsentiert. Der Algorithmus wird motiviert, dessen Korrektheit und lineare Laufzeit wird formal bewiesen, und dessen Implementierung wird in der Praxis evaluiert. Ferner wird eine zusätzliche Erweiterung des Haeupler-Tarjan-Planaritätstests vorgestellt, welche SPQR-Bäume zweifach zusammenhängender Graphen berechnet. Ein SPQR-Baum ist eine Dekomposition eines Graphen anhand seiner separierenden Knotenpaare, wobei diese Datenstruktur viele praktische Anwendungen hat. Nach unserem Wissen ergibt dies nun den ersten Algorithmus, der gleichzeitig einen Planaritätstest ausführt und eine planare Einbettung sowie den SPQR-Baum berechnet, sofern der Graph planar ist.

Contents

1. Introduction

A graph is considered planar if and only if its vertices and edge can be drawn into the two-dimensional plane such that no two edges cross one another. Planar graphs have many practical applications, for example for floor plans in building architecture [WKC88], integrated circuit design [AH92] and graph visualization [FC21]. Furthermore, a planar embedding of a planar graph describes the cyclic order of edges around every vertex in a planar drawing of that graph. A planarity test is an algorithm that decides whether a given input graph is planar. Hopcraft and Tarjan proposed the first linear-time planarity test in 1974 [HT74], using the so-called "path-addition" approach, which turned out to be complex in practice and which resulted in the need for further elaborating research [Tam13]. The "vertex-addition" approach, based on the work of Lempel, Even and Cederbaum in 1967 [LEC67], also later turned out to be realizable in linear time [Tam13]. A more detailed historical overview on research of planarity tests can be found in [Tam13]. All currently known planarity tests can be categorized into "cycle based algorithms" and "vertex-addition algorithms" [Tam13].

This paper uses the vertex-addition algorithm with a data structure called "PC-trees" devised by Haeupler and Tarjan in 2008 [HT08] as a basis for a new embedding algorithm of planar graphs. The approach is inspired by the embedding algorithm of Chiba et al. [CNAO85], which uses the "vertex-addition" planarity test from Lempel, Even and Cederbaum [LEC67] with "PQ-trees" as a basis. This embedding algorithm cannot be reused with the Haeupler-Tarjan planarity test due to constraints that where given in the planarity test from Lempel, Even and Cederbaum, e.g. the usage of a so-called *st*-numbering in order to avoid traversing the graph in a general DFS tree, but only in a single path. Our algorithm extends the Haeupler-Tarjan planarity test by gathering information from PC-trees during the planarity test and by adding a post-processing step to compute a planar embedding. The algorithm aims to be easy to understand conceptually, as PC-trees are conceptually simpler than PQ-tree [FPR21]. Additionally, PC-trees are faster in practice [FPR21], so we aim to translate this speed into our embedding algorithm.

Furthermore, this paper introduces an approach on how to compute an "SPQR-tree" from a graph by extending the Haeupler-Tarjan planarity test even further. An SPQR-tree represents a decomposition of a given biconnected graph into skeletons, which are separated by "separation pairs" of the graph, i.e. pairs of vertices whose removal disconnects the graph [FR23]. SPQR-trees where first introduced by Di Battista and Tamassia for usage in incremental planarity testing [DBT89], but now, they also find practical applications in integrated circuit design [CHH99] and business processes modeling [VVK09]. For a more

detailed overview of applications for SPQR-trees, see [FR23]. In 2001, Gutwenger and Mutzel proposed a linear time algorithm for computing SPQR-trees [GM01] based on the approach by Hopcraft and Tarjan [HT73]. For a given input graph, our approach performs the planarity test and calculates the SPQR-tree of the given input graph simultaneously. To our knowledge, this is the first algorithm that combines these steps.

In Chapter 2, basic definitions and concepts are presented that are necessary for understanding this thesis. Next, Chapter 3 presents our embedding algorithm. The algorithm is motivated and illustrated in Section 3.1. Furthermore, a formal proof of correctness can be found in Section 3.2 and Section 3.3 specifies the implementation, where we also show the runtime to be linear. In Chapter 4, we will describe our extension to the Haeupler-Tarjan planarity test with PC-trees for computing SPQR-tree of the input graph. Then, Chapter 5 presents the evaluation of our implementation of our embedding algorithm compared to the unmodified Haeupler-Tarjan planarity test and compared to the embedding algorithm and the planarity test found in the Open Graph Drawing Framework. Finally, Chapter 6 gives a summary of our findings and presents an outlook into possible further research.

2. Preliminaries

This chapter provides the basic definitions and concepts required by this thesis. Additionally, some conventions are established.

2.1. Graphs

Let V be a finite set of *vertices*. Given $E \subseteq \{\{v, w\} \mid v, w \in V \wedge v \neq w\}$, $G = (V, E)$ is an *undirected simple graph*. Given $E \subseteq V \times V$, $G = (V, E)$ is a *directed simple graph*. Given that E is a multiset of directed edges from $\{\{v, w\} \mid v, w \in V\}$, $G = (V, E)$ is an *undirected multigraph*. Given that E is a multiset of undirected edges from $V \times V$, $G = (V, E)$ is a *directed multigraph*. Given a graph G, $V(G)$ is the vertex set and $E(G)$ is the edge set of G. The two *endpoints* of an edge are the two vertices that an edge is attached to. Note that all graphs except undirected simple graphs may contain edges that represent *self-loops*, i.e. edges with two equal endpoints. Furthermore, for directed graphs $G = (V, E)$ we define source : $E \to V, (v, w) \mapsto v$ and target : $E \to V, (v, w) \mapsto w$. Given a graph G, the *degree* $\deg(v)$ of a vertex $v \in V(G)$ is the number of endpoints from all edges that are equal to v.

2.1.1. Walks and paths

Let $G = (V, E)$ be a graph. A *walk* a in G is an ordered set of vertices and edges $(x_1, e_1, x_2, e_2, x_3, \ldots, x_k, e_k, x_{k+1})$ with $k \in \mathbb{N}_0$, $x_1, \ldots, x_{k+1} \in V$ and $e_1, \ldots, e_k \in E$ such that $\forall i \in \{1, \ldots, k\} : e_i = (x_i, x_{i+1})$ if G is directed and $\forall i \in \{1, \ldots, k\} : e_i = \{x_i, x_{i+1}\}$ if G is undirected. Additionally, a has *length* k. A vertex v is called *reachable* from w if there exists a walk from v to w. A *path* p in G is a walk where no vertices nor edges are repeated. A *cycle* c in G is a walk where no vertices nor edges are repeated except the first and last vertices, which are equal. Two or more paths are called *independent* if they are pair-wise vertex-distinct with respect to their inner vertices, i.e. ignoring their endpoints [Die17]. If a path inside a graph G from $v \in V(G)$ to $w \in V(G)$ exists, then $v \underset{G}{\to} w$ denotes any such path.

2.1.2. Special graphs

A *forest* F is a simple graph that contains no cycles and no self loops. *Leaves* are vertices $v \in F(V)$ in forests with $\deg(v) \leq 1$, *inner vertices* are vertices $v \in F(V)$ with $\deg(v) > 1$. Note that every path in a forest is unique. A graph $G = (V, E)$ is *k-connected* if there exist k independent paths in G between any two vertices in V. A subgraph $G' = (V', E')$

Algorithm 2.1: Depth-first search

```
1  E_T ← ∅
2  E_B ← ∅ multiset
3  E_S ← ∅ multiset
4  Function DFS (G = (V, E))
5      forall v ∈ V do
6          v.visited ← false
7      forall v ∈ V do
8          if not v.visited then
               // e is a self-loop
9              DFS-CONNECTED(G, v, 0, null)
10 Function DFS-CONNECTED(G = (V, E), v, currentDepth, previousTreeEdge)
11     v.visited ← true
12     depth(v) ← currentDepth
13     forall e ∈ E with v ∈ e do
14         if e = {v, v} then
15             E_S ← E_S ∪ {(v, v)}
16         else
17             let w ∈ e such that w ≠ v
18             if not w.visited then
                   // e is a tree edge
19                 E_T ← E_T ∪ {(v, w)}
20                 DFS-CONNECTED(G, w, currentDepth +1, e)
21             else if e ≠ previousTreeEdge then
                   // e is a back edge
22                 E_B ← E_B ∪ {(v, w)}
```

of G with $V' \subseteq V$ and $E' \subseteq E$ is *k-connected in G* if there exist k independent paths in G between any pair of vertices in V'. We define *connected*, *biconnected* and *triconnected* as 1-connected, 2-connected and 3-connected respectively. A *tree* is a connected forest. Given a graph G, there exists a partition of $V(G)$ and $E(G)$ such that the corresponding vertex and edge sets of these partitions form connected graphs, the *connected components* of G.

2.1.3. Depth-first search

Let $G = (V, E)$ be an undirected graph. A *depth-first search (DFS)* is an algorithm that traverses all vertices of G. A recursive definition is given in Algorithm 2.1. The algorithm partitions the edges into *tree edges* E_T, *back edges* E_B and *self-loops* E_S. Note that all tree edges and back edges are directed, even though edges in E are undirected. The graph $T = (V, E_T)$ is a forest, the *DFS forest*. In case G is connected, T is also connected and therefore a tree, the *DFS tree*. A vertex $r \in V$ that has no incoming tree edge in T is called a *DFS root*. Given a vertex $u \in V$, subtree(u) denotes the subtree that is rooted at u, i.e. subtree(u) is the restriction on T that contains all vertices that are reachable from u. Furthermore, given a back edge $b \in E_B$, the *corresponding tree edge* of b is the tree edge $t \in E_T$ which, outgoing from target(b), points into the subtree that contains source(b), i.e. source(t) = target(b) and source(b) $\in$ subtree(target(t)). Algorithm 2.1 also computes the *depth* of every vertex v, which is the length of the path from the root of the connected component to v in T.

2.2. Planar graphs

The following definitions are similar to [Tam13]. A *drawing* D of a graph G is a function that maps all vertices of G to points in the two-dimensional real plane and that maps all edges of G to Jordan arcs with the endpoints of an arc being the mappings of the endpoints of the corresponding edge. A drawing D is *planar* if no vertex lies on an edge, except at its endpoints, if no vertices overlap and if there exist no *intersections*, also called *crossings*, of distinct edges in D, except for at their endpoints. A graph G is *planar* if there exists a planar drawing of G. A planar drawing D partitions the plane into connected regions, the *faces* of D. Since the graph is finite, there exists an unbounded face, the *outer face* of D.

2.2.1. Embeddings

To describe embeddings, we first need to introduce cyclic orders. A *partial cyclic order* C of a set or multiset X is an equivalency class of linear orders over a set $Y_C \subseteq X$ where for all linear orders $L, L' \in C$, we can obtain L' from L by cyclically shifting L, i.e. for $|Y_C| = n$ and $L = (x_0, \ldots, x_{n-1})$ with $x_1, \ldots, x_n \in X$, there exists $i \in \{0, \ldots, n-1\}$ such that $L' = (x_{i \bmod n}, x_{i+1 \bmod n}, \ldots, x_{i+n-1 \bmod n})$. A *(complete) cyclic order* C' of a set X is a partial cyclic order with $Y_{C'} = X$. Note that for illustration purposes, in this paper, cyclic orders are considered to be counter-clockwise. Given a partial cyclic order C over X, we write $(x_1, \ldots, x_k) \triangleright C$ with $x_1, \ldots, x_k \in X$ and $k \in \mathbb{N}$ if and only if there exists $L \in C$ such that the elements $x_1, \ldots, x_k$ appear in L in this order. Similarly, given a linear order P over a set X, we write $(x_1, \ldots, x_k) \triangleright P$ if and only if $x_1, \ldots, x_k$ appear in P in that order. Moreover, let $\zeta(X)$ be the set of all possible partial cyclic orders of X, and let $\zeta_C(X)$ be the set of all possible complete cyclic orders of X. A *cut* of a partial cyclic order C over $Y_C \subseteq X$ at an element $x \in Y_C$ is a linear order $L \in C$ such that x is the last element in L. Given a partial cyclic order C of X and a set $Y \subseteq X$, the restriction C_Y of C on Y is a partial cyclic order on Y that keeps all orderings of C on elements in Y. Additionally, we need the concept of consecutivity in cyclic orders: Let C be a partial cyclic order on a set X. Then $Y \subseteq X$ is *consecutive* in C if and only if for all $a, b \in Y$, for all $c, d \in X \setminus Y$ and for all $L \in C$, we have that $(a, c, b, d) \triangleright L$ does not hold. Furthermore, we say that $a, b \in X$ are consecutive in C if and only if $\{a, b\}$ is consecutive in C. Given a partial cyclic order C on X, as inspired by [Nov82], we define the *dual* flipped(C) of C as the set of all reversed linear orders from C, i.e. flipped$(C) := \{\text{reverse}(L) \mid L \in C\}$.

A *(combinatorial) embedding* $\mathcal{E}$ of a graph $G = (V, E)$ describes for every vertex $v \in V$ a cyclic order of all edges incident to v. Formally, an embedding $\mathcal{E}$ is a function $\mathcal{E} : V \to \zeta(V)$ such that for every $v \in V$, $\mathcal{E}(v)$ is a cyclic order of all edges incident to v (self-loops occur twice in the cyclic order). Given an embedding $\mathcal{E}$ of a graph $G = (V, E)$ and given a subset of edges $P \subseteq E$, the restriction $\mathcal{E}_P$ of $\mathcal{E}$ on P describes only the embedding of edges in P. Formally, $\mathcal{E}_P : V \to \zeta(V), v \mapsto (\mathcal{E}(v))_P$. Given a graph G and an embedding $\mathcal{E}$ of G, then a *face* f of G is defined by the cycle obtained from starting at a vertex v in G and walking along some edge e incident to G to $w \in V(G)$, then taking the next edge counter-clockwise in $\mathcal{E}(w)$, and repeating this until we arrive at the starting vertex.

2.2.2. Planar embeddings

The following definitions are similar to [Tam13]. A drawing D of a graph G prescribes a cyclic order of edges around each vertex and therefore describes an embedding $\mathcal{E}$ of G. In fact, there exist many drawings of G which describe the same embedding. Thus, an embedding $\mathcal{E}$ describes an equivalency class of drawings which all implement $\mathcal{E}$. $\mathcal{E}$ is *planar* if and only if there exists a planar drawing in the equivalency class of drawings described by $\mathcal{E}$. Note that every embedding of a forest (and thus of every tree) is planar and thus every forest and every tree is planar. If a graph is a cycle, then the graph is planar and a planar embedding of it has exactly two faces, the *outer face* and the *inner face*.

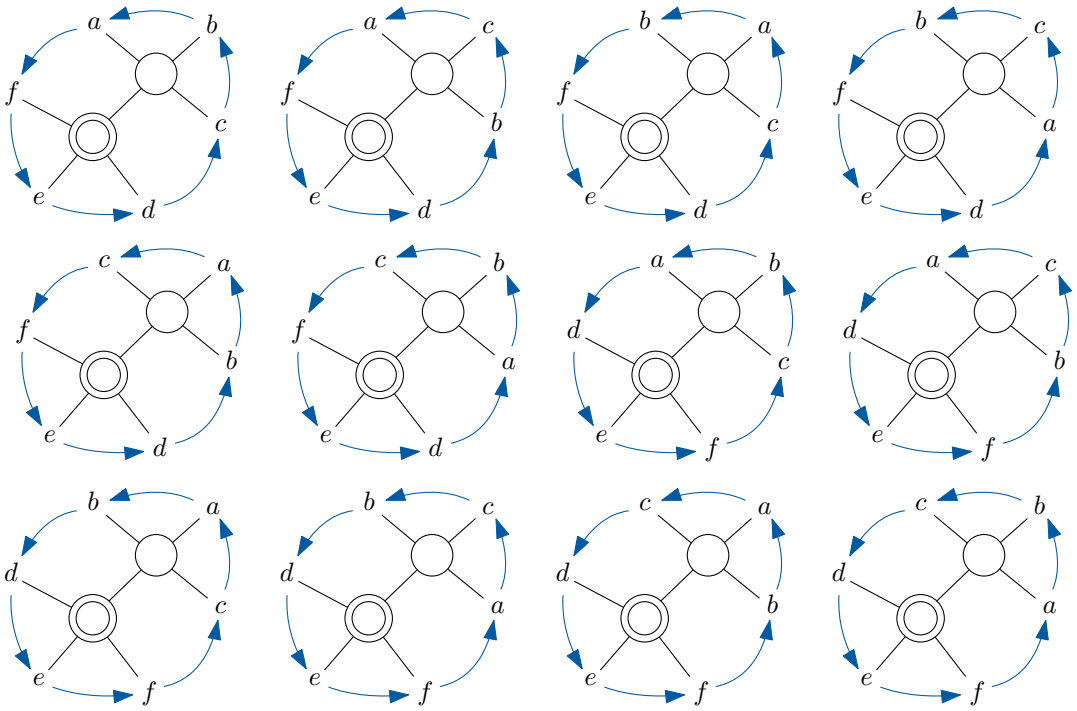


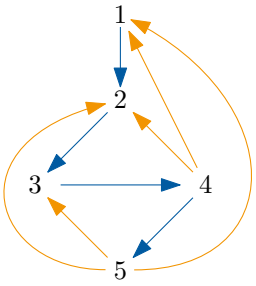
Figure 2.1.: Illustration of all possible embeddings for a PC-tree. P-nodes are illustrated using circles, C-nodes are double-circles. Circular orders are depicted using the blue arrows.

2.2.3. PC-trees

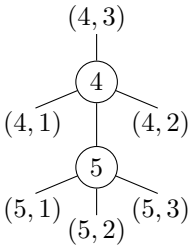
The following description of PC-trees is taken from [FPR21]. Given a finite set X, a *PC-tree* T aims to describe a set of cyclic orders $\text{ord}(T) \subseteq \zeta(X)$. T is an undirected tree, where every inner node is either a *P-node* or a *C-node* and every leaf is an element in X, the elements which are to be ordered. Additionally, for every C-node c, there exists a given cyclic order K_c of all edges connected to c. Now, we obtain the set of described cyclic orders $\text{ord}(T)$ by drawing T according to all possible embeddings $\mathcal{E}$ of T with $\mathcal{E}(c) = K_c$ or $\mathcal{E}(c) = \text{flipped}(K_c)$ for every C-node c and then by taking the cyclic order of encountered vertices $x \in X$ when tracing around the boundary of the outer face of the drawing for every such drawing of T. As an example, Figure 2.1 shows all possible embeddings and cyclic orders for a given PC-tree. Given a PC-tree T and $Y \subseteq X$, a *restriction* is an operation that produces a new PC-tree T' which restricts $\text{ord}(T)$ such that Y is consecutive in all cyclic orderings of T', i.e. $\text{ord}(T') \subseteq \text{ord}(T)$, Y is consecutive in C for all $C \in \text{ord}(T')$ and $\text{ord}(T')$ is maximal. A restriction on $Y \subseteq X$ in T is *impossible* if and only if such a restriction would produce a new PC-tree T' with $\text{ord}(T') = \emptyset$, i.e. if and only if there exists no $R \subseteq \text{ord}(T)$ such that $R \neq \emptyset$, $\text{ord}(T') \subseteq \text{ord}(T)$ and Y is consecutive in C for all $C \in R$. Given a restriction on $Y \subseteq X$, we assign labels to all nodes in T. A leaf $l \in X$ of T is *full* if $l \in Y$ and *empty* otherwise. Recursively, an inner node n of T is *full* if at least $\deg(n) - 1$ neighbors of n are full, otherwise n is *partial* if at least one neighbor of n is full, and empty otherwise. For a detailed description on how to efficiently compute restrictions on PC-trees and these labels of PC-nodes, see [FPR21].

2.2.4. Planarity test

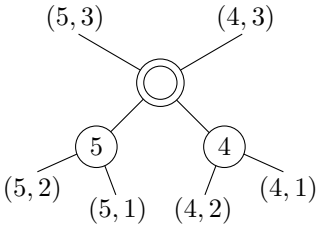
A planarity test is an algorithm that decides whether a given undirected graph G is planar [Tam13]. In the following, the Haeupler-Tarjan planarity test [HT08] is explained, which we will use as a basis for our algorithms in this paper. The planarity test uses a



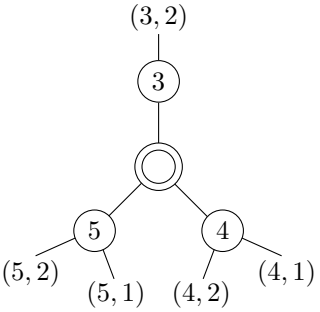
(a) Input graph with DFS. Blue edges represent tree edges, orange edges represent back edges.



(b) PC-tree before the retreat at the tree edge $(4, 3)$.



(c) PC-tree after the restriction for all edges connecting to 3.



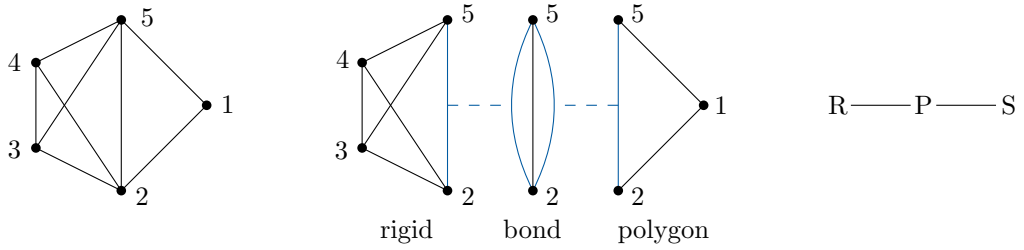
(d) PC-tree after deleting all full nodes and attaching to the P-node with label 3.

Figure 2.2.: Example of the retreat step for the tree edge $(4, 3)$.

DFS to traverse an input graph and then processes every vertex in reverse DFS discovery order. Here, PC-trees are used to track the cyclic orders of outgoing edges from the already processed subgraphs. In more detail, the algorithm performs the following steps to determine the planarity of an undirected input graph G. First, a DFS on G is started. This DFS traverses G and discovers tree edges, back edges and self-loops. Self-loops are ignored, since they have no influence on the planarity of G. When advancing along a tree edge (v, w), a new PC-tree is constructed that contains a P-node with label w and an attached leaf that represents the edge (w, v). When advancing along a back edge (v, w), we add a new leaf to the P-node with label v that represents the back edge (v, w). When retreating along a tree edge (v, w) (retreating from w to v), we perform a reduction on the PC-tree T of w (i.e. the PC-tree that contains or contained the label w) to make all leaves consecutive which represent edges connecting to v. Now, if this reduction is found to be impossible, then we stop the algorithm and G is determined not to be planar. Otherwise, the reduction is executed and all full nodes in T are deleted, such that the subtree T' remains. Now, T' is attached to another PC-tree at the P-node with label v. Finally, if the algorithm successfully executed without stopping, G is determined to be planar. In Figure 2.2, we see an example of the retreat step.

2.3. SPQR-trees

The following definitions are taken from [FR23]. Given a graph $G = (V, E)$, a *separating k-set* of G is a set S of k vertices of G such that $G - S$ has more connected components than G, where $G - S$ denotes the removal of all vertices of S and their incident edges in G. A separating 1-set is called a *cut-vertex*, a separating 2-set is a *separation pair*. A



(a) Graph G.

(b) Skeletons of G. Real edges are black, virtual edges are blue.

(c) The SPQR-tree of G.

Figure 2.3.: Example of an SPQR-tree.

separation pair v, w of a biconnected graph G splits G into *bridges*, which are the connected components of $G - v - w$. Now, the SPQR-tree D of a biconnected undirected multigraph $G = (V, E)$ represents a decomposition of G along its separation pairs in a tree with three node types, *P-nodes* that represent *bonds*, i.e. two vertices with at least three connecting edges, *S-nodes* that represent *polygons*, i.e. simple cycles, and *R-nodes* that represent *rigids*, i.e. triconnected components, such that no two P-nodes and no two S-nodes are adjacent in D. The bonds, polygons and rigids are then the *skeletons* of G. An edge of a skeleton is either a *real edge* if the corresponding edge also exists in G, or a *virtual edge* if that edge corresponds to a non-empty bridge. Note that a pair of virtual edges in different skeletons corresponds to an edge in the SPQR-tree. For a formal definition of SPQR-trees, see [FR23]. Note that for simplicity, this paper assumes that corresponding vertices in the graph and its skeletons are equal. In Figure 2.3, we see an example of an SPQR-tree.

2.4. Tuple ordering

Let $n \in \mathbb{N}$, let $v = \begin{pmatrix} v_1 \\ \vdots \\ v_n \end{pmatrix}, w = \begin{pmatrix} w_1 \\ \vdots \\ w_n \end{pmatrix} \in \mathbb{R}^n$. Then $v < w$ is defined as the *lexicographic order* on the components of v and w, i.e. $v < w$ if and only if there exists an $i \in \{1, \ldots, n\}$ such that $v_j = w_j$ for all $j \in \{1, \ldots, i-1\}$ and $v_i < w_i$. Moreover, $v \leq w$ holds if and only if $v = w$ or $v < w$.

3. Embedding Algorithm

In this chapter, we construct an algorithm that solves the following problem: Given an arbitrary undirected multigraph $G = (V, E)$, determine whether G is planar. Additionally, if this is the case, output an arbitrary planar embedding $\mathcal{E}$ of G. The algorithm should run in linear time $\mathcal{O}(|V| + |E|)$. In order to check for planarity, we resort to the planarity test by Haeupler and Tarjan [HT08]. If this test discovers that the input graph G is not planar, then there exists no planar embedding $\mathcal{E}$ of G. Otherwise, we need to compute $\mathcal{E}$, which is done by extending the planarity testing algorithm. In Section 3.1, the embedding algorithm is motivated and intuitively explained. Next, Section 3.2 elaborates on the correctness of our algorithm by devising a formal proof. Finally, Section 3.3 provides necessary implementation details and shows that the algorithm indeed runs in linear time.

3.1. Orderings

Let $G = (V, E)$ be a planar undirected multigraph. W.l.o.g. we can assume that G is connected, because for any graph that is not connected, we can consider its connected components separately. Based on a DFS with root r such that r is not a cut-vertex of G, let E_T, E_B, E_S be the sets of tree edges, back edges and self-loops, respectively, and let $T = (V, E_T)$ be the DFS tree. We refer to tree edges as *directed downwards*, directed away from the DFS root, and back edges are *directed upwards*. To describe the orderings in a planar embedding $\mathcal{E}$ of G, for all vertices v we first describe orderings of back edges connecting to the path $r \underset{T}{\rightarrow} v$. Afterwards, at each vertex, we distinguish different types of edges resulting from the DFS that connect to v: outgoing tree edges, outgoing back edges, incoming back edges, incoming tree edges and self-loops. These different types of edges are also illustrated in Figure 3.1.

3.1.1. Orderings at root paths

Let $v \in V$ and be the DFS root of the connected component that contains v, let $w_1, \ldots, w_k \in V$ be the DFS children of v and let $\mathcal{E}$ be a planar embedding of G. For all $w_i, w_j \in V$ of v with $i \neq j$, we observe that the DFS guarantees that there exist no edges with one endpoint in subtree(w_i) and the other one in subtree(w_j). For a vertex $u \in V$, let B_u be the set of all back edges originating in subtree(u) and let $B'_u \subseteq B_u$ be the set of all back edges originating in subtree(u) that also target vertices on the tree edge path $r \underset{T}{\rightarrow} u$ but that do not target u itself, i.e. depth(target(b)) $<$ depth(u) for all $b \in B'_u$. Additionally, let

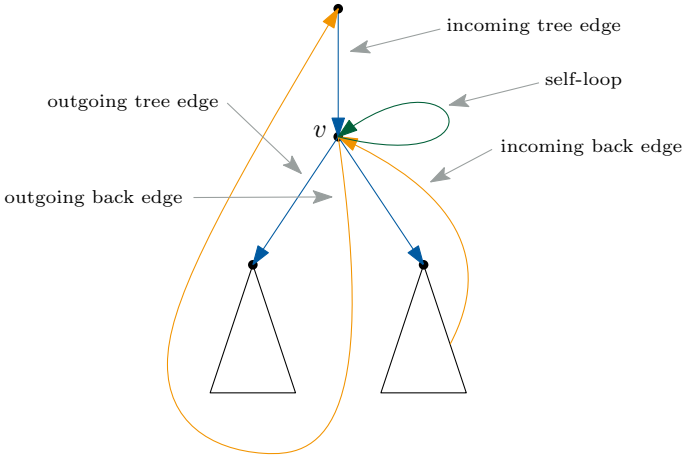


Figure 3.1.: Edge types.

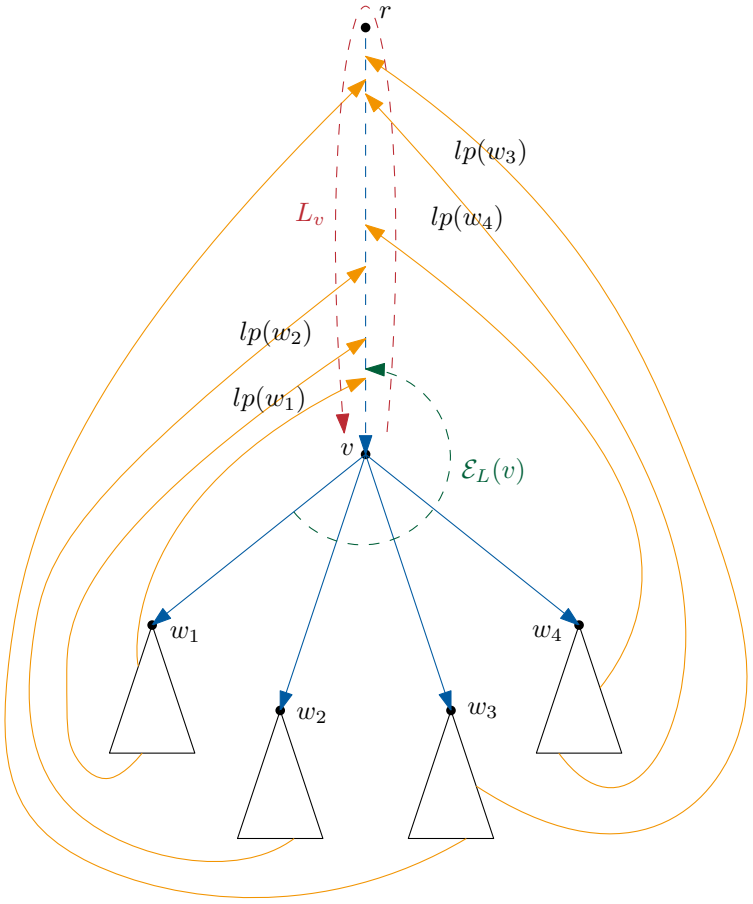


Figure 3.2.: Edge ordering around vertex v. The blue arrows denote tree edges or tree edge paths, the orange arrows denote back edges, the red arrow denotes the linear order around the path $r \xrightarrow[T]{} v$, the green arrow denotes the linear order of the embedding around v.

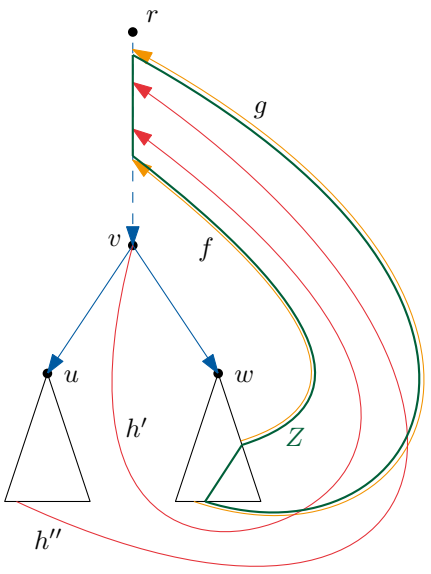


Figure 3.3.: Illustration of non-consecutive back edges during the proof of Lemma 3.1. h' and h'' show the two possibilities of h, where it either originates at v or in a subtree of a child of v.

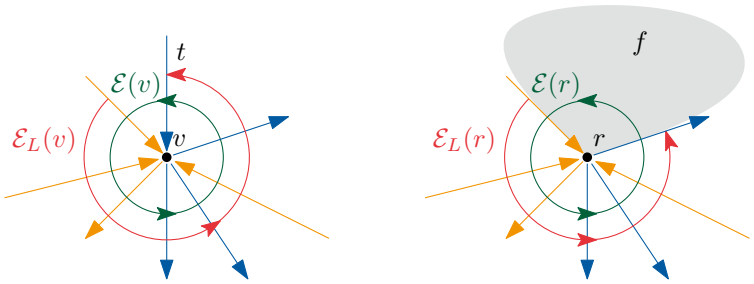
L_u be the linear order of back edges in B'_u obtained by tracing around $r \underset{T}{\rightarrow} u$ in a planar drawing of G that respects $\mathcal{E}$. As illustrated in Figure 3.2, the back edges in B'_v, form some kind of layers in a planar drawing. Specifically, there exist some restrictions on the linear order L_v that result from a planar embedding of G. This observation can be formalized using the following lemma.

Lemma 3.1. *Let $\mathcal{E}$ be a planar embedding of a connected planar undirected multigraph G, let there be a DFS on G with DFS tree T and with a non-cut-vertex root, let $v \in V(G)$, let $w \in V(G)$ be a DFS child of v from a DFS on G with non-cut-vertex root. Then B'_w is consecutive in L_v.*

Proof. Assume the statement is wrong, then there exist $f, g \in B'_w$ and $h \in E_B \setminus B'_w$ such that $(f, h), (h, g) \triangleright L_v$, where either $\text{source}(h) = v$ or $h \in B'_u$ with u being a DFS child of v and $u \neq w$. Let Z be the cycle in G formed by f, g, the tree edge path with endpoints $\text{source}(f)$ and $\text{source}(g)$ and the tree edge path with endpoints $\text{target}(f)$ and $\text{target}(g)$. The situation is now depicted in Figure 3.3. Since $(f, h), (h, g) \triangleright L_v$, h is connected to $\text{target}(h)$ at the inner face of Z. Additionally, v lies on the outer face of Z by definition of f and g, so $\text{source}(h)$ also lies on the outer face of Z, thus h is connected to $\text{source}(h)$ in the outer face of Z. Consequently, h must intersect Z in every planar drawing of G that respects $\mathcal{E}$, therefore $\mathcal{E}$ is not planar, a contradiction. □

3.1.2. Outgoing tree edges

Let $v \in V$ and let $w_1, \ldots, w_k \in V$ be all DFS children of v. Given a planar embedding $\mathcal{E}$ of G, we now focus on the outgoing tree edges originating at v to describe their orderings in $\mathcal{E}$. Based on the findings from Lemma 3.1, we define an equivalency relation $\sim_v$ on the set B'_v: $b \sim_v c$ if and only if $\text{source}(b) \neq v \neq \text{source}(c)$ and b, c originate in the subtree of the same child of v. Thus, we get the following equivalency classes for $\sim_v$: Every outgoing back edge at v has its own equivalency class and there exists an equivalency class B'_{w_i} for all $i \in \{1, \ldots, k\}$, corresponding to $\text{subtree}(w_i)$ and thus to the outgoing tree edge (v, w_i).



(a) Definition of $\mathcal{E}_L(v)$ with $v \neq r$.

(b) Definition of $\mathcal{E}_L(r)$.

Figure 3.4.: Example of the definition of $\mathcal{E}_L$.

Since here, we focus on outgoing tree edges from v, moving forward we only consider the second type of equivalency classes.

From Lemma 3.1, we conclude that all back edges in the same equivalency class are consecutive in L_v. Therefore, when describing the ordering in L_v of back edges in B'_{w_i} in relation to back edges in B'_{w_j} for $i, j \in \{1, \dots, k\}$ with $i \neq j$, we can instead describe the ordering in L_v by the relation of representatives of those classes. As a representative for the equivalency class $B'_{w_i} \neq \emptyset$ with $i \in \{1, \dots, k\}$, we choose the low pointer $\mathrm{lp}(w_i)$ of subtree(w_i), which is defined as follows:

$$\mathrm{lp}(w_i) \in B_{w_i} \text{ such that } \mathrm{depth}(\mathrm{target}(\mathrm{lp}(w_i))) = \min_{e \in B_{w_i}} \mathrm{depth}(\mathrm{target}(e))$$

Such a low pointer $\mathrm{lp}(w_i)$ may not exist because no back edges originate in subtree(w_i), i.e. $B_{w_i} = \emptyset$. In this case, we write $\mathrm{lp}(w_i) = \perp$. Additionally, $\mathrm{lp}(w_i)$ may not be uniquely defined, as there may exist back edges $b, d \in B_{w_i}$ with $b \neq d$ and $\mathrm{depth}(\mathrm{target}(b)) = \mathrm{depth}(\mathrm{target}(d))$. In this case, $\mathrm{lp}(w_i)$ refers to an arbitrary one of those candidates. Observe that since $B'_{w_i} \subseteq B_{w_i}$, $B'_{w_i} = \emptyset$ and $\mathrm{lp}(w_i) \neq \perp$ may hold. Obviously, in this case, $\mathrm{lp}(w_i)$ is not a representative of B'_{w_i}, but we see that in this case, all outgoing back edges from subtree(w_i) connect to v, so the position of the outgoing tree edge (v, w_i) among the other outgoing tree edges in $\mathcal{E}(v)$ does not matter, i.e. there exists a planar embedding $\mathcal{E}$ of G for every possible position of (v, w_i).

Now, using low pointers as representatives, we can show that there exist restrictions on orderings of outgoing tree edges at v for every planar embedding $\mathcal{E}$ of G. In general, we observe that when tracing around $r \underset{T}{\rightarrow} v$ in clockwise direction, we encounter low pointers of children of v in counter-clockwise order. For every embedding $\mathcal{E}$ of G and every vertex $v \in V \setminus \{r\}$ with incoming tree edge t into v, let $\mathcal{E}_L(v)$ be the cut of $\mathcal{E}(v)$ at t, i.e. $\mathcal{E}_L(v) \in \mathcal{E}(v)$ such that t is the last element in $\mathcal{E}_L(v)$. Additionally, we define $\mathcal{E}_L(r)$ as follows: Let f be a face of G according to $\mathcal{E}$ such that r is incident to f. Since r is no cut-vertex, f is incident to r at exactly one position in $\mathcal{E}(r)$. If we assume this is not the case, for every face f incident to r there would exist $(a, b, c, d) \triangleright \mathcal{E}(r)$ such that f is incident to r between a, b and between c, d respectively. By removing r, we would get that w.l.o.g. the components of G that are connected to r via a, c and b, d respectively would be separated by the face f' of $G - r$, which is the enlarged face f from the removal of r. Therefore, this contradicts r being no cut-vertex. Now, let $\mathcal{E}_L(r)$ be the cut of $\mathcal{E}(r)$ such that the edges next to the incidence point of f at r are the first and last elements in $\mathcal{E}_L(r)$. This definition of $\mathcal{E}_L$ is also illustrated in Figure 3.4. Note that $B'_r = \emptyset$, so L_r is also empty. Having defined $\mathcal{E}_L$, we now formalize our intuition on the layers visible in Figure 3.2, which are formed by back edges in $B'_{w_1}, \dots, B'_{w_k}$. We need this formalization later.

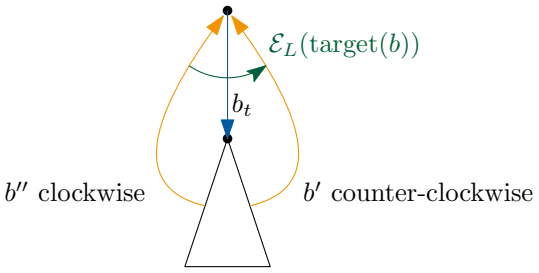


Figure 3.5.: Illustration of the definition of the orientations clockwise and counter-clockwise.

Lemma 3.2. *Let $\mathcal{E}$ be a planar embedding of a connected planar undirected multigraph G, let there be a DFS on G with DFS tree T and with a non-cut-vertex root, let $v \in V$, let a, c be outgoing tree edges from v, let $b \in B'_{\text{target}(a)}$ and $d \in B'_{\text{target}(c)}$. Then $(a, c) \triangleright \mathcal{E}_L(v) \iff (d, b) \triangleright L_v$.*

Proof. Assume $(a, c) \triangleright \mathcal{E}_L(v)$ and $(b, d) \triangleright L_v$. Let D be the cycle in G formed by d and the path $\text{target}(d) \underset{T}{\rightarrow} \text{source}(d)$. Since $(a, c) \triangleright \mathcal{E}_L(v)$, w.l.o.g. $\text{target}(a)$ lies in the outer face of D, and because of $(b, d) \triangleright L_v$, b connects to $\text{target}(b)$ at the inside of D. Therefore, the path $\text{source}(a) \underset{T}{\rightarrow} \text{source}(b)$ followed by b must cross D in every planar drawing of G that respects $\mathcal{E}$, a contradiction to $\mathcal{E}$ being planar. By assuming $(d, b) \triangleright L_v$ and $(c, a) \triangleright \mathcal{E}_L(v)$, we can make the same argument by swapping a, c and b, d respectively. □

In the following, we define the *orientations clockwise* and *counter-clockwise* for back edges as follows: Given a back edge $b \in E_B$, its corresponding tree edge $b_t \in E_T$ and a planar embedding $\mathcal{E}$ of G, b is clockwise if $(b, b_t) \triangleright \mathcal{E}_L(\text{target}(b))$ and b is counter-clockwise if $(b_t, b) \triangleright \mathcal{E}_L(\text{target}(b))$. This definition is also illustrated in Figure 3.5. We now want to describe the restrictions that are placed on outgoing tree edges in $\mathcal{E}_L(v)$ by Lemma 3.1. To achieve this, we first use a function $\sigma : E_B \to \mathbb{Z}$ with the following properties for all $v \in V$ and $b, c \in B'_v$:

Properties 3.3.

(i) $\sigma(b) < 0$ *if and only if b is clockwise*

(ii) $\sigma(b) > 0$ *if and only if b is counter-clockwise*

(iii) If b, c are clockwise and $\text{depth}(\text{target}(b)) > \text{depth}(\text{target}(c))$*, then* $\sigma(b) < \sigma(c)$

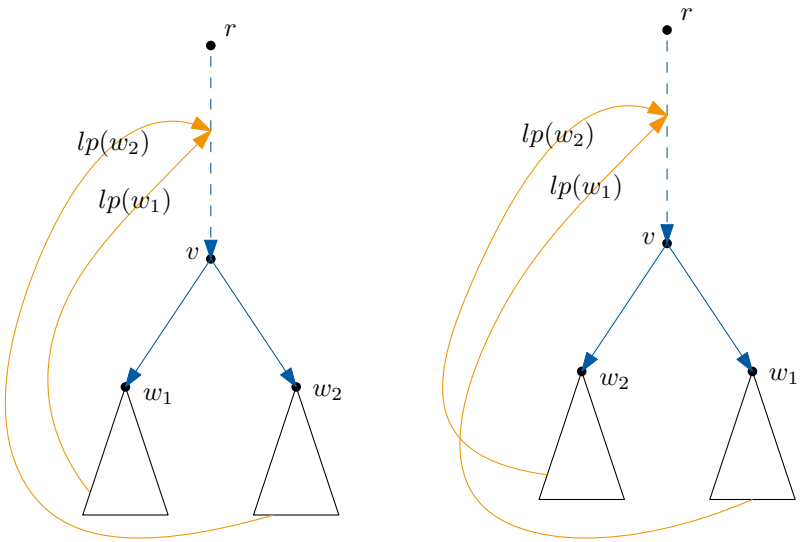
(iv) If b, c are counter-clockwise and $\text{depth}(\text{target}(b)) < \text{depth}(\text{target}(c))$*, then* $\sigma(b) < \sigma(c)$

(v) $\sigma(b) = \sigma(c)$ *if and only if b, c have the same orientation and* $\text{target}(b) = \text{target}(c)$

Note that σ depends on a given planar embedding and a given DFS. Using these properties, we get the following lemma.

Lemma 3.4. *Let $\mathcal{E}$ be a planar embedding of a connected planar undirected multigraph G, let there be a DFS on G with a non-cut-vertex root, let $v \in V$, let $b, c \in B'_v$ such that* $\text{target}(b) \neq \text{target}(c)$ *if b, c have the same orientation. Then $\sigma(b) < \sigma(c) \iff (c, b) \triangleright L_v$.*

Proof. Assume $\sigma(b) < \sigma(c)$. If both σ-vales have different signs, then $\sigma(b) < 0 < \sigma(c)$ must hold, so by Properties 3.3 (i) and 3.3 (ii) we have b being clockwise and c being counter-clockwise, thus we get $(c, b) \triangleright L_v$. Otherwise, w.l.o.g. let $0 < \sigma(b) < \sigma(c)$ (the other



(a) Ordering $(w_1, w_2) \triangleright \mathcal{E}_L(v)$ (b) Ordering $(w_2, w_1) \triangleright \mathcal{E}_L(v)$

Figure 3.6.: Example for $\sigma(\text{lp}(w_1)) = \sigma(\text{lp}(w_2))$. With $\mathcal{E}(\text{target}(\text{lp}(w_1)))$ being fixed, the relative ordering of w_1, w_2 in $\mathcal{E}(v)$ is not arbitrary.

case is analogous). Therefore, b and c have the same orientation by Property 3.3 (ii). From Property 3.3 (iv), we now get $\text{depth}(\text{target}(b)) < \text{depth}(\text{target}(c))$, thus $(c, b) \triangleright L_v$.

Assume $(b, c) \triangleright L_v$. If b and c have the same orientation, then b must be counter-clockwise and c must be clockwise, therefore Properties 3.3 (i) and 3.3 (ii) lead to $\sigma(b) < 0 < \sigma(c)$. Otherwise, w.l.o.g. let b and c are both counter-clockwise (the other case is analogous). Now, from $(c, b) \triangleright L_v$ and $\text{target}(b) \neq \text{target}(c)$ follows $\text{depth}(\text{target}(b)) < \text{depth}(\text{target}(c))$, hence we get $\sigma(b) < \sigma(c)$ by Property 3.3 (iv). □

With Lemma 3.4, we see that the function σ is sufficient for dealing with graphs where no two low pointers connect to the same node and have the same orientation. However, if such two low pointers b and c exist, $\sigma(b) = \sigma(c)$ holds by Property 3.3 (v). Therefore, for such cases, σ is not sufficient to describe the ordering of outgoing tree edges at v. An example of this is illustrated in Figure 3.6. We now define another function $\varphi : E_T \cup E_B \to \mathbb{Z}$ as a tie-breaker for such cases. Note that in Section 3.1.4, we will define $\varphi(e) = 0$ for all tree edges $e \in E_T$ to have a more compact notation. Until then, we only focus on values of φ for back edges. The function describes the relative ordering of incoming back edges at a vertex by having the following properties for all $b, c \in E_B$:

Properties 3.5.

(i) $\varphi(b) < 0$ *if and only if* b *is clockwise*

(ii) $\varphi(b) > 0$ *if and only if* b *is counter-clockwise*

(iii) If $\text{target}(b) = \text{target}(c)$ *and* b, c *have the same orientation, then* $\varphi(b) > \varphi(c) \iff (b, c) \triangleright \mathcal{E}_L(\text{target}(b))$

From Property 3.5 (iii), we also get the following statement.

Lemma 3.6. *Let* $b, c \in E_B$ *with* $\text{target}(b) = \text{target}(c)$ *and* b, c *having the same orientation. Then* $\varphi(b) > \varphi(c)$ *if and only if* $(c, b) \triangleright L_u$ *for all* $u \in (\text{target}(b) \underset{T}{\to} \text{source}(b) \cap \text{target}(c) \underset{T}{\to} \text{source}(c))$ *with* $\text{depth}(u) > \text{depth}(\text{target}(b))$.

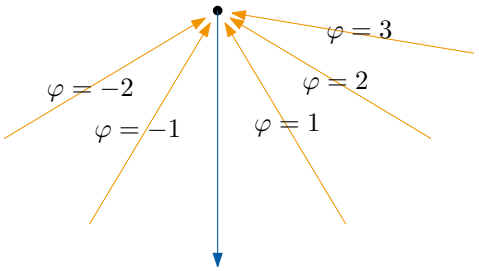


Figure 3.7.: Example of assigned φ-values.

Proof. Simply follows from Property 3.5 (iii) and the definitions of $\mathcal{E}_L$ and L_u. □

Note that here, φ depends on a given planar embedding and on a given DFS. For an exemplar assignment of φ-values to back edges, see Figure 3.7. Based on these properties for φ, we can now describe the ordering of outgoing tree edges in $\mathcal{E}_L(v)$ even further.

Lemma 3.7. *Let $\mathcal{E}$ be a planar embedding of a connected planar undirected multigraph G, let there be a DFS on G with a non-cut-vertex root, let $v \in V$, let $b, c \in B'_v$. Then*

$$\begin{pmatrix} \sigma(b) \\ -\varphi(b) \end{pmatrix} < \begin{pmatrix} \sigma(c) \\ -\varphi(c) \end{pmatrix} \iff (c, b) \triangleright L_v$$

Proof. If $target(b) \neq \text{target}(c)$ or b, c have the same orientation, the statement simply follows from Lemma 3.4. Therefore, now assume $\text{target}(b) = \text{target}(c)$ and b, c having the same orientation. Hence, $\sigma(b) = \sigma(c)$ holds due to Property 3.3 (v). Assuming $\begin{pmatrix} \sigma(b) \\ -\varphi(b) \end{pmatrix} < \begin{pmatrix} \sigma(c) \\ -\varphi(c) \end{pmatrix}$ holds, then we get $\varphi(b) > \varphi(c)$ due to $\sigma(b) = \sigma(c)$. Consequently, we get $(c, b) \triangleright L_v$ from Lemma 3.6. Assuming $(c, b) \triangleright L_v$ holds, we get $\varphi(b) > \varphi(c)$ due to Lemma 3.6. Hence, $\begin{pmatrix} \sigma(b) \\ -\varphi(b) \end{pmatrix} < \begin{pmatrix} \sigma(c) \\ -\varphi(c) \end{pmatrix}$. □

Now, based on Lemma 3.1, we can use low pointers as representatives of subtrees and apply Lemma 3.7, which gives us our ordering of outgoing tree edges due to Lemma 3.2.

Lemma 3.8. *Let G be a connected planar undirected multigraph, let $v \in V(G)$, let u, w be DFS children of v based on a DFS on G with a non-cut-vertex root, and let e_u, e_w be the respective tree edges. If $B'_u \neq \emptyset \neq B'_w$, then for any planar embedding $\mathcal{E}$ of G, we have:*

$$\begin{pmatrix} \sigma(\text{lp}(u)) \\ -\varphi(\text{lp}(u)) \end{pmatrix} < \begin{pmatrix} \sigma(\text{lp}(w)) \\ -\varphi(\text{lp}(w)) \end{pmatrix} \iff (e_u, e_w) \triangleright \mathcal{E}_L(v)$$

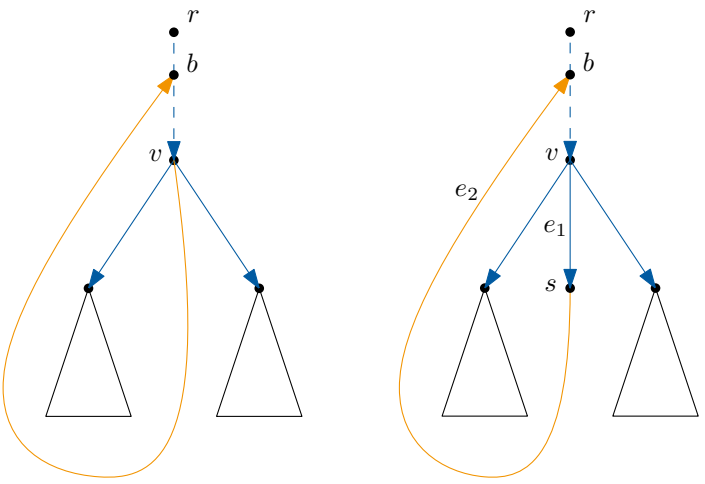
Additionally, there exists a planar embedding $\mathcal{E}$ of G such that we have

$$\begin{pmatrix} \sigma(\text{lp}(u)) \\ -\varphi(\text{lp}(u)) \end{pmatrix} \leq \begin{pmatrix} \sigma(\text{lp}(w)) \\ -\varphi(\text{lp}(w)) \end{pmatrix} \iff (e_u, e_w) \triangleright \mathcal{E}_L(v)$$

where $\sigma(\perp) := 0$ and $\varphi(\perp) := 0$.

Proof. Assume $B'_u \neq \emptyset \neq B'_w$, then we get $\text{lp}(u) \neq \perp \neq \text{lp}(w)$ and $\text{lp}(u), \text{lp}(w)$ act as representatives for the equivalency classes B'_u, B'_w respectively. Let $\mathcal{E}$ be a planar embedding of G, then we have

$$\begin{pmatrix} \sigma(\text{lp}(u)) \\ -\varphi(\text{lp}(u)) \end{pmatrix} < \begin{pmatrix} \sigma(\text{lp}(w)) \\ -\varphi(\text{lp}(w)) \end{pmatrix} \underset{\text{Lemma 3.7}}{\iff} (\text{lp}(w), \text{lp}(w)) \triangleright L_v \underset{\text{Lemma 3.2}}{\iff} (e_u, e_w) \triangleright \mathcal{E}_L(v)$$



(a) G (b) G'

Figure 3.8.: Example of G and its subdivision G'.

Additionally, if $B'_u = \emptyset$, then subtree(u) only has connections to v, so e_u may be placed in an arbitrary position in $\mathcal{E}_L(v)$ among the other outgoing tree edges. Therefore, in this case, the value of $\begin{pmatrix} \sigma(\mathrm{lp}(u)) \\ -\varphi(\mathrm{lp}(u)) \end{pmatrix}$ may be arbitrarily chosen, which implies the existence of a planar embedding $\mathcal{E}$ that satisfies the above equivalence. Ditto for e_w if $B'_w = \emptyset$. □

3.1.3. Outgoing back edges

We now focus on the ordering of outgoing back edges from v in a planar embedding $\mathcal{E}$ of G. Let $e \in E_B$ be such an outgoing back edges at v, let $b := \mathrm{target}(e)$. This setting can now be reduced to handling an outgoing tree edge: Consider a graph G' which is a subdivision of G where a vertex $s \notin V$ splits the edge e into $e_1, e_2 \notin E$ such that $e_1 = \{v, s\}$ and $e_2 = \{s, b\}$. An example for G and G' can be seen in Figure 3.8. Obviously, for every planar embedding $\mathcal{E}$ of G, we have a corresponding planar embedding $\mathcal{E}'$ of G' where e_1 has the same position in $\mathcal{E}'(v)$ as e has in $\mathcal{E}(v)$ and e_2 has the same position in $\mathcal{E}'(b)$ as e has in $\mathcal{E}(b)$. Considering Lemma 3.8 and given that e and e_2 have the same orientation, we have $\mathrm{lp}(s) = e_2$ and thus $\sigma(\mathrm{lp}(s)) = \sigma(e_2) = \sigma(e)$ by Property 3.3 (v). Additionally, for all $d \in E_B$ with $\mathrm{target}(d) = \mathrm{target}(\mathrm{lp}(s))$ we have $\varphi(\mathrm{lp}(s)) < \varphi(d) \implies \varphi(e_2) < \varphi(d)$ as well as $\varphi(\mathrm{lp}(s)) > \varphi(d) \implies \varphi(e_2) > \varphi(d)$ based on Properties 3.5 (i) to 3.5 (iii). Therefore, w.l.o.g. we can assume $\varphi(\mathrm{lp}(s)) = \varphi(e_2)$. We now conclude our description of orderings for outgoing tree and back edges at v:

Lemma 3.9. *Let G be a connected planar undirected multigraph, let there be a DFS on G with a non-cut-vertex root, let $v \in V(G)$. For an edge e that is incident to v and $w \in e$ being the other endpoint of e, we define $M(e)$ depending on the type of e as follows. Note that $\sigma(\perp) := 0$, $\varphi(\perp) := 0$.*

Type of e	*outgoing tree edge*	*outgoing back edge*
$M(e)$	$\begin{pmatrix} \sigma(\mathrm{lp}(w)) \\ -\varphi(\mathrm{lp}(w)) \end{pmatrix}$	$\begin{pmatrix} \sigma(e) \\ -\varphi(e) \end{pmatrix}$

Let g, h be outgoing tree edges or outgoing back edges at v. If $e \in E_B \implies B'_{\mathrm{target}(e)} \neq \emptyset$ for all $e \in \{g, h\}$, then for any planar embedding $\mathcal{E}$ of G, we have:

$$M(g) < M(h) \iff (g, h) \triangleright \mathcal{E}_L(v)$$

Additionally, there exists a planar embedding $\mathcal{E}$ of G such that

$$M(g) \leq M(h) \iff (g,h) \triangleright \mathcal{E}_L(v)$$

Proof. Simply follows from the above construction of G' and Lemma 3.8. □

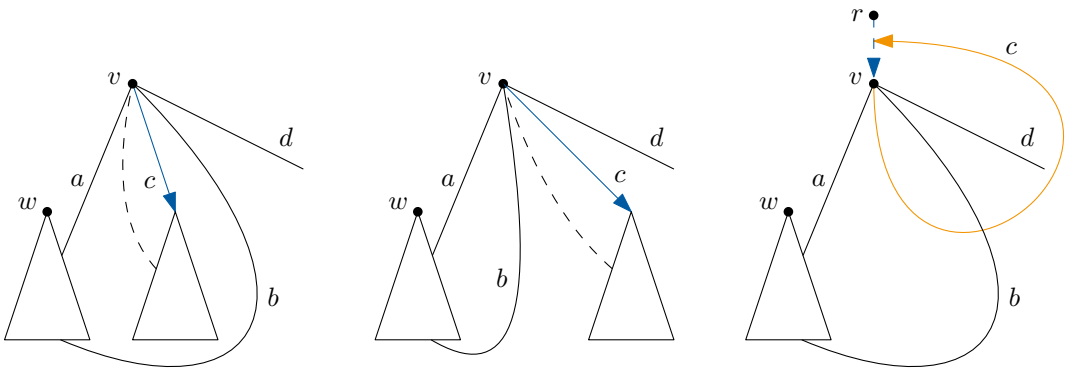
3.1.4. Incoming back edges

Now, we address the ordering of incoming back edges at a vertex in a planar embedding $\mathcal{E}$ of G. Given a vertex $w \in V$ with $w \neq r$, we define I_w to be the set of all back edges originating in subtree(w) that are incoming back edges into the DFS parent of w. We now show that all incoming back edges and their respective outgoing tree edge may be consecutive in a planar embedding.

Lemma 3.10. *Let G be a connected planar undirected multigraph, let there be a DFS on G with non-cut-vertex root r, let $\mathcal{E}$ be a planar embedding of G. Then there exists a planar embedding $\mathcal{E}'$ of G such that all outgoing tree edges, outgoing back edges and incoming tree edges have the same relative positions in $\mathcal{E}'$ as in $\mathcal{E}$ and such that for all $w \in V(G) \setminus \{r\}$ with incoming tree edge t, $I_w \dot\cup \{t\}$ is consecutive in $\mathcal{E}'(\text{parent}(w))$.*

Proof. Let $w \in V(G) \setminus \{r\}$ with incoming tree edge t such that $I_w \dot\cup \{t\}$ is not consecutive in $\mathcal{E}(\text{parent}(w))$. Let $v := \text{parent}(w)$. There now must exist $a, b \in I_w \dot\cup \{t\}$ and $(c,d) \triangleright \mathcal{E}(v) \setminus (I_w \dot\cup \{t\})$ such that $(a,c,b,d) \triangleright \mathcal{E}(v)$. We now show that there exists a planar embedding $\mathcal{E}'$ such that $(a,b,c,d) \triangleright \mathcal{E}'(v)$, which is sufficient to prove this lemma. Consider the cycle Z formed by a, b and the tree edge path between the endpoints of a and b in subtree(w). We see that $v \in Z$ because v is an endpoint of a. Due to $(a,c,b,d) \triangleright \mathcal{E}(v)$, c connects at the inside of Z and d connects at the outside of Z. First, assume that c is an outgoing tree edge from v. This case is illustrated in Figure 3.9a. Since c connects at the inside of Z and $\mathcal{E}$ is planar, subtree(target(c)) must only have connections to v, as connections to subtree(w) would violate the DFS and edges to other parts of the DFS tree would cross Z in every drawing that respects $\mathcal{E}$, violating the planarity of $\mathcal{E}$. Therefore, we can create another planar embedding $\mathcal{E}'$ of G from $\mathcal{E}$ by moving the connections to subtree(target(c)) in $\mathcal{E}(v)$ to the outside of Z, i.e. $(a,b,e,d) \triangleright \mathcal{E}'(v)$ for all $e \in I_{\text{target}(c)} \cup \{c\}$. This is also illustrated in Figure 3.9b. Next, assume that c is an outgoing back edge at v. This case is illustrated in Figure 3.9c. Then c connects to v at the inside of Z, but it also connects to $\text{target}(c) \triangleright r \underset{T}{\rightarrow} v$, $\text{target}(c) \neq v$, on the outside of Z. Thus, c would cross Z in every drawing that respects $\mathcal{E}$, contradicting the planarity of $\mathcal{E}$, so this assumption is wrong. Next, assume that c is an incoming back edge at v. This case is illustrated in Figure 3.9d. Since $c \notin I_w$, c must originate in another subtree. Let u be the root of this subtree such that u is a child of v. Given that $\mathcal{E}$ is planar, subtree(u) must lie on the inside of Z. Therefore, there exists an outgoing back edge s from v to u such that $(a,s,b,d) \triangleright \mathcal{E}(v)$. We now apply the above case concerning outgoing back edges. Finally, assume that c is the incoming tree edge at v, so v is not a DFS root. This case is illustrated in Figure 3.9e. Therefore, the subgraph $G - \text{subtree}(v)$ lies on the inside of Z. Thus, for all children u of v with $u \neq w$, subtree(u) has no connections to $G - \text{subtree}(v)$, i.e. $B'_u = \emptyset$. Let $f, g \in \{a, b\}$ such that $f \neq g$ and $f \in I_w$ and w.l.o.g. $(g,c,f,d) \triangleright \mathcal{E}(v)$. Now, we can create another planar embedding $\mathcal{E}'$ of G from $\mathcal{E}$ by moving f to the other far side of I_w in $\mathcal{E}(v)$, i.e. such that $(f,g,c,d) \triangleright \mathcal{E}'(v)$. This is also illustrated in Figure 3.9f. Then $\mathcal{E}'$ is also planar because Z separates the same parts of subtree(w) and $G - \text{subtree}(v)$ in $\mathcal{E}'$ as in $\mathcal{E}$. □

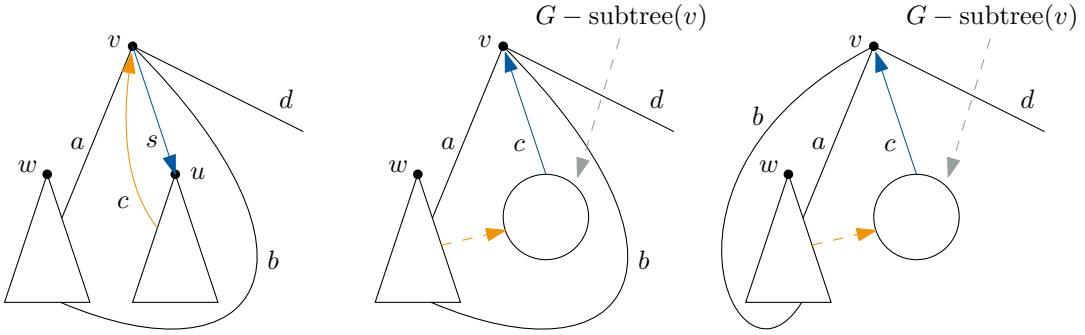
Based on this finding, we can attempt to use the ordering values shown in Lemma 3.8 for the respective outgoing tree edges to describe a possible embedding of G, but we must



(a) Case where c is outgoing tree edge, illustration of $\mathcal{E}$.

(b) Case where c is outgoing tree edge, illustration of $\mathcal{E}'$.

(c) Case where c is outgoing back edge.

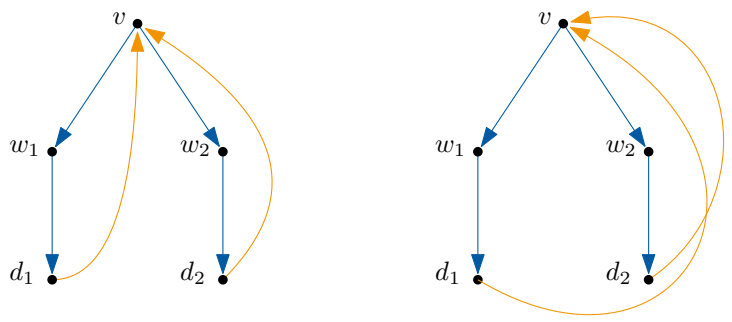


(d) Case where c is incoming back edge.

(e) Case where c is incoming tree edge, illustration of $\mathcal{E}$.

(f) Case where c is incoming tree edge, illustration of $\mathcal{E}'$.

Figure 3.9.: Illustration for the proof of Lemma 3.10.



(a) Correct ordering with consecutivity.

(b) Incorrect ordering without consecutivity

Figure 3.10.: Example of ambiguous ordering of incoming back edges. Here, $\sigma(w_1) = \sigma(w_2)$ and $\varphi(\mathrm{lp}(w_1)) = \varphi(\mathrm{lp}(w_2))$ hold.

consider the following edge case: For $v \in V$ and for u, w being DFS children of v, we may have $\sigma(\mathrm{lp}(u)) = \sigma(\mathrm{lp}(w))$ and $\varphi(\mathrm{lp}(u)) = \varphi(\mathrm{lp}(w))$, suggesting an ambiguous ordering. An example of such a case is shown in Figure 3.10. Now, to use as a tie-breaker, we introduce an injective function $N : V \to \mathbb{Z}$, which maps every vertex to a unique identifier. The value of this function is used to group edges by subtrees. This ensures that an embedding as in Lemma 3.10 is always achieved. Next, we need to describe the ordering of incoming back edges and their respective outgoing tree edge, i.e. the set $I_w \cup \{t\}$, among themselves. For this, we use the values of $\varphi(e)$ for all $c \in I_w$ and the value 0 for the tree edge. As referenced at the definition of φ, we here define $\varphi(e) = 0$ for all tree edges $e \in E_T$. We now get the following lemma.

Lemma 3.11. *Let G be a connected planar undirected multigraph, let there be a DFS on G with a non-cut-vertex root, let $\mathcal{E}$ be a planar embedding of G that satisfies Lemma 3.10, let $v \in V(G)$, let w be a child of v, let t be the tree edge from v to w. Then we have $\varphi(a) < \varphi(b) \iff (a, b) \triangleright \mathcal{E}_L(v)$ for all $a, b \in I_w \cup \{t\}$.*

Proof. Let $a, b \in I_w \cup \{t\}$. W.l.o.g. assume that $\varphi(a) \geq 0$ (the other case is symmetric). Assume that $\varphi(a) < \varphi(b)$ holds. If $\varphi(a) = 0$, then $a = t$, so b is a back edge and a its corresponding tree edge. Now, $\varphi(b) > 0$, so b is counter-clockwise by Property 3.5 (ii). Consequently, $(a, b) \triangleright \mathcal{E}_L(v)$ holds by definition of counter-clockwise. If $\varphi(a) > 0$, then a, b are both back edges and by Property 3.5 (ii), they are both counter-clockwise. From Property 3.5 (iii), we now get $(a, b) \triangleright \mathcal{E}_L(v)$. Assume that $(a, b) \triangleright \mathcal{E}_L(v)$ holds. If $a = t$, then $\varphi(a) = 0$. Because of $(a, b) \triangleright \mathcal{E}_L(v)$, b is counter-clockwise, so $\varphi(b) > 0$ based on Property 3.5 (ii). Hence, we have $\varphi(a) = 0 < \varphi(b)$. If $a \neq t$, then $\varphi(a) \neq 0$ by Properties 3.5 (i) and 3.5 (ii), so $\varphi(a) > 0$ and a, b are both back edges. Therefore, a is counter-clockwise due to Property 3.5 (ii). Since $(a, b) \triangleright \mathcal{E}_L(v)$, we get $\varphi(a) < \varphi(b)$ by Property 3.5 (iii). □

Based on the above insights, we now get the following description for orderings of incoming back edges:

Lemma 3.12. *Let G be a connected planar undirected multigraph, let there be a DFS on G with a non-cut-vertex root, let $v \in V(G)$. For an edge e that is incident to v and $w \in e$ being the other endpoint of e, we define $M(e)$ depending on the type of e as follows. Note that $\sigma(\perp) := 0$, $\varphi(\perp) := 0$ and x is the target of the corresponding tree edge of e if $e \in E_B$.*

Type of e	*outgoing tree edge*	*outgoing back edge*	*incoming back edge*
$M(e)$	$\begin{pmatrix} \sigma(\mathrm{lp}(w)) \\ -\varphi(\mathrm{lp}(w)) \\ N(w) \\ 0 \end{pmatrix}$	$\begin{pmatrix} \sigma(e) \\ -\varphi(e) \\ 0 \\ 0 \end{pmatrix}$	$\begin{pmatrix} \sigma(\mathrm{lp}(x)) \\ -\varphi(\mathrm{lp}(x)) \\ N(x) \\ \varphi(e) \end{pmatrix}$

Let g, h be outgoing tree, incoming back edges or outgoing back edges at v. Then there exists a planar embedding $\mathcal{E}$ of G such that

$$M(g) \leq M(h) \iff (g, h) \triangleright \mathcal{E}_L(v)$$

Proof. Simply follows from Lemmas 3.9 to 3.11 and the construction of the vertex identifier function N. □

3.1.5. Incoming tree edges

Let $v \in V \setminus \{r\}$. By definition of $\mathcal{E}_L(v)$, the incoming tree edge t into v is the last element in $\mathcal{E}_L(v)$. Therefore, we use the value ∞ to describe the ordering of incoming tree edges among all other edges.

Lemma 3.13. *Let G be a connected planar undirected multigraph, let there be a DFS on G with a non-cut-vertex root, let $v \in V(G)$. For an edge e that is incident to v and $w \in e$ being the other endpoint of e, we define $M(e)$ depending on the type of e as follows. Note that $\sigma(\perp) := 0$, $\varphi(\perp) := 0$ and x is the target of the corresponding tree edge of e if $e \in E_B$.*

Type of e	*outgoing tree edge*	*outgoing back edge*	*incoming back edge*	*incoming tree edge*
$M(e)$	$\begin{pmatrix} \sigma(\mathrm{lp}(w)) \\ -\varphi(\mathrm{lp}(w)) \\ N(w) \\ 0 \end{pmatrix}$	$\begin{pmatrix} \sigma(e) \\ -\varphi(e) \\ 0 \\ 0 \end{pmatrix}$	$\begin{pmatrix} \sigma(\mathrm{lp}(x)) \\ -\varphi(\mathrm{lp}(x)) \\ N(x) \\ \varphi(e) \end{pmatrix}$	$\begin{pmatrix} \infty \\ 0 \\ 0 \\ 0 \end{pmatrix}$

Let g, h be outgoing tree, incoming back edges, outgoing back edges or incoming tree edges at v. Then there exists a planar embedding $\mathcal{E}$ of G such that

$$M(g) \leq M(h) \iff (g, h) \triangleright \mathcal{E}_L(v)$$

Proof. Simply follows from Lemma 3.12 and the definition of $\mathcal{E}_L$. □

3.1.6. Self-loops

Dealing with self-loops is conceptually trivial. For every vertex $v \in V$, let $S_v \subseteq E_S$ be the set of self-loops that connect to v. We now get the following result.

Lemma 3.14. *Let G be a connected planar undirected multigraph, let there be a DFS on G with a non-cut-vertex root, let $\mathcal{E}$ be a planar embedding of G, Then there exists an embedding $\mathcal{E}'$ such that all tree edges and back edges in $\mathcal{E}$ have the same relative positions in $\mathcal{E}$ as in $\mathcal{E}'$ and such that for every vertex $v \in V(G)$ and for every self loop $s \in S_v$, the two appearances of s in $\mathcal{E}(v)$ are consecutive.*

Proof. Let $v \in V(G)$, let $S_v = \{s_1, s_2, \dots, s_l\}$. Then define $\mathcal{E}'_L(v) := (s_1, s_1, s_2, s_2, \dots, s_l, s_l) \cup (\mathcal{E}_L(v) \setminus S_v)$, where $\cup$ denotes concatenation. □

To achieve such an ordering, we first use the value $-\infty$ to move all self-loops to the front of $\mathcal{E}_L(v)$. Then, we use an injective function $\iota : E_S \to \mathbb{Z}$ to assign identifiers to all self-loops for grouping their endpoints, as shown in the proof of Lemma 3.14. Using this, we get the following ordering for all edges.

Lemma 3.15. *Let G be a connected planar undirected multigraph, let there be a DFS on G with a non-cut-vertex root, let $v \in V(G)$. For an edge e that is incident to v and $w \in e$ being the other endpoint of e, we define $M(e)$ depending on the type of e as follows. Note that $\sigma(\perp) := 0$, $\varphi(\perp) := 0$ and x is the target of the corresponding tree edge of e if $e \in E_B$.*

Type of e	*outgoing tree edge*	*outgoing back edge*	*incoming back edge*	*incoming tree edge*	*self-loop*
$M(e)$	$\begin{pmatrix} \sigma(\mathrm{lp}(w)) \\ -\varphi(\mathrm{lp}(w)) \\ N(w) \\ 0 \end{pmatrix}$	$\begin{pmatrix} \sigma(e) \\ -\varphi(e) \\ 0 \\ 0 \end{pmatrix}$	$\begin{pmatrix} \sigma(\mathrm{lp}(x)) \\ -\varphi(\mathrm{lp}(x)) \\ N(x) \\ \varphi(e) \end{pmatrix}$	$\begin{pmatrix} \infty \\ 0 \\ 0 \\ 0 \end{pmatrix}$	$\begin{pmatrix} -\infty \\ \iota(e) \\ 0 \\ 0 \end{pmatrix}$

Let g, h be edges at v. Then there exists a planar embedding $\mathcal{E}$ of G such that

$$M(g) \leq M(h) \iff (g, h) \triangleright \mathcal{E}_L(v)$$

Proof. Simply follows from Lemmas 3.13 and 3.14. □

3.2. Correctness

Given a planar undirected multigraph G, we have now seen that there exists a planar embedding $\mathcal{E}$ of G that satisfies the orderings described in Lemmas 3.1, 3.8, 3.9, 3.12, 3.13 and 3.15. Note that Lemma 3.15 also summarizes the Lemmas 3.8, 3.9, 3.12 and 3.13. Now, we have no planar embedding of G given, and we want to obtain a planar embedding of G. We do this by assigning values to the functions N, ι φ, lp and σ, such that, based on a DFS of G, Lemma 3.1, Properties 3.3 (i) to 3.3 (v) for σ, and Properties 3.5 (i) to 3.5 (iii) of φ are satisfied for some planar embedding $\mathcal{E}$, and such that the definitions for N, ι and lp are satisfied (which do not depend on a planar embedding). Lemma 3.1 and the properties for φ are ensured by using PC-trees, as we will describe in Section 3.3.1.2, and the properties of σ are ensured by computing σ using a formula, as we will describe in Section 3.3.1.4. Next, we compute the cyclic orderings $\mathcal{E}(v)$ for every $v \in V(G)$ based on Lemma 3.15. We thus obtain an embedding $\mathcal{E}$ of G. In this section, we prove that this resulting embedding is indeed planar. Afterwards, in Section 3.3, we discuss more details on to how compute the embedding. We now show the following theorem.

Theorem 3.16. *Let G be an undirected planar multigraph, let $\mathcal{E}$ be a combinatorial embedding that respects the Lemmas 3.1 and 3.15. Then $\mathcal{E}$ is planar.*

Let $G = (V, E)$ be a planar undirected multigraph. W.l.o.g. we can assume that G is connected, since Lemmas 3.1 and 3.15 independently apply to all connected components of G. Let $\mathcal{E}$ an embedding that respects the orderings from Lemmas 3.1 and 3.15. The beginning of our proof is analogous to [Bra09]. Let T be the DFS tree of G based on some DFS with non-cut-vertex root, let $E_T \dot\cup E_B \dot\cup E_S$ the partition of edges in E from the DFS.

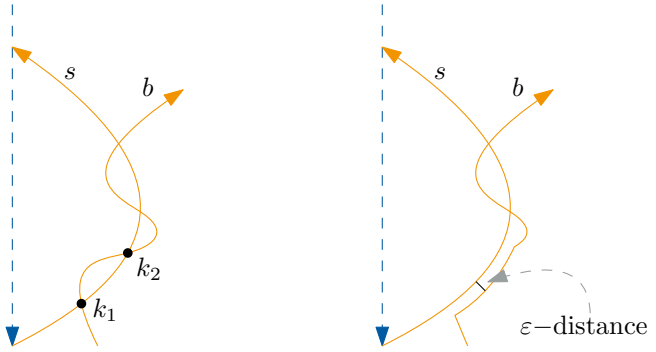
Claim 1. *W.l.o.g. we can assume $E_S = \emptyset$.*

Proof. Since Lemma 3.14 holds for $\mathcal{E}$ due to Lemma 3.15, we can infer that all self-loops of G are embedded into $\mathcal{E}$ such that they do not interfere which any other edges in $E_T \cup E_B$. □

Next, let D_{E_T} be a planar drawing of T according to $\mathcal{E}_{E_T}$. D_{E_T} exists because a tree is always planar and each embedding of it is planar. Let $P \subseteq E_B$ be a maximal subset of back edges such that there is a planar drawing $D_{E_T \cup P}$ of the graph $(V, E_T \cup P)$ according to $\mathcal{E}_{E_T \cup P}$. Now, if $P = E_B$ holds, then $D_{E_T \cup P} = D_{E_T \cup E_B}$ is equivalent to a planar drawing of G, so $\mathcal{E}$ is planar, done. Thus, we continue by assuming $P \subsetneq E_B$. Our goal is now to show that this assumption leads to a contradiction. The maximality of P implies that there exists $s \in E_B \setminus P$ such that s cannot be added to $D_{E_T \cup P}$ according to $\mathcal{E}_{E_T \cup P \cup \{s\}}$ without introducing an edge crossing. Now, choose s such that the depth of its target is minimal, i.e. $\text{depth}(\text{target}(s)) = \min_{e \in E_B \setminus P} \text{depth}(\text{target}(e))$. Given that the tree T has only one face, s can be added to $D_{E_T \cup P}$ according to $\mathcal{E}_{E_T \cup P \cup \{s\}}$ without s crossing a tree edge (but s crosses at least one back edge). Now, let $D_{E_T \cup P \cup \{s\}}$ be such a drawing. Furthermore, let $Q \subseteq P$ be the set of all edges in P that cross s in the drawing $D_{E_T \cup P \cup \{s\}}$.

Claim 2. *W.l.o.g. we can assume that every edge in Q crosses s exactly once.*

Proof. Let $b \in Q$ such that s and b cross at least twice in $D_{E_T \cup P \cup \{s\}}$, let B be the cycle in G formed by b and the path $\text{target}(b) \underset{T}{\rightarrow} \text{source}(b)$, let K_b be the ordered set of all crossings of s and b in $D_{E_T \cup P \cup \{s\}}$. We have $|K_b| \geq 2$. Since s does not cross any tree edges, K_b contains exactly all crossings of s and B. Therefore, observe that for two successive crossings $k_1, k_2 \in K_b$, w.l.o.g. s enters B with k_1 and exits B with k_2. This is illustrated in



(a) Drawing $D_{E_T \cup P \cup \{s\}}$ with the consecutive crossings k_1, k_2 of s and b.

(b) Drawing $D'_{E_T \cup P \cup \{s\}}$ without the previous crossings of s and b.

Figure 3.11.: Illustration of eliminating two consecutive crossings of back edges from Claim 2.

Figure 3.11a. We now can eliminate those two crossings by redrawing s such that s stops right before k_1, proceeds next to b at an ε-distance ($\varepsilon \in \mathbb{R}^+$) until k_2, and then continues as in $D_{E_T \cup P \cup \{s\}}$. From this, we get a new drawing $D'_{E_T \cup P \cup \{s\}}$ such that $|K'_b| = |K_b| - 2$, where K'_b is the ordered set of all crossings of s and b in $D'_{E_T \cup P \cup \{s\}}$. This is illustrated in Figure 3.11b. We now observe that if b crosses s an even number of times, we can eliminate all crossings, which leads to a contradiction of $b \in Q$. If b crosses s an odd number of times, we can eliminate all but one crossing. □

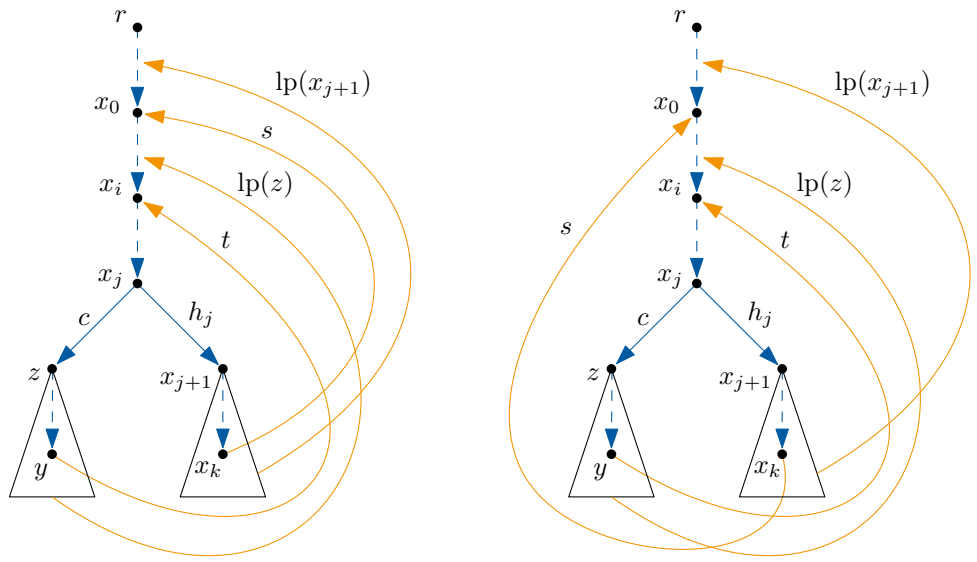
Now, choose $t \in Q$ such that the depth of is target is minimal, i.e. $\text{depth}(\text{target}(t)) = \min_{e \in Q} \text{depth}(\text{target}(e))$.

Claim 3. *W.l.o.g. we can assume* $\text{depth}(\text{target}(s)) \leq \text{depth}(\text{target}(t))$.

Proof. Let $P' := (P \setminus Q) \cup \{s\}$. By definition of Q, $P' \subseteq E_B$ is maximal w.r.t. adding edges such that a drawing $D_{E_T \cup P'}$ according to $\mathcal{E}_{E_T \cup P'}$ is planar. Let $D_{E_T \cup P' \cup \{t\}}$ be a drawing according to $\mathcal{E}_{E_T \cup P' \cup \{t\}}$ which extends $D_{E_T \cup P'}$. By definition of P', $D_{E_T \cup P' \cup \{t\}}$, t must cross an edge in $E_T \cup P'$ at least once, and w.l.o.g. t crosses an edge in P'. Let $Q' \subseteq P'$ be the set of all edges in P' that cross t in the drawing $D_{E_T \cup P' \cup \{t\}}$. Due to the above, we have $Q' \neq \emptyset$. Similar to Claim 2, we can w.l.o.g. assume that all edges in Q' cross t exactly once. We have now constructed the same situation as in our above proof, but where the roles of s and t are swapped. This justifies our claim, that w.l.o.g. we can assume $\text{depth}(\text{target}(s)) \leq \text{depth}(\text{target}(t))$. □

Let $x_0, \ldots, x_k \in V$ with $\text{source}(s) = x_k$ and $\text{target}(s) = x_0$ be the vertices of the tree edge path $p_s := x_0 \underset{T}{\rightarrow} x_k$. Define S as the cycle formed by p_s and s and let $p_t := \mathit{target}(t) \underset{T}{\rightarrow} \text{source}(t)$. Additionally, let $i, j \in \{0, \ldots, k\}$ such that $x_i = \mathit{target}(t)$ and such that x_j is the common vertex of the tree edge paths p_s and p_t with the highest depth, i.e. $x_j \in p_s \cap p_t$ and $\text{depth}(x_j) = \max_{u \in p_s \cap p_t} \text{depth}(u)$. x_j is well-defined, because if $p_s \cap p_t = \emptyset$ would hold, then this leads to a contradiction of $t \in Q$, i.e. of t crossing s once. W.l.o.g. we can assume $\varphi(t) > 0$, since the other case would be symmetric. Therefore, by Property 3.5 (ii), t is counter-clockwise.

In general, $i \leq j$ must hold because t is a back edge in a DFS tree. Furthermore, $k > 0$ holds, otherwise s would be a self-loop. Additionally, if $\text{source}(t) = x_j$ holds, then $i < j$, as



(a) Case where s is counter-clockwise.

(b) Case where s is clockwise.

Figure 3.12.: Illustration of the general situation, i.e. if $\text{source}(t) \neq x_j$ and $j < k$.

otherwise t would be a self loop. In particular, if $\text{source}(t) = x_j$, then $j > 0$. Additionally, we have the following definitions: In the case $\text{source}(t) \neq x_j$ holds, let $y := \text{source}(t)$, let z be the DFS child of x_j which lies on the tree edge path $x_j \underset{T}{\rightarrow} y$ and let $c \in E_T$ be the tree edge from x_j to z. Also, for every $l \in \{0, \ldots, k-1\}$, let $h_l \in E_T$ be the tree edge from x_l to x_{l+1}. An illustration of the current situation is given in Figure 3.12. In case $j < k$ holds, let $s^* := \text{lp}(x_{j+1})$. In case $y \neq x_j$ holds, let $t^* := \text{lp}(z)$. Now, we want to consider these low pointers and show their respective relations to s and t.

Claim 4. *Assume that $y \neq x_j$ holds if $q = t$ or that $j < k$ holds if $q = s$. Let $(q, q^*) \in \{(s, s^*), (t, t^*)\}$. Then w.l.o.g. we can assume $\varphi(q) = \varphi(q^*)$ and $\sigma(q) = \sigma(q^*)$.*

Proof. We get that q and q^* must be consecutive in L_{x_j} by Lemma 3.1. Thus, the definition of lp and the minimality of $\text{depth}(\text{target}(q))$ now imply $\text{depth}(\text{target}(q^*)) = \text{depth}(\text{target}(q))$. Therefore, we get $\text{target}(q^*) = \text{target}(q)$. Given that q and q^* are consecutive in L_{x_j}, they are now also consecutive in $\mathcal{E}(\text{target}(q))$, so w.l.o.g. we can assume that $\varphi(q) = \varphi(q^*)$ holds based on Lemma 3.12. Additionally, if q^* has the same orientation as q, then by Property 3.3 (v), we get $\sigma(q) = \sigma(q^*)$. If they have different orientations, then w.l.o.g. we can nevertheless assume $\sigma(q) = \sigma(q^*)$, because q and q^* are consecutive in L_{x_j}. □

Next, we reduce the case $j = k$ to the more general case $j < k$.

Claim 5. *W.l.o.g. we can assume $j < k$.*

Proof. Based on the definitions in Lemma 3.12, we see that for embedding either h_j or s at x_j, we have

$$\begin{pmatrix} \sigma(\text{lp}(\text{target}(h_j))) \\ -\varphi(\text{lp}(\text{target}(h_j))) \\ N(\text{target}(h_j)) \\ 0 \end{pmatrix} = \begin{pmatrix} \sigma(s) \\ -\varphi(s) \\ N(x_{j+1}) \\ 0 \end{pmatrix}$$

as ordering values for $j < k$ and

$$\begin{pmatrix} \sigma(s) \\ -\varphi(s) \\ 0 \\ 0 \end{pmatrix}$$

as ordering values for $j = k$. From Lemma 3.8, we get that there exist no $e \in B'_{x_j}$ such that

$$\begin{pmatrix} \sigma(s) \\ -\varphi(s) \end{pmatrix} = \begin{pmatrix} \sigma(e) \\ -\varphi(e) \end{pmatrix}$$

Therefore, the value $N(x_{j+1})$ is redundant, and we conclude that s for $j = k$ and h_j for $j < k$ are embedded in the same position in $\mathcal{E}(x_j)$. □

Next, we reduce the case $y = x_j$ to the more general case $y \neq x_j$.

Claim 6. *W.l.o.g. we can assume $y \neq x_j$.*

Proof. Based on the definitions in Lemma 3.12, we see that for embedding either c or t at x_j, we have

$$\begin{pmatrix} \sigma(\mathrm{lp}(\mathrm{target}(c))) \\ -\varphi(\mathrm{lp}(\mathrm{target}(c))) \\ N(\mathrm{target}(c)) \\ 0 \end{pmatrix} = \begin{pmatrix} \sigma(t) \\ -\varphi(t) \\ N(z) \\ 0 \end{pmatrix}$$

as ordering values for $y \neq x_j$ and

$$\begin{pmatrix} \sigma(t) \\ -\varphi(t) \\ 0 \\ 0 \end{pmatrix}$$

as ordering values for $y = x_j$. From Lemma 3.8, we get that there exist no $e \in B'_z$ such that

$$\begin{pmatrix} \sigma(t) \\ -\varphi(t) \end{pmatrix} = \begin{pmatrix} \sigma(e) \\ -\varphi(e) \end{pmatrix}$$

Therefore, the value $N(z)$ is redundant, and we conclude that t for $y = x_j$ and c for $y \neq x_j$ are embedded in the same position in $\mathcal{E}(x_j)$. □

The following claim will be used to simplify the further proof.

Claim 7. *Let $d \in V$, let $e, f \in B'_d$ with* $\mathrm{target}(e) = \mathrm{target}(f)$. *Then $\varphi(e) \neq \varphi(f)$.*

Proof. Simply follows from Properties 3.5 (i) to 3.5 (iii): W.l.o.g. let $\varphi(e) \leq \varphi(f)$. If e, f have different orientations, then $\varphi(e) < 0 < \varphi(f)$ by Properties 3.5 (i) and 3.5 (ii). If e, f have the same orientation, then $\varphi(e) < \varphi(f)$ by Property 3.5 (iii). □

Now, we eliminate the case $j = 0$.

Claim 8. *If $j = 0$, then we get a contradiction to Claim 7.*

Proof. Since $\varphi(t) > 0$, $(c, t, s) \triangleright \mathcal{E}(x_0)$ due to Lemma 3.12. Additionally, s and t cross once, so we also have $(c, h_0, t) \triangleright \mathcal{E}(x_0)$. From Lemma 3.12, we now get

$$\begin{pmatrix} \sigma(t) \\ -\varphi(t) \\ N(z) \\ 0 \end{pmatrix} = \begin{pmatrix} \sigma(t^*) \\ -\varphi(t^*) \\ N(z) \\ 0 \end{pmatrix} \leq \begin{pmatrix} \sigma(s^*) \\ -\varphi(s^*) \\ N(x_{j+1}) \\ 0 \end{pmatrix} \leq \begin{pmatrix} \sigma(t) \\ -\varphi(t) \\ 0 \\ 0 \end{pmatrix}$$

Therefore, we get $\sigma(t) = \sigma(s^*) = \sigma(s)$ and $\varphi(t) = \varphi(s^*) = \varphi(s)$. Property 3.3 (v) now implies $\text{target}(s) = \text{target}(t)$. With $\varphi(t) = \varphi(s)$, we see that Claim 7 is violated. □

The previous claims showed that we can now assume the following: $0 < j < k$, $y = \text{source}(t) \neq x_j$, $\sigma(s) = \sigma(s^*)$, $\sigma(t) = \sigma(t^*)$, $\varphi(s) = \varphi(s^*)$ and $\varphi(t) = \varphi(t^*)$. Recall that $s^* = \text{lp}(x_{j+1})$ and $t^* = \text{lp}(x_z)$. Now, let S be the cycle formed by s and $x_0 \underset{T}{\rightarrow} x_k$. Now, t either connects to $\text{source}(t) = y$ on the inside or on the outside of S. The two following claims deal with these two remaining cases.

Claim 9. *If t connects to $\text{source}(t) = y$ on the inside of S, we get a contradiction to Claim 7.*

Proof. If t connects to $\text{source}(t) = y$ on the inside of S, then t must connect to x_i on the outside of S, which is only possible for $\text{target}(t) = x_i = x_0$, e.g. $i = 0$, and $(h_0, s, t) \triangleright \mathcal{E}(x_0)$. Therefore, from Lemma 3.12 we get that $\varphi(s) \leq \varphi(t)$ holds. Additionally, for t to be connecting to y on the inside of S, we need $(h_j, c, h_{j-1}) \triangleright \mathcal{E}(x_j)$. Lemma 3.8 now yields

$$\begin{pmatrix} \sigma(s) \\ -\varphi(s) \end{pmatrix} = \begin{pmatrix} \sigma(s^*) \\ -\varphi(s^*) \end{pmatrix} \leq \begin{pmatrix} \sigma(t^*) \\ -\varphi(t^*) \end{pmatrix} = \begin{pmatrix} \sigma(t) \\ -\varphi(t) \end{pmatrix}$$

Since s, t have the same orientation, Property 3.3 (v) now implies $\sigma(s) = \sigma(t)$, thus we have $-\varphi(s) \leq -\varphi(t)$, hence $\varphi(s) \geq \varphi(t)$. With the above, we thus get $\varphi(s) = \varphi(t)$, which contradicts Claim 7. □

Claim 10. *If t connects to $\text{source}(t) = y$ on the outside of S, we get a contradiction to Claim 7.*

Proof. Considering the case that t connects to $\text{source}(t) = y$ on the outside of S, we have $(c, h_j, h_{j-1}) \triangleright \mathcal{E}(x_j)$ for s and t to be crossing once. Hence, from Lemma 3.8 we get

$$\begin{pmatrix} \sigma(t) \\ -\varphi(t) \end{pmatrix} = \begin{pmatrix} \sigma(t^*) \\ -\varphi(t^*) \end{pmatrix} \leq \begin{pmatrix} \sigma(s^*) \\ -\varphi(s^*) \end{pmatrix} = \begin{pmatrix} \sigma(s) \\ -\varphi(s) \end{pmatrix}$$

If $\sigma(t) < \sigma(s)$ holds, then $0 < \sigma(t) < \sigma(s)$, so Property 3.3 (ii) implies that s, t are both counter-clockwise. Now, from Property 3.3 (iv) we get $\text{depth}(\text{target}(t)) < \text{depth}(\text{target}(s))$, which is a contradiction. If $\sigma(t) = \sigma(s)$ holds, then $-\varphi(t) \leq -\varphi(s)$ must hold. From $\sigma(t) = \sigma(s)$, Property 3.3 (v) leads to s, t having the same orientation and $x_i = \text{target}(t) = \text{target}(s) = x_0$, i.e. $i = 0$. For s, t to be crossing once, $(h_0, t, s) \triangleright \mathcal{E}(x_0)$ holds, so from Lemma 3.12 we get

$$\begin{pmatrix} \sigma(\text{lp}(x_1)) \\ -\varphi(\text{lp}(x_1)) \\ N(x_1) \\ 0 \end{pmatrix} \leq \begin{pmatrix} \sigma(\text{lp}(x_1)) \\ -\varphi(\text{lp}(x_1)) \\ N(x_1) \\ \varphi(t) \end{pmatrix} \leq \begin{pmatrix} \sigma(\text{lp}(x_1)) \\ -\varphi(\text{lp}(x_1)) \\ N(x_1) \\ \varphi(s) \end{pmatrix}$$

This leads to $\varphi(t) \leq \varphi(s)$, thus $\varphi(t) = \varphi(s)$ with the above, which contradicts Claim 7. □

Due to these contradictions, our assumption of $P \subsetneq E_B$ was wrong, therefore $P = E_B$ must hold and $D_{E_T \cup P} = D_{E_T \cup E_B}$ is equivalent to a planar drawing of G, so $\mathcal{E}$ is planar.

3.3. Implementation

In this section, we describe the implementation of our embedding algorithm, which extends the Haeupler-Tarjan planarity test. During the planarity test, as described in described in Section 3.3.1, we compute additional information for the following embedding step. If the graph is found not to be planar, the algorithm exists and the additional information is discarded. Otherwise, the graph is planar and therefore, there exists a planar embedding of it. To find such a planar embedding, the embedding step is now run as described in Section 3.3.2, which returns a planar embedding.

3.3.1. Computing ordering values

In order to compute the orderings, we need suitable values for the functions N, ι φ, lp and σ.

3.3.1.1. Computing identifier

The injective function $N : V \to \mathbb{N}$, which serves as an identifier function for vertices, can simply be implemented as a bijective function $N : V \to \{1, \dots, |V|\}$ by numbering the vertices. Hence, the inverse N^{-1}, which returns the vertex for a given valid identifier, can also easily be computed in constant time. Similarly, we can implement the injective function $\iota : E_S \to \mathbb{Z}$ as a bijective numbering $\iota : E_S \to \{1, \dots, |E_S|\}$ of all self-loops in E_S. This can be implemented by simply using a global counter: When discovering a new self-loop e during the DFS, we set $\iota(e)$ to the value of the global counter and then increment the counter.

3.3.1.2. Computing relative ordering values

During the course of the planarity test, the relative orderings that φ aims to describe are encoded into the data structure T used for managing all permitted cyclic orders of outgoing back edges, which in our case is a PC-tree. In particular, when adding a new node v, a restriction on T is performed to make all back edges targeting v and the corresponding tree edge t consecutive, resulting in a new PC-tree T'. A valid relative ordering of all back edges that target v, which are represented by the full leaves of T', can now be obtained by traversing T'. This approach was also taken by Frey [Fre22]. Given some legal embedding of T', i.e. where the orders K_c or flipped(K_c) around C-nodes c are respected, we walk along the outer face of T' (which is the only face of T' since it is a tree) in both directions, starting at the full tree edge t, until we respectively discover an empty node. This results in two linear orders D_L, D_R of discovered leaves, where all back edges targeting v are either contained in D_L and D_R. Conceptually, we now see that reverse$(D_L) \cup (t) \cup D_R$ is the consecutive part of the cyclic order that is represented by T in the chosen embedding. For every leaf $l \in D_L \dot\cup D_R$, we define the distance of l to be the index (1-indexed) of the position of l in either D_L or D_R respectively. For every back edge e which is represented by a leaf $l \in D_L \dot\cup D_R$, $|\varphi(e)|$ is assigned the distance of l. Additionally, we assign $\text{sign}(\varphi(e)) = -1$ if $l \in D_L$ and $\text{sign}(\varphi(e)) = +1$ if $l \in D_R$. Altogether, we thus obtain a value for $\varphi(e) \in \mathbb{Z} \setminus \{0\}$. The pseudocode for the traversal and the φ-assignment is given in Algorithm 3.1. Note that in the implementation of PC-trees that we used, PC-trees are actually rooted, and in the planarity test, the root is the current tree edge that has been made consecutive along with all back edges that connect to the new node [FPR21]. Furthermore, we observe that the usage of PC-trees in fact guarantees that Lemma 3.1 is respected in our calculated embedding. This is because all back edges that originate in the same subtree are consecutive in all cyclic orders of the PC-tree. This property, which is the main property that actually guarantees the proper working of the planarity test, is illustrated by Figure 3.13.

Algorithm 3.1: PC-tree traversal with φ-assignment

```
1  Function Traverse(D')
2      finished ← TraverseRec(D'.GetRoot(), 1, 0)
3      if not finished then
4          TraverseRec(D'.GetRoot(), −1, 0)
5  Function TraverseRec(v, direction, nextValue)
6      if v.IsEmpty() then
7          return True
8      else if v.IsLeaf() then
9          φ(PCTreeLeafToBackEdge(v)) ← nextValue
10         return False
11     else
12         if direction = 1 then
13             begin ← v.RightmostChild
14             end ← v.LeftmostChild
15         else
16             begin ← v.LeftmostChild
17             end ← v.RightmostChild
18         forall children w of v from begin to end do
19             finished ← TraverseRec(w, direction, nextValue)
20             if finished then
21                 return True
22         return False
```

(a) Graph G.

(b) PC-tree T after processing vertices 5 and 4. Notice that all outgoing back edges from subtree(5) are consecutive in all orders in ord(T).

Figure 3.13.: Example of respected consecutivity in PC-trees.

Now, the problem remains that the choices of the embedding at C-nodes c, where either K_c or flipped(K_c) are taken to calculate φ as described above, can vary during the course of the planarity test. This is due to fact that an implementation PC-trees may only store one legal embedding of the tree, which is used for traversing the tree as above, but then the orderings at C-nodes may be individually flipped during the execution of a restriction on the PC-tree [FPR21]. As a result, the signs of the φ-values may be incompatible with the signs of previously computed φ-vales. This problem has already been identified by Chiba et al. in their embedding algorithm [CNAO85]. To solve this, we take a similar approach as Chiba et al. [CNAO85] and Frey [Fre22]: We use *indicators* to track whether an ordering of a C-node has been flipped since the creation of the C-node. When a C-node c is created, we also create an indicator i_c. Conceptually, i_c is an "arrow" in the PC-tree with its base base(i_c) being a neighboring node of c and with its target target(i_c) being another neighboring node of c such that base(i_c) and target(i_c) are consecutive in K_c. If the PC-nodes base(i_c) or target(i_c) are deleted or merged into c, we need to keep i_c "valid", i.e. we may need to reassign base and target. Indicators are collected during the tree walk for assigning φ-values. These collected indicators are now removed from the PC-tree, and we store the dependence to the newly created indicator for the new C-node. In the end, these dependencies allow us to traverse the indicator dependency tree using a DFS in order to correct all signs of φ-values.

3.3.1.3. Computing low pointers

For $a \in V$, we have defined a low pointer as follows:

$$\mathrm{lp}(a) \in B_a \text{ such that } \mathrm{depth}(\mathrm{target}(\mathrm{lp}(a))) = \min_{e \in B_a} \mathrm{depth}(\mathrm{target}(e))$$

Computing the low pointer for every vertex can be done during the DFS of the planarity test. When advancing along a back edge $e \in E_B$ with $v := \mathrm{source}(e)$ and $w := \mathrm{target}(e)$ and if w has a smaller depth than the current low pointer of v, assign e as the new low pointer of v. If the low pointer of v does not exist, the assignment also happens. When retreating along a tree edge $e \in E_T$ with $v := \mathrm{source}(e)$ and $w := \mathrm{target}(e)$ and if w has a low pointer with a smaller depth than the current low pointer of v, assign the low pointer of w as the new low pointer of v. If the low pointer of v does not exist, but the low pointer of w does, the assignment also occurs. The pseudocode of this procedure is given in Algorithm 3.2, which is based on the recursive DFS in Algorithm 2.1. It is obvious that using this approach, low pointers can be computed with constant time overhead regarding the DFS.

3.3.1.4. Computing depth-based ordering values

Having computed assignments for φ and lp, the following lemma shows a viable formula for σ.

Lemma 3.17. *Let $G = (V, E)$ be a connected planar undirected multigraph, let there be a DFS on G. We define σ as follows for all $e \in E_B$:*

$$\sigma(e) = (\mathrm{depth}(\mathrm{target}(e)) + 1) \cdot \mathrm{sign}(\varphi(e))$$

Then the Properties 3.3 (i) to 3.3 (v) of σ are satisfied.

Proof. Notice that $\mathrm{sign}(\sigma) = \mathrm{sign}(\varphi)$ because $\mathrm{depth} + 1 > 0$. Therefore, Properties 3.3 (i) and 3.3 (ii) of σ holds due to Properties 3.5 (i) and 3.5 (ii) of φ. Let $v \in V(G)$ and $b, c \in B'_v$. If b, c are clockwise and $\mathrm{depth}(\mathrm{target}(b)) > \mathrm{depth}(\mathrm{target}(c))$, then $\varphi(b) < 0$ and $\varphi(c) < 0$

Algorithm 3.2: Depth-first search with low pointers

```
1  Function DFS (G = (V, E))
2      forall v ∈ V do
3          v.visited ← false
4      forall v ∈ V do
5          if not v.visited then
6              DFS-CONNECTED(G, v, null)
7  Function DFS-CONNECTED(G = (V, E), v, previousTreeEdge)
8      v.visited ← true
9      forall e ∈ E with v ∈ e do
10         if e = {v, v} then
               // e is a self-loop
11             E_S ← E_S ∪ {e}
12         else
13             let w ∈ e such that w ≠ v
14             if not w.visited then
                   // e is a tree edge
15                 DFS-CONNECTED(G, w, e)
16                 if lp(w) ≠ null and (lp(v) = null or depth(target(lp(w))) <
                    depth(target(lp(v)))) then
17                     lp(v) ← lp(w)
18             else if e ≠ previousTreeEdge then
                   // e is a back edge
19                 if lp(v) = null or depth(w) < depth(target(lp(v))) then
20                     lp(v) ← e
```

because of Property 3.5 (i) of φ, and we have $\sigma(b) = (\text{depth}(\text{target}(b)) + 1) \cdot \text{sign}(\varphi(b)) = -(\text{depth}(\text{target}(b)) + 1) < -(\text{depth}(\text{target}(c)) + 1) = (\text{depth}(\text{target}(c)) + 1) \cdot \text{sign}(\varphi(c)) = \sigma(c)$, thus Property 3.3 (iii) holds. Analogously, we get Property 3.3 (iv). Concerning Property 3.3 (v), if $\sigma(b) = \sigma(c)$, we get $\text{sign}(\varphi(b)) = \text{sign}(\varphi(c))$ and $\text{depth}(\text{target}(b)) = \text{depth}(\text{target}(c))$ by the formula of σ, therefore b, c have the same orientation by Properties 3.5 (i) and 3.5 (ii) of φ and $\text{target}(b) = \text{target}(c)$ due to $b, c \in B'_v$. On the other hand, if b, c have the same orientation and $\text{target}(b) = \text{target}(c)$ holds, then we get $\text{sign}(\varphi(b)) = \text{sign}(\varphi(c))$ by Properties 3.5 (i) and 3.5 (ii) of φ and $\text{depth}(\text{target}(b)) = \text{depth}(\text{target}(c))$. □

Using this formula for σ, we can also show that the function is linearly bounded by the input size.

Lemma 3.18. *Let σ be defined as in Lemma 3.17. Then $-(|V| + 1) < \sigma < |V| + 1$.*

Proof. Let $e \in E_B$. Then we have $\sigma(e) = (\text{depth}(\text{target}(e))+1) \cdot \text{sign}(\varphi(e)) \leq \text{depth}(\text{target}(e))+1 \leq |V| - 1 + 1 = |V| < |V| + 1$ and $\sigma(e) = (\text{depth}(\text{target}(e)) + 1) \cdot \text{sign}(\varphi(e)) \geq -(\text{depth}(\text{target}(e)) + 1) \geq -(|V| - 1 + 1) = -|V| > -(|V| + 1)$ for all $e \in E_B$. □

3.3.2. Embedding step

After the execution of the DFS and therefore after the planarity test, which we have extended by computing the necessary values as described above, we now compute a planar embedding $\mathcal{E}$ of G by computing $\mathcal{E}$ such that it satisfies the Lemmas 3.1 and 3.15 as in described in Theorem 3.16. This is done by sorting the edges according to their orderings at every vertex. To achieve a linear running time in total, we hereby use radix sort and sort all the edges at once. In particular, we construct two vectors for every edge $e \in E_T \cup E_B \cup E_S$. The vector $M_O(e)$ contains the ordering values for the outgoing edge e at source(e) and the vector $M_I(e)$ contains the ordering values for the incoming edge e at target(e), see Lemma 3.15. These vectors are put into an array A. For $e \in E_T \cup E_B \cup E_S$ with $v := \text{source}(e)$ and $w := \text{target}(e)$, we define the vectors as follows. Note that if e is a back edge, then $x \in V$ is the target of the corresponding tree edge.

$$M_O(e) := \begin{cases} (N(v), \sigma(\text{lp}(w)), -\varphi(\text{lp}(w)), N(w), 0) & \text{if } e \in E_T \wedge \text{lp}(w) \neq \bot \\ (N(v), 0, 0, 0, 0) & \text{if } e \in E_T \wedge \text{lp}(w) = \bot \\ (N(v), \sigma(e), -\varphi(e), 0, 0) & \text{if } e \in E_B \\ (N(v), -(|V|+1), \iota(e), 0, 0) & \text{if } e \in E_S \end{cases}$$

$$M_I(e) := \begin{cases} (N(w), |V|+1, 0, 0, 0) & \text{if } e \in E_T \\ (N(w), \sigma(\text{lp}(x)), -\varphi(\text{lp}(x)), N(x), \varphi(e)) & \text{if } e \in E_B \\ (N(v), -(|V|+1), \iota(e), 0, 0) & \text{if } e \in E_S \end{cases}$$

We construct $M_I(e)$ for a tree edge e such that in the final sorted order, it is placed after all other vectors of edges that connect to target(e), as seen in Lemma 3.13. For this, the value $|V|+1$ is used instead of ∞, because we have $\sigma(f) < |V|+1$ for all $f \in E_B$ as shown in Lemma 3.18. Similarly, for a self-loop e, we need to place it before any tree or back edges in the final sorted order as seen in Lemma 3.15. We use the value $-(|V|+1)$ instead of $-\infty$ in the ordering tuples because we have $\sigma(f) > -(|V|+1)$ for all $f \in E_B$ as shown in Lemma 3.18. The first component of every vector is the identifier of the vertex that e connects to. By adding this component, we can sort all vectors at once, which yields a grouping of connection points. Therefore, by using radix sort and counting sort on A, we can achieve a total linear running time. Note that vectors are sorted lexicographically. Having sorted the vectors to obtain an array A', we can obtain the resulting planar embedding $\mathcal{E}$ of G by iterating through A' and assigning the cyclic orders as we encounter vectors of corresponding edges. The algorithm is illustrated in Algorithm 3.3.

Algorithm 3.3: Embedding step

```
1  Function Embed(G with DFS edges E_T ∪ E_B ∪ E_S)
2      A ← new array
3      forall e ∈ E_T ∪ E_B ∪ E_S do
4          A.append(M_O(e))
5          A.append(M_I(e))
6      RadixSort(A)
7      E ← empty embedding
8      forall t = (id, _, _, _, _) ∈ A in order do
9          v ← N^{-1}(id)
10         E(v).append(t.GetCorrespondingEdge())
11     return E
```

3.3.3. Running time

In this section, we want to show that our embedding algorithm indeed runs in linear time.

Theorem 3.19. *Given an input graph $G = (V, E)$, our embedding algorithm runs in linear time $\mathcal{O}(|V| + |E|)$.*

The Haeupler-Tarjan planarity test runs in linear time [HT08]. Thus, we only have to investigate our extension of the algorithm for computing a planar embedding. It is clear that we can compute the identifier functions N and ι in linear time $\mathcal{O}(|V|)$ and $\mathcal{O}(|E_S|) \subseteq \mathcal{O}(|E|)$ respectively. We now discuss the runtime of computing and correcting φ-values.

Claim 1. *Given an input graph $G = (V, E)$, φ can be correctly computed for all back edges E_B in time $\mathcal{O}(|V| + |E|)$.*

Proof. The tree walk Traverse in Algorithm 3.1 for assigning φ-values only introduces constant overhead for every non-empty PC-tree node. Since assigning full and partial labels to PC-tree nodes takes linear time in total regarding the planarity test [FPR21], we get that this part of assigning φ-values also only takes linear time in total during the planarity test. Considering the sign correction, the described indicators from Section 3.3.1.2 can be implemented such that for every PC-tree node n, there exists at most one indicator having n as a base and there exists at most one indicator having n as a target. This is done by collecting indicators greedily whenever an indicator is moved such that bases or targets would be shared. Using this approach, a constant overhead for all C-node operations can be achieved and therefore, keeping indicators "valid" and collecting indicators during φ-value assignment only introduces linear overhead in total. Additionally, the correction of φ-values with the traversal of the indicator dependency tree has a runtime of $\mathcal{O}(\text{number of indicators} + \text{number of } \varphi\text{-values}) \subseteq \mathcal{O}(|V| + |E_B|) \subseteq \mathcal{O}(|V| + |E|)$, so this also introduces an additional linear overhead in total. We therefore see that computing correct φ-values only adds a linear running time overhead in total. □

Furthermore, we see in Section 3.3.1.3 and Algorithm 3.2 that low pointers can be computed with constant overhead over the DFS. Additionally, using the formula for σ given in Section 3.3.1, we can also compute this function in constant time after all φ-values have been correctly assigned. Note that depth can be computed with constant overhead over the DFS, see Section 2.1.3. It remains that we investigate the running time for the embedding step, which occurs after the planarity test and the φ-value correction.

Claim 2. *Given an input graph $G = (V, E)$, the algorithm Embed from Algorithm 3.3 runs in time $\mathcal{O}(|V| + |E|)$.*

Proof. The first for-loop in Embed trivially runs in time $\Theta(|E|)$. Additionally, after this loop, A contains $2 \cdot |E|$ elements. The element count in A does not change with the call RadixSort(A). Therefore, the second for-loop has $2 \cdot |E|$ iterations. Since GetCorrespondingEdge and N^{-1} can be computed in $\mathcal{O}(1)$, we get a running time of $\Theta(|E|)$ for the second for-loop. It is left to consider the running time for the call RadixSort(A). For this, we examine the value ranges for all components of the vectors M_O and M_I. Note that those vectors only contain integers. In the following, let $e \in E_T \cup E_B \cup E_S$, let $M \in \{M_O, M_I\}$ and let $M(e) = (m_1, m_2, m_3, m_4, m_5)$. Concerning the first component, we see that $m_1 = N(\cdot) \in \{1, \ldots, |V|\}$ as described in Section 3.3.1.1, therefore $1 \leq m_1 \leq |V|$. Similarly, for the forth component we either have $m_4 = 0$

or $m_4 = N(\cdot) \in \{1, \ldots, |V|\}$, so $0 \leq m_4 \leq |V|$. For the second component, we have $m_2 \in \{\sigma(\cdot), 0, |V|+1, -(|V|+1)\}$. Because of $-(|V|+1) < \sigma < |V|+1$ as shown in Lemma 3.18, we get $-(|V|+1) \leq m_4 \leq |V|+1$. Regarding the third and fifth components, we have $m_3, m_4 \in \varphi(E_B) \cup -\varphi(E_B) \cup \iota(E_S)$. From the algorithm in Section 3.3.1.2, we infer that $|\varphi| \leq |E_B|$ must hold because there can at most occur $|E_B|$ back edges when traversing the PC-tree. Moreover, Section 3.3.1.1 yields $\iota(E_S) = \{1, \ldots, |E_S|\}$, so we have $\iota \leq |E_S|$. Therefore, we get $-|E| \leq -(|E_B| + |E_S|) \leq m_i \leq |E_B| + |E_S| \leq |E|$ for $i \in \{3, 4\}$. In total, we now have $|m_i| \leq |V| + |E| + 1$ for all $i \in \{1, \ldots, 5\}$. The running time of radix sort is in general $\Theta(d \cdot (n + k))$, where d is the number of values in the tuple, n is the number of tuples to be sorted and k is the number of possible values for each component in the tuple [CLRS22]. In our case, we have $d = 5$, $n = |A| = 2 \cdot |E|$ and $k \leq 2 \cdot (|V| + |E| + 1) + 1 = 2 \cdot |V| + 2 \cdot |E| + 3$. Therefore, the call RadixSort(A) runs in time $\mathcal{O}(5 \cdot (2 \cdot |E| + 2 \cdot |V| + 2 \cdot |E| + 3)) = \mathcal{O}(|V| + |E|)$. In total, for Embed we get the running time $\mathcal{O}(|E| + |V| + |E| + |E|) = \mathcal{O}(|V| + |E|)$. □

In total, we see that our embedding algorithm terminates in time $\mathcal{O}(|V| + |E|)$.

4. SPQR-tree construction

In this chapter, we develop an algorithm approach that, based on the Haeupler-Tarjan planarity test with PC-trees [HT08], computes the SPQR-tree for the given input graph. Specifically, we want the algorithm to solve the following problem: Given a biconnected undirected multigraph $G = (V, E)$, determine whether G is planar. Additionally, if this is the case, output the skeletons of the SPQR-tree of G. To develop such an algorithm, we focus on the duality of PC-trees and SPQR-trees. In Section 4.1, we investigate some general observations regarding the relation between planarity test and SPQR-trees. Then, we individually describe how bonds, rigids and polygons of the SPQR-tree can be derived from the planarity test in Section 4.2, 4.3 and 4.4 respectively.

4.1. General observations

First, we investigate the placement of separation pairs in a DFS-tree.

Lemma 4.1. *Let $G = (V, E)$ be a biconnected undirected multigraph, let $v, w \in V$ be a separation pair in G. Then for every DFS of G, v is an ancestor of w or w is an ancestor of v.*

Proof. Assume that the statement is wrong. Then there exists $a \in V$ with DFS children x, y such that $v \in \text{subtree}(x)$ and $w \in \text{subtree}(y)$. Let $a_1, \ldots, a_k \in V$ be the vertices on the DFS tree edge path such that a_1 is the DFS root and $a_k = a$. Let $S := \{a_1, \ldots, a_k\}$. Since G is biconnected, G contains no cut-vertex, thus $G - v$ and $G - w$ are connected respectively. Let $b \in \text{subtree}(v)$ and $d \in \text{subtree}(w)$. There exists a path $p_b := b \xrightarrow[G-v]{} x$ such that $w \notin p_b$, because v and w are separated in $G - S$. Analogously, there exists a path $q_d := d \xrightarrow[G-w]{} y$ with $v \notin q_d$. Therefore, by concatenating p_b, the edges $\{x, a\}, \{a, y\}$ and q_d, there exists a path from b to d in $G - v - w$. Additionally, we obviously have that $G - \text{subtree}(v) - \text{subtree}(w)$ is connected. Altogether, we get that $G - v - w$ is connected, a contradiction to v, w being a separation pair of G. □

In addition, we observe a correlation between the graph and the PC-trees during the planarity test with regard to existent paths.

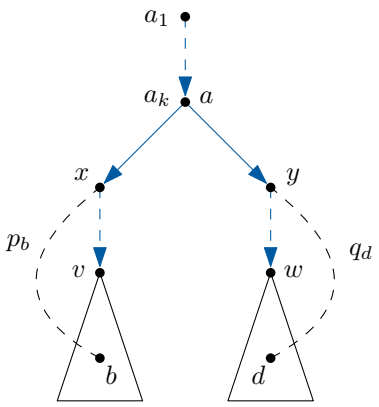


Figure 4.1.: Illustration of the proof of Lemma 4.1.

Lemma 4.2. *Let T be a PC-tree during the planarity test of a biconnected undirected multigraph G, let $v, w \in V(G)$ be labels of the nodes $x, y \in V(T)$, let $u \in V(G) \setminus \{v, w\}$. Then there exists a path from v to w in G that does not contain u if and only if there exists a path in T from x to y that does not contain a node with label u or if there exists paths from x and y to a leaf of T respectively that both contain no nodes with label u.*

Proof. Assume there exists a path p from v to w in G that does not contain u. If u has not already been merged into T, then T contains a path from x to y that does not contain the label u. This path follows the labels as in p, aside from possible contractions that occurred in the PC-trees. If u has already been merged into T and if the path in T from x to y contains a node with label u, then x and y have respective paths to leaves of T that do not contain the label u due to the biconnectivity of G. The other direction of the equivalency is analogous. □

4.2. Deriving bonds

In SPQR-trees, bonds are represented by P-nodes. Bonds consist of exactly two vertices, which are connected by at least three edges [FR23]. Let D be the SPQR-tree of G, let $B \in V(D)$ be a P-node. Let $v, w \in V$ be the vertices in G that form the bond that is represented by B. By Lemma 4.1, we get that w.l.o.g. v is an ancestor of w in every DFS tree of G. Let c be the child of v such that $w \in \text{subtree}(c)$. During the planarity test, when retreating along the tree edge (v, c) (retreating from c to v), we are now able to discover the bond v, w by using the following rule.

Rule 1 (Bonds)**.** *Let $G = (V, E)$ be a biconnected undirected multigraph, let T' be a PC-tree during the planarity test on G that results from a restriction on the edges connecting to the new node $v \in V$, let p be a P-node in T' with label $w \in V$, $\deg(p) \geq 3$ and with at least 2 full neighbors. Then v, w is a bond in the SPQR-tree of the graph.*

To justify this rule, we first show that the P-node in Rule 1 actually has at least $\deg(p) - 1$ full neighbors.

Lemma 4.3. *Let $G = (V, E)$ be a biconnected undirected multigraph, let T' be a PC-tree during the planarity test on G that results from a restriction on the edges connecting to a new node, let p be a P-node in T' with $\deg(p) \geq 3$, let Q be the set of all nodes that is p is adjacent to, let F be the set of all full nodes in T'. Then $|Q \cap F| \geq 2 \iff |Q \cap F| \geq \deg(p) - 1$.*

Proof. The direction $|Q \cap F| \geq 2 \impliedby |Q \cap F| \geq \deg(p) - 1$ is trivial. Assume that $|Q \cap F| \geq 2$ holds. Note that $|Q| = \deg(p) \geq 3$. If $|Q| = 3$, then we have $|Q \cap F| \geq 2 = |Q| - 1 = \deg(p) - 1$. If $|Q| \geq 4$, then there exist $a, b \in Q \cap F$ with $a \neq b$ due to $|Q \cap F| \geq 2$. We now assume that $|Q \cap F| \leq \deg(p) - 2$ holds. Therefore, there exist $c, d \in Q \setminus F$ with $c \neq d$. For $q \in Q$, let subtree(q) be the subtree of T' that is obtained by cutting T' at the edge $\{q, p\}$ and that contains q. We now have that a, b are full and c, d are not full, so there exist full leaves $a' \in \text{subtree}(a), b' \in \text{subtree}(b)$ and empty leaves $c' \in \text{subtree}(c), d' \in \text{subtree}(d)$. By embedding T' using an embedding $\mathcal{E}$ with $(a, c, b, d) \triangleright \mathcal{E}(p)$, we now get that there exists a possible cyclic ordering $C \in \text{ord}(T')$ with $(a', c', b', d') \triangleright C$, so the full leaves are not consecutive in C, which contradicts the restriction on the PC-tree. □

Now, we show that in Rule 1, the vertices of the constructed bond are a separation pair in the graph, and we show that a bridge of the bond corresponds to a neighboring subtree of the selected P-node, while we define a subtree of a neighbor a of a node p in a PC-tree T as the connected component that contains a in $T - p$.

Lemma 4.4. *Let $G = (V, E)$ be a biconnected undirected multigraph, let T' be a PC-tree during the planarity test on G that results from a restriction on the edges connecting to the new node $v \in V$, let p be a P-node in T' with label $w \in V$,* $\deg(p) \geq 3$ *and with at least* 2 *full neighbors. Then v, w is a separation pair in G and a neighboring subtree of p corresponds to a bridge of the bond formed by v, w.*

Proof. We need to show that the bridges induced by v, w are not connected. Since $\deg(p) \geq 3$, there exist neighbors a, b of p in T' with $a \neq b$. Let $x, y \in V$ be labels in the subtrees of a and b respectively, while the subtree of a neighbor d of p is defined as the connected component of $T' - p$ that contains d. Similar to the proof of Lemma 4.2, we now get that there exist no paths from x to y in G that neither contain v nor w, because the subtrees of a and b either only contain full leaves that connect to v or only contain empty leaves that do not connect to v. Given that a subtree of a neighbor of p is connected in $T' - w$, we see that the corresponding subgraph in $G - v - w$ is also connected. Altogether, we get that a neighbor of p corresponds to a bridge □

4.3. Deriving rigids

In SPQR-trees, rigids, which are simple triconnected graphs, are represented by R-nodes [FR23]. During the planarity test, they are represented by C-nodes in the PC-tree, as we make clear in the following. To track the information which vertices of the graph are part of the current triconnected component that is represented by a C-node, every C-node c stores a subgraph R_c of G with internal edges and with boundary edges to the neighbors of c in the PC-tree. Given a PC-tree T before a restriction during the planarity test, let T' be the PC-tree that results from T after the next restriction regarding a new node n, let T'' the PC-tree that results from deleting all full nodes and appending the remaining tree to the P-node of n. Additionally, let t be the terminal path in T for the restriction regarding n. Note that t partitions the PC-tree into a *full region*, an *empty region* and t itself. W.l.o.g. we assume that t contains no nodes with degree 2. Let $C_t \dot\cup P_t = V(t)$ be the partition of edges in the terminal path t into its C-nodes C_t and its P-nodes P_t respectively. During one step of the planarity test from T to T'', there can occur three different events: *Rigid construction*, *Rigid expansion*, and *Rigid finalization*. Construction of a C-node $a \in V(T'')$ occurs if and only if $C_t = \emptyset$ and a is created during the restriction operation. Expansion of a C-node $a \in V(T'')$ occurs if and only if $C_t \neq \emptyset$ and a is created during the restriction operation. Finalization occurs if and only if a C-node $a \in V(T')$ is deleted during the merge operation from T' to T'', i.e. if and only if a is full.

Rule 2 (Rigid construction)**.** *If we have $C_t = \emptyset$, let $a \in V(T'')$ be the C-node that was created by the restriction operation. Then we define the subgraph R_a stored in a as follows: As vertices, R_a contains all labels of the P-nodes in P_t and the newly merged vertex n. As internal edges, R_a contains the edges of t with their endpoints being mapped to their labels and edges to n that represent the bridges to this new vertex. As boundary edges, R_a contains the edges that connect the labels of the P-nodes in P_t to their corresponding PC-nodes.*

Rule 3 (Rigid expansion)**.** *If we have $C_t \neq \emptyset$, let $a \in V(T'')$ be the C-node that was created by the restriction operation. Then we define the subgraph R_a stored in a as follows: As vertices, R_a contains the newly merged vertex n, as well as all labels of the nodes in the terminal path t, where equal labels are merged. As internal edges, R_a contains edges from the terminal path t, edges from all C-nodes in C_t and edges to n that represent the bridges to this new vertex. As boundary edges, R_a contains the edges that connect the labels of the nodes in the terminal path and n to their corresponding PC-nodes.*

Rule 4 (Rigid finalization)**.** *If we have that a C-node $a \in V(T')$ is full. Then a is deleted during the merging procedure for the new node n, and we get a new R-node for the SPQR-tree of G with $R_a + n$ as skeleton with the internal edges of $R_a + n$ as its edges, while $R_a + n$ has vertices $V(R_a) \cup \{n\}$ and edges the internal edges of R_a as well as edges that represent the bridges to n.*

We now show that for every PC-tree T_i during the planarity test and every C-node $c \in V(T_i)$, R_c is triconnected in G. For this, we need to make the following assumption, which seems to be true, but we were not able to proof it.

Assumption 1. *Let c be a C-node in an PC-tree during the planarity test, let p be a path in G between two vertices in $V(R_c)$, let q be a path in G between two vertices in $V(G) \setminus V(R_c)$. Then p and q are independent.*

Using this assumption, we can now continue with our analysis.

Lemma 4.5. *Let T be a PC-tree before a restriction during the planarity test, let $c \in V(T)$ be a C-node in T, let R_c be the subgraph of G that is represented by c. Then $V(R_c)$ is triconnected in G.*

Proof. We perform an induction over the steps $i \in \{1, \dots, |E_T| + 1\}$ of the planarity test, where each step corresponds to a state of the PC-trees between the retreat operations at tree edges. As the base case, we have $i = 1$. This is before the first retreat operation, thus no restrictions on PC-trees have taken place, so there exist no C-nodes. Therefore, the statement trivially holds for all C-nodes in this step. In the induction step, we obtained PC-tree T_{i+1} from T_i by retreating on a tree edge with source n. Let t be the terminal path for the restriction from T_i to T_i'. Let $v_1, \dots, v_m \in V(t)$ be the nodes of the terminal path. t now partitions T_i into a full region, an empty region and t itself. Let d be a C-node in $T_i'' = T_{i+1}$. There are now three different cases how d is generated. If there is a C-node c in T_i in the empty region and $c = d$, then c and R_c remained unchanged from T_i to $T_i'' = T_{i+1}$. Therefore, by the induction hypothesis, the property still holds for d.

If d was generated by the restriction and t only contains P-nodes, then Rule 2 was applied. We now have that for every node $v \in V(t)$, $\deg(v) \geq 3$ holds in T_i' after deletions of degree-1-nodes, contractions of degree-2-P-nodes and merges of C-nodes into c have taken place. Given a node $p \in V(t)$, let $P_F(v)$ be a node in the full region that is connected to p in T_i, if such a node exists, and let $P_E(v)$ be a node in the empty region that is

connected to p in T_i, if such a node exists. Now, $P_F(v_1), P_F(v_m), P_E(v_1), P_E(v_m)$ all exist because otherwise, we would get a contradiction to the construction of the terminal path. Additionally, for all nodes v_i with $i \in \{2, \dots, k-1\}$, there exists at least $P_E(t)$ or $P_F(t)$ in T_i, because t contains no degree-2-nodes. Let $a, b \in \{1, \dots, m\}$ with $a < b$. We now need to show that there exist three independent paths between v_a and v_b. As a first path p_1, we simply take the subpath in t between v_a and v_b. Next, w.l.o.g. assume that v_a is connected to $P_E(v_a)$, the other case is symmetric. If v_b is connected to $P_E(v_b)$, then take p_2 as the path $v_a \to P_E(v_a) \to \dots \to P_E(v_b) \to v_b$ and p_3 as the path $v_a \to v_1 \to P_F(v_1) \to \dots \to n \to \dots \to P_F(v_m) \to v_m \to v_b$. These paths are illustrated in Figure 4.2a. If v_b is connected to $P_F(v_b)$, then take p_2 as the path $v_a \to P_E(v_a) \to \dots \to P_E(v_m) \to v_m \to v_b$ and p_3 as the path $v_a \to v_1 \to P_F(v_1) \to \dots \to n \to \dots \to P_F(v_b) \to v_b$. These paths are illustrated in Figure 4.2b. Now, we see that the paths p_1, p_2, p_3 are independent. Additionally, given $i \in \{1, \dots, m\}$, we have to show that there exist three independent paths between v_i and m in G. We have constructed a new C-node during this restriction, so we have at least two full leaves in the T'. Therefore, we can easily find these three independent paths by going through the full region with two paths and going through the empty region with one path, which is possible since G is biconnected.

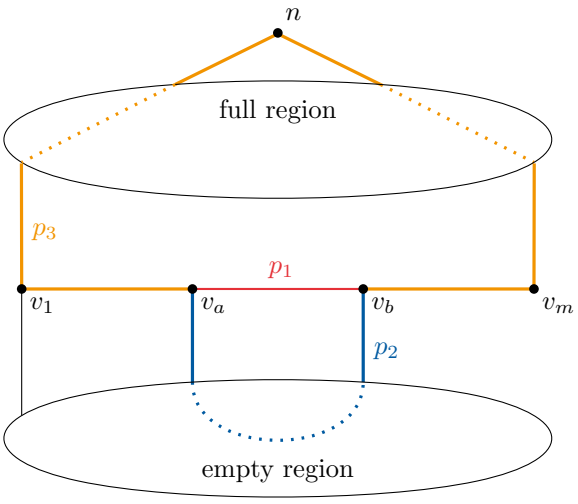
If d was generated by the restriction and t contains P-nodes and C-nodes, then Rule 3 was applied. We have that for every node $v \in V(t)$, $\deg(v) \geq 3$ holds, analogously to the base case. Observe that now, all C-nodes in C_t are merged into d. Let $v, w \in V(R_d)$. We now need to show that there exist three independent paths between v and w in G. By the induction hypothesis, we see that this holds for $v, w \in V(R_c)$ for all $c \in C_t$. Additionally, analogously to the induction base, we see that this holds for v, w coming from P-nodes in t. Note that here, by Assumption 1, we can simply walk through C-nodes. Therefore, we only have to consider two remaining cases, v, w coming from different C-nodes in C_t and w.l.o.g. v coming from a C-node and w coming from a P-node. First, consider $v \in V(R_a), w \in V(R_b)$ with $a, b \in C_t$ and $a \neq b$. Let $(c, u) \in \{(a, v), (b, w)\}$. Since $\deg(c) \geq 3$, define the paths p_1, p_2, p_3 analogously to in the previous case for Rule 2. p_1, p_2, p_3 now connect to $x, y, z \in V(R_c)$ respectively. It remains to show that there exist three independent paths q_1, q_2, q_3 from u to x, y and z in G respectively. Then, respectively concatenating those paths, i.e. $p_1 \cup q_1, p_2 \cup q_2, p_3 \cup q_3$, yields the required independent paths. Now, let $h \notin V(R_c)$ be a new vertex, and add h to R_c to obtain R'_c by connecting it x, y and z, i.e. $R'_c := (V(R_c) \dot\cup \{h\}, E(R_c) \dot\cup \{\{x, h\}, \{y, h\}, \{z, h\}\})$. Then for all $s, t \in V(R'_c)$, there exist independent paths between s and t. In particular, let q'_1, q'_2, q'_3 be three independent paths between u and h. Then w.l.o.g. $x \in q'_1$, $y \in q'_2$ and $z \in q'_3$ must hold, so q_1, q_2, q_3 with $q_i := q'_i - h$ for $i \in \{1, 2, 3\}$ are the required independent paths from u to x, y and z. Note that by Assumption 1, the paths q_1, q_2, q_3 inside the C-nodes do not share vertices of G with the paths p_1, p_2, p_3 outside the C-nodes. Analogously to the case with Rule 3, we also see that the new node n is also triconnected w.r.t. the other vertices in the new C-node. □

It now remains to show that when applying Rule 4, we obtain a maximally triconnected component of G as a rigid. We here also require the following assumption:

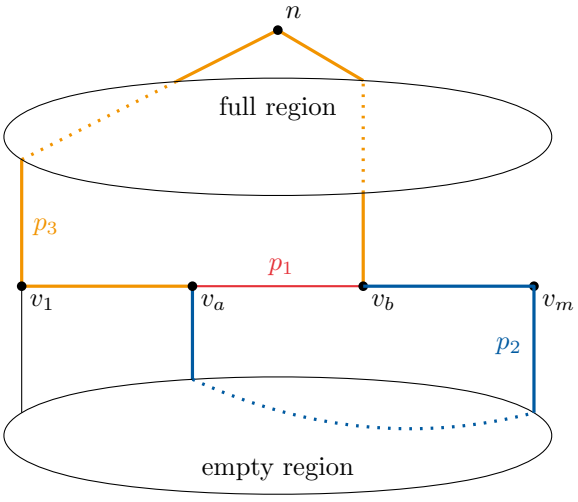
Assumption 2. *When Rule 4 is applied, then the resulting rigid is maximal regarding all vertices that have already been merged into the PC-tree in previous steps.*

Using this assumption, we can now in part proof the maximality of the generated rigids.

Lemma 4.6. *Let T be a PC-tree before a restriction during the planarity test on a biconnected planar multigraph $G = (V, E)$, let $c \in V(T')$ be a full C-node in the PC-tree*



(a) Paths if v_b is connected to $P_E(v_b)$.



(b) Paths if v_b is connected to $P_F(v_b)$.

Figure 4.2.: Illustration of paths p_1, p_2, p_3 during C-node construction.

after the restriction for the new node n and before the merge. Then $R_c + n$ is a maximally triconnected component of G.

Proof. We first show that $R_c + n$ is triconnected in G. Because of $\deg(c) \geq 3$, there exist three independent paths from every vertex $x \in R_c$ to n: Inside R_c, we construct the independent paths to the "border" of R_c just like the paths q'_1, q'_2, q'_3 in the proof of Lemma 4.5. Outside R_c, given that c is full, we either have that c has exactly $\deg(c) - 1$ full neighbors or exactly $\deg(c)$ full neighbors. If c has exactly $\deg(c) \geq 3$ full neighbors, then there exist three independent paths from c to full leaves in T', which represent edges connecting to n, thus three independent paths from x to n also exist in G for all $x \in R_c$. If c has exactly $\deg(c) - 1 \geq 3 - 1 = 2$ full neighbors, then there exist two independent paths from c to leaves in T' that represent edges connecting to n. Moreover, there exists another path from c to an empty leaf in T'. Therefore, three independent paths from x to n also exist in G for all $x \in R_c$.

Now, let $M := R_c + n$. We now want to show that M is indeed maximally triconnected. We focus on vertices that are merged into the PC-tree in later steps. Since c is full, we now would have at most two boundary edges in M, one edge connected to n and one edge connected to the possibly existent vertex in R_c that is connected to a non-full neighbor of c. Due to these two "bottlenecks", there cannot exist another vertex z that is merged in a later step such that there exist at least three independent paths between z and any vertex in M. In conjunction with Assumption 2, we now get the maximality of the generated rigid. □

4.4. Deriving polygons

In SPQR-trees, polygons are represented by S-nodes. Polygons are simple cyclic graphs. First, we get the following result that characterizes separation pairs of G in polygons.

Lemma 4.7. *Let S be a polygon of an SPQR-tree of a biconnected undirected multigraph G, let $v, w \in V(S)$ with $v \neq w$. Then v, w is not a separation pair of G if and only if $\{v, w\} \in E(S)$ and $\{v, w\}$ is a real edge.*

Proof. Simply follows from the definition of polygons as skeletons and the fact that if $\{v, w\} \in E(S)$ is a real edge, then removing $\{v, w\}$ from G does not increase the count of connected components since the edge itself would be removed as well. □

We now can describe the positions of the vertices of polygons in DFS trees.

Lemma 4.8. *Let S be a polygon of an SPQR-tree of a biconnected undirected multigraph G, let T be a directed DFS tree of G. Then all vertices of S are part of a single path in T.*

Proof. Assume the above statement is wrong, then there exist $v, w \in V(S)$ such that v and w lie in disjunct subtrees of T. If v, w is a separation pair of G, then this is a contradiction to Lemma 4.1. If v, w is not a separation pair of G, then we get that $\{v, w\} \in E(S)$ and $\{v, w\}$ is a real edge from Lemma 4.7. Therefore, we have $\{v, w\} \in E$, which violates the DFS as v and w lie in disjunct subtrees. □

Now, let S be a polygon of an SPQR-tree of a biconnected undirected multigraph G. From Lemma 4.8, we infer that when retreating to a new vertex $v \in V(S)$ during the planarity test, all other vertices of $V(S)$ with greater depth than v have been merged previously in

the "same" PC-tree. Therefore, for every PC-tree, we need to consider all vertices that have been merged into a PC-tree to find polygons. In general, during the planarity test, when retreating to a new vertex v, a P-node is created with label v. During further operations, this information on which vertices had been encountered by the PC-tree may though be lost. In Section 4.3, we have seen that vertices, which are part of rigids, are stored in the subgraphs of C-nodes, thus these labels are not lost. Therefore, we only have to consider vertices that are not part of any rigids. In general, label lists on edges are now used to store the labels of removed P-nodes in PC-trees. Given a PC-tree T and an edge $\{p, q\} \in E(T)$ with $p, q \in V(T)$, we thus define the $l_T(p, q)$ as the list of labels on the edge $\{p, q\}$. Given a P-node k in a PC-tree, then the label of k is the label *incident* to all edges that are connected to k. Given a C-node k in a PC-tree and an edge e that is incident to k, then the label in k that is *incident* to $\{k, k'\}$ is the vertex in R_k which has a boundary edge that is connected to k'. To generate these label lists, we now define the following rule.

Rule 5 (Label list expansion)**.** *Let T' be a PC-tree after a restriction for a new node n during the planarity test of a biconnected undirected graph $G = (V, E)$, let T'' be the PC-tree after merging a vertex $n \in V$ as a P-node m, let $p \in V(T')$ a P-node or C-node with exactly* $\deg(p) - 1$ *full neighbors such that, given the non-full neighbor q of p,* $\text{subtree}(p)$ *at q contains all full leaves, let v be the label in p that is incident to the edge $\{p, q\}$, let w be the label in q that is incident to the edge $\{p, q\}$ if q is not a leaf. Then we define $l_{T''}(m, q)$ as follows ($\cup$ signifies list concatenation):*

$$l_{T''}(m,q) := \begin{cases} () & \text{if } l_{T'}(p,q) = () \text{ and } (v = w \text{ or } q \text{ is a leaf}) \\ l_{T'}(p,q) & \text{if } v = l_{T'}(p,q).\text{firstElement} \\ (v) \cup l_{T'}(p,q) & \text{otherwise} \end{cases}$$

This rule realizes both edge types to be generated in polygons: Virtual edges represent bridges, which are "compressed" by this rule into a single edge based on the relation of the labels. Real edges are realizes if there was only a single full leaf and p is a P-node with degree 2. Now, having saved the labels in the PC-tree, we construct polygons of the SPQR-tree by using the following rules. Here, a polygon is described by a linear order L_S of vertices, whose ends can be connected to form the cyclic order of vertices in the polygon.

Rule 6 (Polygon construction from circle)**.** *Let T' be a PC-tree after a restriction for a new node n during the planarity test of a biconnected undirected graph $G = (V, E)$, let $p \in V(T')$ be full with exactly* $\deg(p) - 1$ *full neighbors such that, given the non-full neighbor q of p,* $\text{subtree}(p)$ *at q does not contain all full leaves, let v be the label in p that is incident to the edge $\{p, q\}$ if p is not a leaf, let w be the label in q that is incident to the edge $\{p, q\}$ if q is not a leaf. We define $k_p := (v)$ if p is not a leaf and $k_p := ()$ if p is a leaf, ditto for k_q. Then we get a polygon of the SPQR-tree of G by the following linear order L_S of vertices ($\cup$ signifies list concatenation):*

$$L_S := k_p \cup l_{T'}(p, q) \cup k_q \cup (n)$$

Rule 7 (Polygon construction from enclosure)**.** *Let T be a PC-tree before a restriction for a new node n during the planarity test of a biconnected undirected graph $G = (V, E)$, let T' be the PC-tree after the restriction, let t be the terminal path in T, let $p, q \in V(t)$ such that $\{p, q\} \in E(t)$ and $l_T(p, q) \neq ()$, let v be the label in p that is incident to the edge $\{p, q\}$, let w be the label in q that is incident to the edge $\{p, q\}$. Then we get a polygon of the SPQR-tree of G by the following linear order L_S of vertices:*

$$L_S := (v) \cup l_T(p, q) \cup (w)$$

Rule 6 formalizes the construction of a polygon when we have found its final vertex. Rule 7 deals with a special case where the nodes at the end of a label list are actually merged into a C-node during the restriction.

5. Evaluation

In this chapter, we evaluate and compare our implementation of our embedding algorithm from Chapter 3. First, we describe implementation and evaluation details in Section 5.1. Then, we present and describe our evaluation results in Section 5.2.

5.1. Implementation and evaluation details

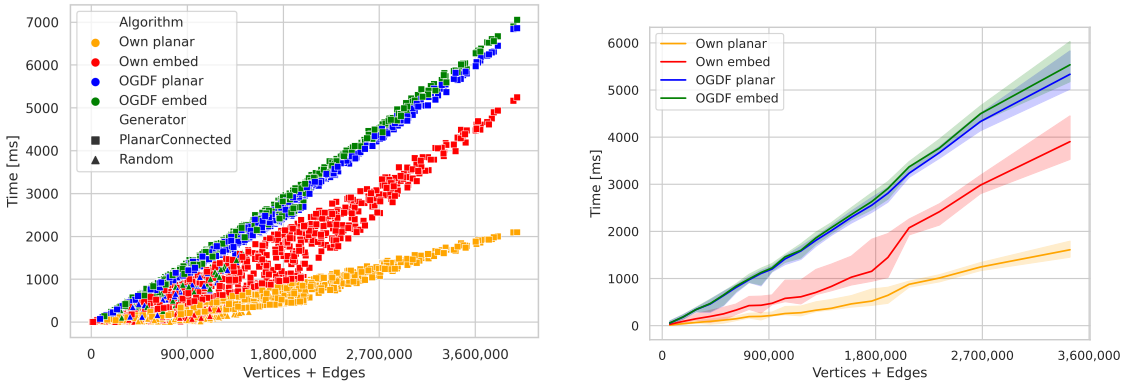
In order to implement our embedding algorithm, we used C++ version 17, the Open Graph Drawing Framework (OGDF) version 2022.02 (Dogwood) and the PC-tree implementation by Fink et al. [FPR21]. For benchmarking, we used one of the "Chimaira"-nodes in the Infosun cluster of the University of Passau, which has the following specifications:

- CPU: Intel Xeon E5-2690v2 (10 cores, 20 threads)
- RAM: 64 GB
- OS: GNU/Linux Debian 11 with kernel version 5.10.0-24-amd64

Note that we ran benchmarks while having exclusive access to that node, i.e. no other jobs of the Slurm cluster were running on that node. We ran at most 8 benchmarks of different graphs concurrently in different processes. For measuring runtimes, we subtracted ending and starting time obtained by calling the function *std::chrono::steady_clock::now*. In order to decrease run-to-run variance by external factors such as CPU-scheduling, CPU-boosting behavior and operating system overhead, we repeated every run on each graph 5 times and took the average running time from those runs as our measured value.

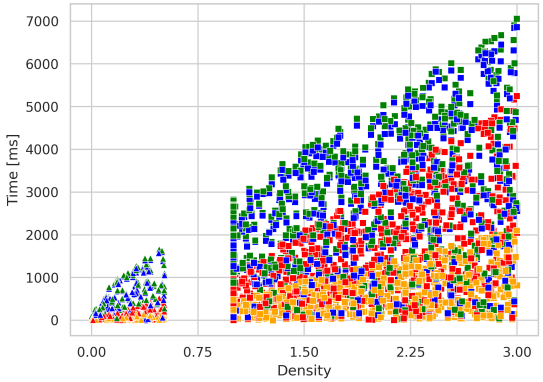
5.2. Evaluation results

We compared our embedding algorithm to the embedding algorithm by Boyer and Myrvold [BM04], which is included in the OGDF. Additionally, we measured our planarity test and the Boyer and Myrvold planarity test contained in the OGDF. Note that our planarity test is simply an implementation of the Haeupler and Tarjan planarity test without our extensions for computing a planar embedding. We used two different datasets for evaluation: randomly generated graphs, see Section 5.2.1, and graph generated by the OGDF test function forEachGraphItWorks, see Section 5.2.2. Most plots in this chapter have two versions next to one another, a scatter plot and a line plot, both depicting the same dataset. Most plots are colored based on the used algorithm. Scatter plots may also use different

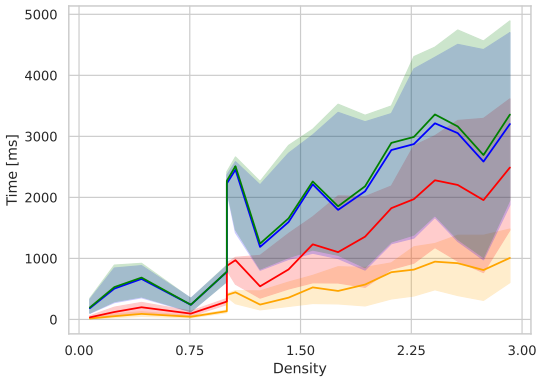


(a) Scatter plot by graph size.

(b) Line plot by graph size.



(c) Scatter plot by density.



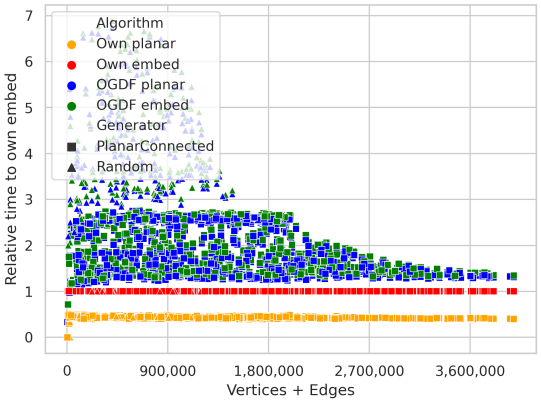
(d) Line plot by density.

Figure 5.1.: Running times for random planar graphs.

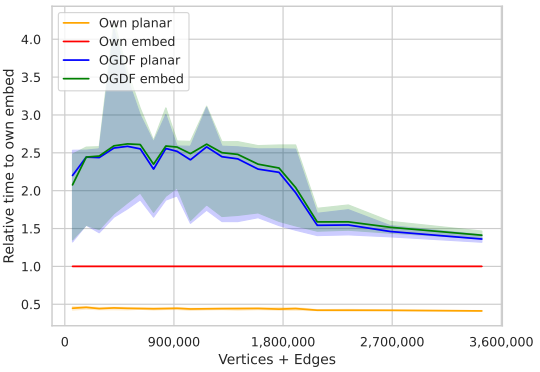
symbols to illustrate differences of the input graphs. Details can be found in the legends of the plots. In the line plots, the line represents the median value and the shaded area represents the interquartile range, based on an assignment of the measurements into 20 buckets with approximately same sizes. The plots show running times in relation to either the sum of vertices and edges or in relation to the graph density. For a graph $G = (V, E)$, we define its *size* as its sum of vertex count and edge count, i.e. $|V| + |E|$. Moreover, its *density* equals its edge count divided by its vertex count, i.e. $\frac{|E|}{|V|}$.

5.2.1. Random graphs

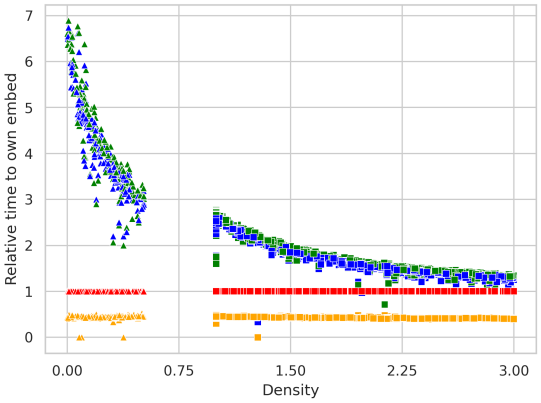
We evaluated our embedding implementation on random graphs. Random planar graphs where specifically generated using the OGDF function *ogdf::randomPlanarConnectedGraph*. Additionally, other graphs were generated using *ogdf::randomGraph*. Those generator functions are abbreviated in the plots with "PlanarConnected" and "Random" respectively. Graphs that are generated by ogdf::randomGraph may be planar or non-planar, thus they were categorized accordingly. In total, we tested $2,000$ random graphs, of which $1,159$ were planar and 841 were non-planar, with up to $1,000,000$ vertices and up to $3,000,000$ edges. In Figure 5.1, we have four plots that show the running time for every random planar graph $G = (V, E)$ based on either its size, see Figures 5.1a and 5.1b or based on its density, see Figures 5.1c and 5.1d. First, we observe that there is a gap in the dataset around the density value 0.75. This can be explained by the usage of both generator functions randomGraph and randomPlanarConnectedGraph. As we see in Figure 5.1c, there are some graphs generated by randomGraph with density about less than 0.5, then we have a gap until density about 1, where we see a bulk of graphs that are generated by randomPlanarConnectedGraph. Since randomPlanarConnectedGraph always generates



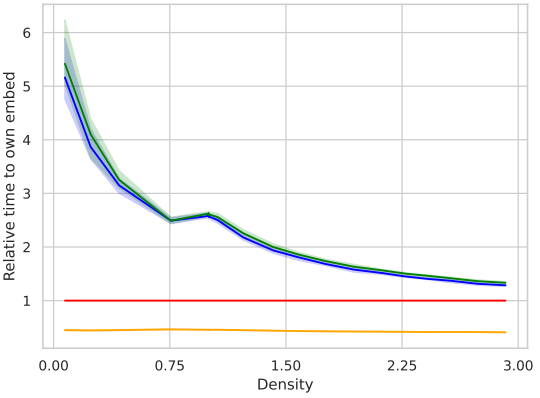
(a) Scatter plot by graph size.



(b) Line plot by graph size.

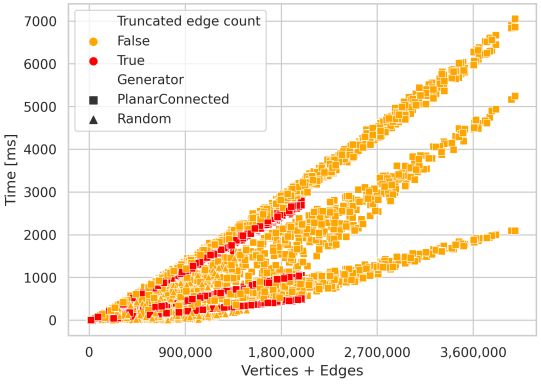


(c) Scatter plot by density.

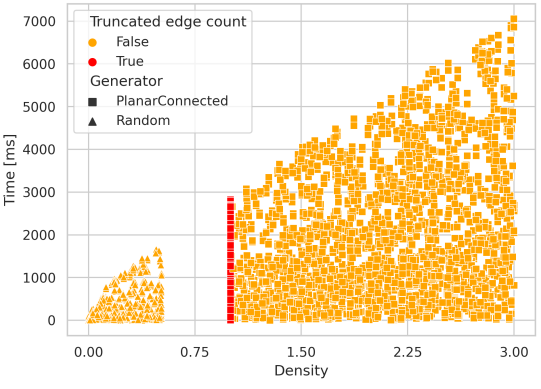


(d) Line plot by density.

Figure 5.2.: Relative running times for random planar graphs.

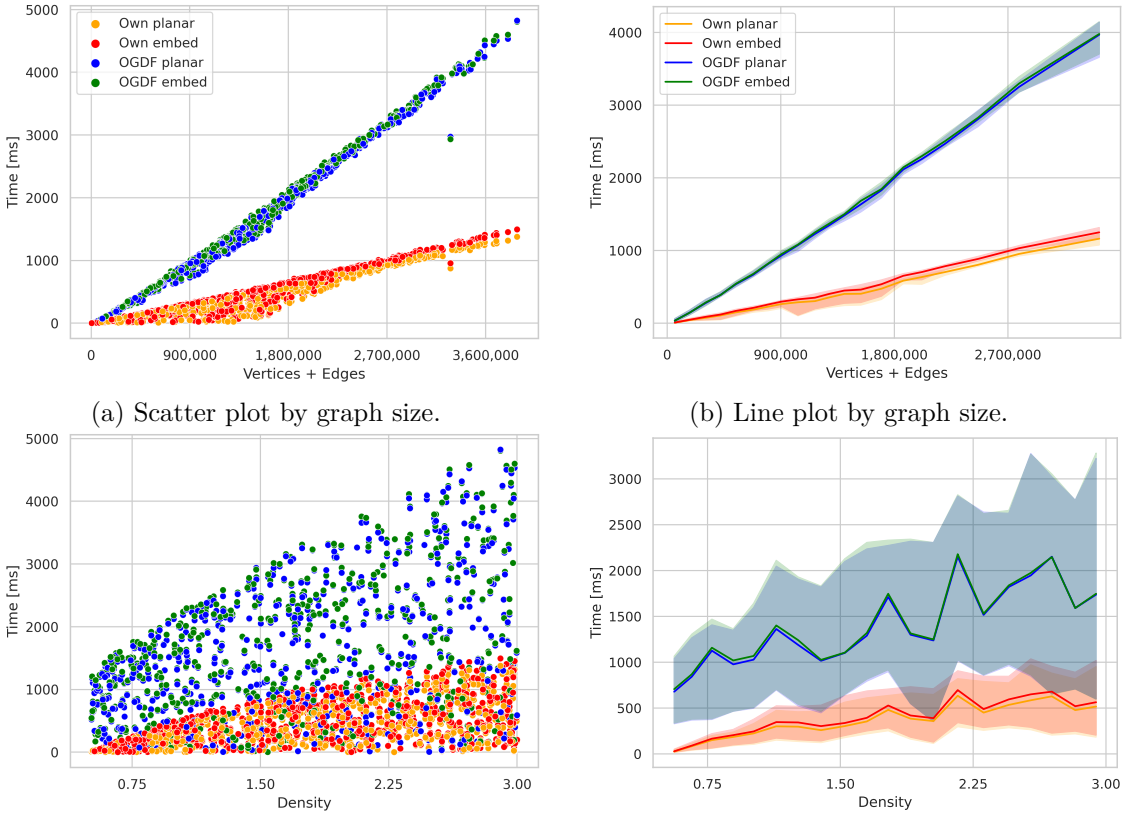


(a) Scatter plot by graph size $|V| + |E|$.



(b) Line plot by graph size $|V| + |E|$.

Figure 5.3.: Running times for random planar graphs with marked truncation of edge counts by randomPlanarConnectedGraph.



(a) Scatter plot by graph size.

(b) Line plot by graph size.

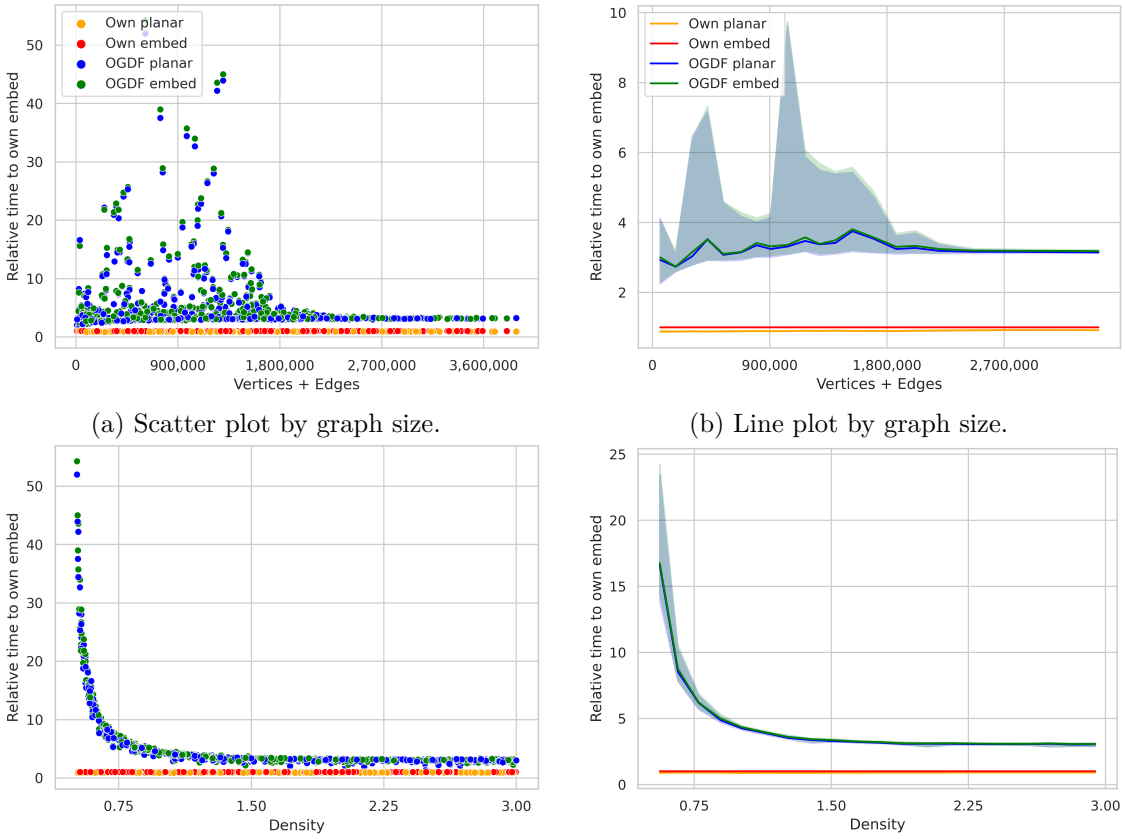
(c) Scatter plot by density.

(d) Line plot by density.

Figure 5.4.: Running times for random non-planar graphs.

connected graphs, these graphs all have a density of $\frac{|E|}{|V|} \geq \frac{|V|-1}{|V|} \xrightarrow[|V|\to\infty]{} 1$. Additionally, the target edge count of a generated graph is selected uniformly at random from $\{0, \ldots, 3 \cdot |V|\}$. Given that randomPlanarConnectedGraph must return a connected graph, the edge count of all graphs with target edge count less than $|V| - 1$ actually is truncated to an edge count of $|V| - 1$, which leads to a density of about 1. This explanation is also supported by Figure 5.3b, which shows where truncation occurs. This explains the vertical "line" in the scatter plot. Furthermore, the probability of a random graph being planar, which are outputted by randomGraph, decreases with the number of edges. Therefore, get this gap around a density of 0.75, and we only find outputs of randomGraph to the left of this gap. Moreover, in the scatter plot of Figure 5.1a, we see that up to a graph size of $2,000,000$, running times are also concentrated into single lines with a lower gradient than the median gradient. These lines are more visible in Figure 5.3a, which also confirms that this is caused by the truncation of randomPlanarConnectedGraph. From the plots by graph size, we infer that all four algorithms seem to have a linear running time regarding the graph size. Additionally, our embedding algorithm on average takes a bit more than twice the time of the Haeupler-Tarjan planarity test, which it is still faster than the OGDF planarity test and embedding algorithm, which are both about 30% slower than our embedding algorithm. This value is also illustrated by the relative time plots in Figure 5.2. In Figure 5.1d, we also see that our algorithms have a lesser variance than the OGDF algorithms. Concerning the plots by density, we also see here that our embedding algorithm is faster than the OGDF algorithms. Note that both OGDF algorithms always have very similar running times, suggesting that the OGDF planarity test may perform unnecessary computations.

We now discuss the plots of random non-planar graphs in Figure 5.4. From the plots by graph size, we infer that all four algorithms seem to have a linear running time regarding



(a) Scatter plot by graph size.

(b) Line plot by graph size.

(c) Scatter plot by density.

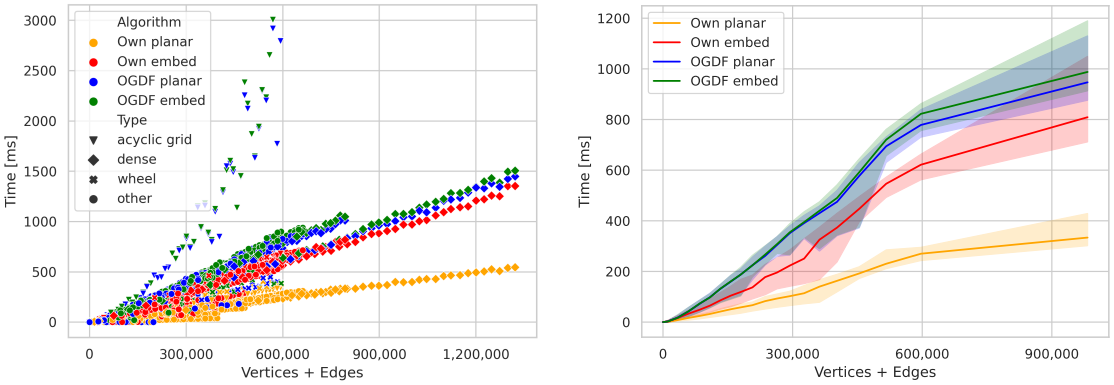
(d) Line plot by density.

Figure 5.5.: Relative running times for random non-planar graphs.

the graph size. From both plots, by size and by density, we infer that on average, our embedding algorithm is only very slightly slower than the Haeupler-Tarjan planarity test on non-planar graphs. This suggests that in practice, the computation for calculating low pointers and the values for φ, which are performed during the course of the planarity test until the test terminates, have hardly any overhead. The plot by graph size shows that on average, the OGDF planarity test and embedding algorithm take around 3 times as long as our algorithms, when disregarding small graph densities, where the OGDF algorithms are much slower, as shown in Figure 5.5.

5.2.2. Graphs from forEachGraphItWorks

In this section, we use a dataset of graphs that was generated by the function *forEachGraphItWorks* found in the testing part of the OGDF library, which generates graphs based on different types. In total, we generated $1,150$ graphs for this dataset, with up to $200,000$ vertices. Note that due to several problems, not all graph types that are generated by forEachGraphItWorks could be included in this dataset. The graph type "path-like tree" was not included due to dependency issues with resource files in the OGDF, the type "connected dense graph" was not included because it generates graphs with $|V|^2/4$ edges, thus large graphs with up to $200,000$ vertices would yield to up to $200,000^2/4 = 10^{10}$ edges, which thus requires at least tens of gigabytes of RAM to generate one graph, which is not practical. For a complete list of all used graph types, see Appendix Section A. In the scatter plots, some noteworthy type distinctions are illustrated using different symbols, as annotated by the legends. Additionally, we omitted density plots for this dataset, as the generated graphs have "very discrete" densities, therefore those plots are not very expressive.



(a) Scatter plot by graph size.

(b) Line plot by graph size.

Figure 5.6.: Running times for planar graphs from forEachGraphItWorks.

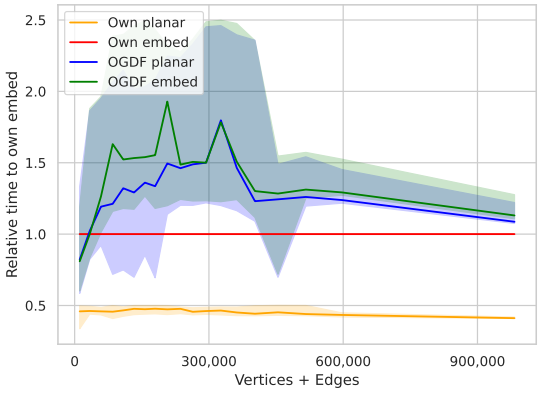
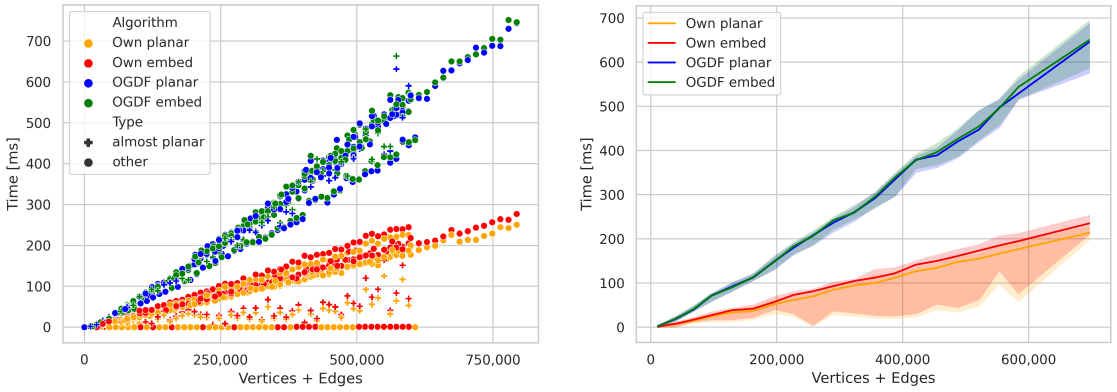


Figure 5.7.: Relative running times for planar graphs from forEachGraphItWorks.

Concerning planar graphs from forEachGraphItWorks, in general, Figure 5.6a shows a linear correlation between graph size and running time for our algorithms. The OGDF algorithms also mostly have this linear correlation, but for the graph type of "acyclic grid" graphs, there seems to be a "quadratic-like" correlation between graph size and processing time, which is not present with our algorithms. Looking at the corresponding line graph in Figure 5.6b, we see that the "acyclic grid" as outlier cases are omitted, since they do not fall in the interquartile range. Moreover, wee see that the horizontal axis only shows values for graph sizes up to about $1,000,000$, which is lower than in the scatter plot. This is due to the bucketing that is performed, because there are few graphs, all dense graphs, with such high sizes in our dataset. Therefore, our line plot in fact represents our scatter plot more zoomed in. We observe that there is a bend in our linear correlation to a more shallow gradient at a graph size of about $600,000$ for all four algorithms, which can be explained by the existence of more "fast" graphs in our dataset, i.e. graphs with lower processing time than the linear correlation to the right of $600,000$. These graphs do not reach higher size values. Concerning the relative running times displayed in Figure 5.7, we see that at around a graph size of $1,000,000$, the OGDF embedding algorithm is about 15% slower than our algorithm, but the OGDF embedding algorithm seems to approach our embedding algorithm with increasing graph size. Due to the above-mentioned problems with the forEachGraphItWorks, we were not able to test if the OGDF algorithm eventually reaches or surpasses the speed of our algorithm. It must be noted that with increasing size, this dataset only specifically contains dense graphs starting from a graph size of about $700,000$, as visible in Figure 5.6a. Note that for this forEachGraphItWorks dataset, we omitted scatter plots for relative runtimes, as outlier data values, where the OGDF



(a) Scatter plot by graph size.

(b) Line plot by graph size.

Figure 5.8.: Running times for non-planar graphs from forEachGraphItWorks.

algorithms have significantly larger relative running time than our algorithm, compress the scale of the vertical axis such that the plot is not expressive.

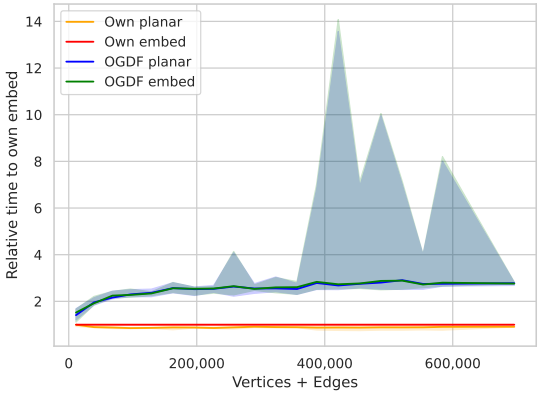


Figure 5.9.: Relative running times for non-planar graphs from forEachGraphItWorks.

Concerning non-planar graphs from forEachGraphItWorks, we see in Figure 5.8 that there is also a linear correlation between graph size and processing time. Notice that here, our embedding algorithm also has a small overhead compared to our Haeupler-Tarjan planarity test, as seen with random graphs. In relation to the OGDF algorithms, we see that our algorithms are about three times as fast for non-planar graphs in this dataset, which is visible in Figure 5.9.

6. Conclusion

In this paper, we have seen that we can extend the Haeupler-Tarjan planarity test to form an embedding algorithm, which also runs in linear time $\mathcal{O}(|V| + |E|)$ given an input graph $G = (V, E)$. During the evaluation of our implementation, we have seen that our embedding algorithm is consistently faster than the embedding algorithm by Boyer and Myrvold that is implemented in the Open Graph Drawing Framework library. Specifically, on random graphs, the library embedding algorithm was consistently at least 30% slower than our algorithm on planar graphs and consistently at least 300% slower on non-planar graphs than our algorithm. Furthermore, we devised a further extension to the Haeupler-Tarjan planarity test to simultaneously compute the SPQR-tree of a biconnected input graph. We thus obtain an algorithm that performs a planarity test, and further outputs a planar embedding and the SPQR-tree of the biconnected input graph if it is planar.

Based on this thesis, further research can be taken into many directions. We have presented an approach for computing SPQR-trees during the Haeupler-Tarjan planarity test, which utilizes the duality of PC-trees and SPQR-trees. Further research may continue to investigate this duality as well as our algorithm, particularly its runtime. Additionally, there not yet exists an implementation of our SPQR-tree algorithm, which may also be tested in practice. Furthermore, our approach only computes SPQR-trees for planar biconnected graphs, but the algorithm could be modified to deal with non-planar and non-biconnected graphs. In case of non-planar graphs, the planarity test normally terminates when discovering an impossible restriction. In order to continue nonetheless, "R-nodes" may be used in PC-trees to represent C-nodes where the cyclic order is not respected anymore, similar to R-nodes in PQR-trees [TM05]. Concerning non-biconnected graphs, during the planarity test, cut-vertices can easily be detected with the help of the PC-trees and the algorithm would output an SPQR-forest. Furthermore, our algorithm for computing an embedding and SPQR-tree may be combined in order to compute embeddings for all triconnected components of the input graph. Finally, the correlation between the Haeupler-Tarjan approach using PC-trees or PQ-trees and the approach by Hopcraft and Tarjan [HT73] and by Gutwenger and Mutzel [GM01] that use low pointers for computing SPQR-trees may be investigated further.

Bibliography

[AH92] L.A. Al-Hakim. A new graph-theoretic formulation for VLSI floorplan design. In *TENCON'92 - Technology Enabling Tomorrow*, volume 1, pages 146–150, 1992.

[BM04] John M. Boyer and Wendy J. Myrvold. On the cutting edge: simplified planarity by edge addition. *Journal of Graph Algorithms and Applications*, 8(3):241–273, 2004.

[Bra09] Ulrik Brandes. The left-right planarity test. *Manuscript submitted for publication*, 3, 2009.

[CHH99] Zhi-Zhong Chen, Xin He, and Chun-Hsi Huang. Finding double Euler trails of planar graphs in linear time [CMOS VLSI circuit design]. In *40th Annual Symposium on Foundations of Computer Science (Cat. No.99CB37039)*, pages 319–329, 1999.

[CLRS22] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms, Forth Edition.* The MIT Press, 2022.

[CNAO85] Norishige Chiba, Takao Nishizeki, Shigenobu Abe, and Takao Ozawa. A Linear Algorithm for Embedding Planar Graphs Using PQ-Trees. *Journal of Computer and System Sciences*, 30(1):54–76, February 1985.

[DBT89] Giuseppe Di Battista and Roberto Tamassia. Incremental Planarity Testing. In *Proceedings of the 30th Annual IEEE Symposium on Foundations of Computer Science (FOCS'89)*, pages 436–441, October 1989.

[Die17] Reinhard Diestel. *Graph Theory.* Graduate Texts in Mathematics. Springer, 2017.

[FC21] Abdul Faisal and Priya Chandran. GLDraw: A Platform for Graph Visualization. In *2021 6th International Conference for Convergence in Technology (I2CT)*, pages 1–6, 2021.

[FPR21] Simon D. Fink, Matthias Pfretzschner, and Ignaz Rutter. Experimental Comparison of PC-Trees and PQ-Trees. In Petra Mutzel, Rasmus Pagh, and Grzegorz Herman, editors, *29th Annual European Symposium on Algorithms (ESA 2021)*, volume 204 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 43:1–43:13, Dagstuhl, Germany, 2021. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

[FR23] Simon D. Fink and Ignaz Rutter. Maintaining Triconnected Components under Node Expansion, 2023.

[Fre22] Valentin Frey. Planarity Testing and Embedding based on the Haeupler-Tarjan algorithm: Implemention and Experiments, 2022.

[GM01] Carsten Gutwenger and Petra Mutzel. A Linear Time Implementation of SPQR-Trees. In *Proceedings of the 8th International Symposium on Graph Drawing (GD'00)*, volume 1984 of *Lecture Notes in Computer Science*, pages 70–90. Springer, January 2001.

[HT73] John E. Hopcroft and Robert E. Tarjan. Dividing a graph into triconnected components. *SIAM Journal on Computing*, 2(3):135–158, September 1973.

[HT74] John E. Hopcroft and Robert E. Tarjan. Efficient Planarity Testing. *Journal of the ACM*, 21(4):549–568, October 1974.

[HT08] Bernhard Haeupler and Robert E. Tarjan. Planarity Algorithms via PQ-Trees (Extended Abstract). *Electronic Notes in Discrete Mathematics*, 31:143–149, 2008. The International Conference on Topological and Geometric Graph Theory.

[LEC67] Abraham Lempel, Shimon Even, and Israel Cederbaum. An Algorithm for Planarity Testing of Graphs. In *Proceedings of the International Symposium on the Theory of Graphs*, pages 215–232. Gordon and Breach, 1967.

[Nov82] Vítězslav Novák. Cyclically ordered sets. *Czechoslovak Mathematical Journal*, 32(3):460–473, 1982.

[Tam13] Roberto Tamassia, editor. *Handbook of Graph Drawing and Visualization* . CRC Press, 2013.

[TM05] Guilherme Telles and Joao Meidanis. Building PQR trees in almost-linear time. *Electronic Notes in Discrete Mathematics*, 19:33–39, June 2005.

[VVK09] Jussi Vanhatalo, Hagen Völzer, and Jana Koehler. The refined process structure tree. *Data & Knowledge Engineering*, 68(9):793–818, 2009. Sixth International Conference on Business Process Management (BPM 2008) – Five selected and extended papers.

[WKC88] Shmuel Wimer, Israel Koren, and Israel Cederbaum. Floorplans, planar graphs, and layouts. *IEEE Transactions on Circuits and Systems*, 35(3):267–278, 1988.

Appendix

A. Graph types from forEachGraphItWorks

The following graph types generated by forEachGraphItWorks were used in the evaluation:

- graph with a single node
- graph with a single node and one self-loop
- graph without any nodes
- graph with three nodes and no edge
- graph with three nodes and one edge
- graph with two nodes and directed parallel edges
- graph with two nodes and no edge
- graph with two nodes and one edge
- graph with two nodes and two edges (one self-loop)
- graph with two nodes and undirected parallel edges
- K2,3
- K3,3
- K4
- K5
- non-upward planar graph
- Petersen graph
- 3-regular arborescence
- 4-regular graph
- acyclic biconnected non-planar graph
- acyclic biconnected planar graph
- acyclic grid graph

- arborescence
- arborescence forest
- biconnected almost planar graph
- biconnected graph
- connected planar graph
- connected sparse graph
- disconnected planar graph
- isolated nodes
- maximal planar graph
- path with multi-edges
- planar dense triconnected multi-graph
- planar sparse triconnected multi-graph
- series parallel DAG
- triconnected graph
- triconnected planar graph
- wheel graph