Disconnectivity and relative positions in simultaneous embeddings ☆,☆☆

Thomas Bläsius *, Ignaz Rutter *

Karlsruhe Institute of Technology (KIT), Germany

A R T I C L E   I N F O

A B S T R A C T

For two planar graphs $G^{①} = (V^{①}, E^{①})$ and $G^{②} = (V^{②}, E^{②})$ sharing a common subgraph $G = G^{①} \cap G^{②}$ the problem SIMULTANEOUS EMBEDDING WITH FIXED EDGES (SEFE) asks whether they admit planar drawings such that the common graph is drawn the same. Previous algorithms only work for cases where G is connected, and hence do not need to handle relative positions of connected components. We consider the problem where G, $G^{①}$ and $G^{②}$ are not necessarily connected.

First, we show that a general instance of SEFE can be reduced in linear time to an equivalent instance where $V^{①} = V^{②}$ and $G^{①}$ and $G^{②}$ are connected. Second, for the case where G consists of disjoint cycles, we introduce the *CC-tree* which represents all embeddings of G that extend to planar embeddings of $G^{①}$. We show that CC-trees can be computed in linear time, and that their intersection is again a CC-tree. This yields a linear-time algorithm for SEFE if all k input graphs (possibly $k > 2$) pairwise share the same set of disjoint cycles. These results, including the CC-tree, extend to the case where G consists of arbitrary connected components, each with a fixed planar embedding on the sphere. Then the running time is $O(n^2)$.

© 2015 Elsevier B.V. All rights reserved.

1. Introduction

To enable a human reader to compare different relational datasets on a common set of objects it is important to visualize the corresponding graphs in such a way that the common parts of the different datasets are drawn as similarly as possible. An example is a dynamic graph that changes over time. Then the change between two points in time can be easily grasped with the help of a visualization showing the parts that did not change in the same way for both graphs. This leads to the fundamental theoretical problem SIMULTANEOUS EMBEDDING WITH FIXED EDGES (or SEFE for short), asking for two graphs $G^{①} = (V^{①}, E^{①})$ and $G^{②} = (V^{②}, E^{②})$ with the common graph $G = (V, E) = (V^{①} \cap V^{②}, E^{①} \cap E^{②})$, whether there are planar drawings of $G^{①}$ and $G^{②}$ such that the common graph G is drawn the same in both.

The problem SEFE and its variants, such as SIMULTANEOUS GEOMETRIC EMBEDDING, where one insists on a simultaneous straight-line drawing, have been studied intensively in the past years; see the recent survey [1] for an overview. Some of
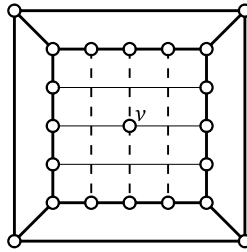
Fig. 1. The bold edges belong to both graphs, the dashed and thin edges are exclusive edges.

the results show, for certain graph classes, that they always admit simultaneous embeddings or that there exist negative instances of SEFE whose input graphs belong to these classes. As there are planar graphs that cannot be embedded simultaneously, the question of deciding whether given graphs admit a SEFE is of high interest. Gassner et al. [2] show that it is $\mathcal{NP}$-complete to decide SEFE for three or more graphs. For two graphs the complexity status is still open. However, there are several approaches yielding efficient algorithms for special cases. Fowler et al. show how to solve SEFE efficiently, if $G^{①}$ and G have at most two and one cycles, respectively [3]. Fowler et al. characterize the class of common graphs that always admit a SEFE [4]. Angelini et al. [5] show that if one of the input graphs has a fixed planar embedding, then SEFE can be solved in linear time. Haeupler et al. solve SEFE in linear time for the case that the common graph is biconnected [6]. Angelini et al. obtain the same result with a completely different approach [7]. They additionally solve the case where the common graph is a star and, moreover, show the equivalence of the case where the common graph is connected to the case where the common graph is a tree and relate it to a constrained book embedding problem. The currently least restrictive result (in terms of connectivity) by Bläsius and Rutter [8] shows that SEFE can be solved in polynomial time for the case that both graphs are biconnected and the common graph is connected.

The algorithms testing SEFE have in common that they use the result by Jünger and Schulz [9] stating that the question of finding a simultaneous embedding for two graphs is equivalent to the problem of finding planar embeddings of $G^{①}$ and $G^{②}$ such that they induce the same embedding on G. Moreover, they have in common that they all assume that the common graph is connected, implying that it is sufficient to enforce the common edges incident to each vertex to have the same circular ordering in both embeddings. Especially in the result by Bläsius and Rutter [8] this is heavily used, as they explicitly consider only orders of edges around vertices using PQ-trees. However, if the common graph is not required to be connected, we additionally have to care about the relative positions of connected components to one another, which introduces an additional difficulty. Note that the case where the common graph is disconnected cannot be reduced to the case where it is connected by inserting additional edges. Fig. 1 shows an instance that admits a simultaneous embedding, which is no longer true if the isolated vertex v is connected to the remaining graph. Other approaches to solve the SEFE problem have only appeared recently. Schaefer [10] characterizes, for certain classes of SEFE instances, the pairs of graphs that admit a SEFE via the independent odd crossing number. Among others, this gives a polynomial-time algorithm for SEFE when the common graph has maximum degree 3 and is not necessarily connected.

In this work we tackle the SEFE problem from the opposite direction than the so far known results, by assuming that the circular order of edges around vertices in G is already fixed and we only have to ensure that the embeddings chosen for the input graphs are *compatible* in the sense that they induce the same relative positions on G. Initially, we assume that the graph G consists of a set of disjoint cycles, each of them having a unique planar embedding. We present a novel data structure, the *CC-tree*, which represents all embeddings of a set of disjoint cycles that can be induced by an embedding of a graph containing them as a subgraph. We moreover show that two such CC-trees can be intersected, again yielding a CC-tree. Thus, for the case that $G^{①}$ and $G^{②}$ have the common graph G consisting of a set of disjoint cycles, the intersection of the CC-trees corresponding to $G^{①}$ and $G^{②}$ represents all simultaneous embeddings. We show that CC-trees can be computed and intersected in linear time, yielding a linear-time algorithm to solve SEFE for the case that the common graph consists of disjoint cycles. Note that this obviously also yields a linear-time algorithm to solve SEFE for more than two graphs if they all share the same common graph consisting of a set of disjoint cycles. We show that these results can be further extended to the case where the common graph may contain arbitrary connected components, each of them with a prescribed planar embedding. However, in this case the corresponding data structure, called $CC^{\oplus}$-tree, may have quadratic size. These results show that the choice of relative positions of several connected components does not solely make the problem SEFE hard to solve.

Note that these results have an interesting application concerning the problem PARTIALLY EMBEDDED PLANARITY. The input of PARTIALLY EMBEDDED PLANARITY is a planar graph G together with a fixed embedding for a subgraph H (including fixed relative positions). It asks whether G admits a planar embedding extending the embedding of H. Angelini et al. [5] introduced this problem and solve it in linear time. The $CC^{\oplus}$-tree can be used to solve PARTIALLY EMBEDDED PLANARITY in quadratic time as it represents all possible relative positions of the connected components in H to one another that can be induced by an embedding of G. It is then easy to test whether the prespecified relative positions can be achieved. In fact, this solves the slightly more general case of PARTIALLY EMBEDDED PLANARITY where not all relative positions have to be fixed.

The above described results have one restriction that was not mentioned so far. The graphs $G^{①}$ and $G^{②}$ are assumed to be connected, otherwise the approach we present does not work. Fortunately, we can show that both graphs of an instance of SEFE can always be assumed to be connected, even if all vertices are assumed to be common vertices (forming isolated vertices when not connected via a common edge). This shows that SEFE can be solved efficiently if the common graph consists of disjoint cycles without further restrictions on the connectivity. Moreover, it is an interesting result on its own as it applies to arbitrary instances of SEFE, not only to the special case we primarily consider here.

As connectivity plays an important role in this work we fix some basic definitions in the following. A graph is *connected* if there exists a path between any pair of vertices. A *separating k-set* is a set of k vertices whose removal disconnects the graph. Separating 1-sets and 2-sets are *cutvertices* and *separation pairs*, respectively. A connected graph is *biconnected* if it does not have a cut vertex and *triconnected* if it does not have a separation pair. The maximal biconnected components of a graph are called *blocks*. The *cut components* with respect to a separating k-set S are the maximal subgraphs that are not disconnected by removing S.

Outline. In Section 2 we show that, for any given instance of SEFE, there exists an equivalent instance such that both input graphs are connected, even if each vertex is assumed to be a common vertex. With this result instances of SEFE can always be assumed to have this property. In Section 3 we show how to solve SEFE in linear time if the common graph consists of disjoint cycles, including a compact representation of all simultaneous embeddings. In Section 4 we show how to extend these results to solve SEFE in quadratic time for the case that the common graph consists of arbitrary connected components, each with a fixed planar embedding. We conclude in Section 5.

2. Connecting disconnected graphs

Let $G^{①} = (V, E^{①})$ and $G^{②} = (V, E^{②})$ be two planar graphs with common graph $G = (V, E)$ with $E = E^{①} \cap E^{②}$. We show that the problem SEFE can be reduced to the case where $G^{①}$ and $G^{②}$ are required to be connected. First note that the connected components of the union of $G^{①}$ and $G^{②}$ can be handled independently. Thus we can assume that $G^{①} \cup G^{②}$ is connected. We first ensure that $G^{①}$ is connected without increasing the number of connected components in $G^{②}$. Afterwards we can apply the same steps to $G^{②}$ to make it connected, maintaining the connectivity of $G^{①}$.

Assume $G^{①}$ and $G^{②}$ consist of $k^{①}$ and $k^{②}$ connected components, respectively. Since the union of $G^{①}$ and $G^{②}$ is connected, we can always find an edge $e^{②} = \{v_1, v_2\} \in E^{②}$ such that the vertices v_1 and v_2 belong to different connected components $H_1^{①}$ and $H_2^{①}$ in $G^{①}$. We construct the *augmented instance* $(G_+^{①}, G_+^{②})$ of SEFE with respect to the edge $e^{②}$ by introducing a new vertex v_{12} and new edges $e = \{v_1, v_{12}\} \in E$ and $e^{①} = \{v_{12}, v_2\} \in E^{①}$. Note that $G_+^{①}$ has $k^{①} - 1$ connected components since $H_1^{①}$ and $H_2^{②}$ are now connected via the two edges e and $e^{①}$. Moreover, the number $k^{②}$ of connected components in $G^{②}$ does not change, since the edge e connects the new vertex v_{12} to one of its connected components. It remains to show that the original instance and the augmented instance are equivalent.

Lemma 1. *Let $(G^{①}, G^{②})$ be an instance of* SEFE *and let $(G_+^{①}, G_+^{②})$ be the augmented instance with respect to the edge $e^{②} = \{v_1, v_2\}$. Then $(G^{①}, G^{②})$ and $(G_+^{①}, G_+^{②})$ are equivalent.*

Proof. If the augmented instance admits a SEFE, then obviously the original instance does. To show the other direction assume the original instance $(G^{①}, G^{②})$ has a SEFE $(\mathcal{E}^{①}, \mathcal{E}^{②})$ inducing the embedding $\mathcal{E}$ for the common graph. We show how to construct an embedding $\mathcal{E}'^{①}$ such that (i) $(\mathcal{E}'^{①}, \mathcal{E}^{②})$ is a SEFE, and (ii) the vertices v_1 and v_2 lie on the border of a common face in $\mathcal{E}'^{①}$. Then we can easily add the vertex v_{12} together with the two edges e and $e^{①}$, yielding a SEFE of the augmented instance $(G_+^{①}, G_+^{②})$. Note that the first property, namely that $(\mathcal{E}'^{①}, \mathcal{E}^{②})$ is a SEFE, is satisfied if and only if the embeddings $\mathcal{E}^{①}$ and $\mathcal{E}'^{①}$ induce the same embedding $\mathcal{E}$ on the common graph. Fig. 2 illustrates the proof.

Consider a face f of the embedding $\mathcal{E}$ of the common graph. The embedding $\mathcal{E}^{①}$ of the graph $G^{①}$ splits this face f into a set of faces $\mathcal{F}^{①}(f) = \{f_1^{①}, \ldots, f_k^{①}\}$. We say that a face $f^{①} \in \mathcal{F}^{①}(f)$ is *contained* in f. Note that every face of $\mathcal{E}^{①}$ is contained in exactly one face of $\mathcal{E}$. The same definition can be made for the second graph.

The edge $e^{②} = \{v_1, v_2\}$ borders two faces $f_1^{②}$ and $f_2^{②}$ of $\mathcal{E}^{②}$. Since $e^{②}$ belongs exclusively to $G^{②}$ (otherwise v_1 and v_2 would not have been in different connected components in $G^{①}$) both faces $f_1^{②}$ and $f_2^{②}$ are contained in the same face f of the embedding $\mathcal{E}$ of the common graph G. We assume without loss of generality that f is the outer face. The face f may be subdivided by edges belonging exclusively to the graph $G^{①}$. However, we can find faces $f_1^{①}$ and $f_2^{①}$ of $\mathcal{E}^{①}$, both contained in f, such that v_1 and v_2 are contained in the boundary of these faces. If $f_1^{①} = f_2^{①}$ we are done since v_1 and v_2 lie on the boundary of the same face in $\mathcal{E}^{①}$. Otherwise, we split $G^{①}$ into two subgraphs $G_1^{①}$ and $G_2^{①}$ with the embeddings $\mathcal{E}_1^{①}$ and $\mathcal{E}_2^{①}$ induced by $\mathcal{E}^{①}$ as follows. The connected component $H_i^{①}$ (for $i = 1, 2$) containing v_i belongs to $G_i^{①}$ and all connected components that are completely contained in an internal face of $H_i^{①}$ also belong to $G_i^{①}$. All remaining connected components belong either to $G_1^{①}$ or to $G_2^{①}$. Note that this partition ensures that there is a simple closed curve in the outer
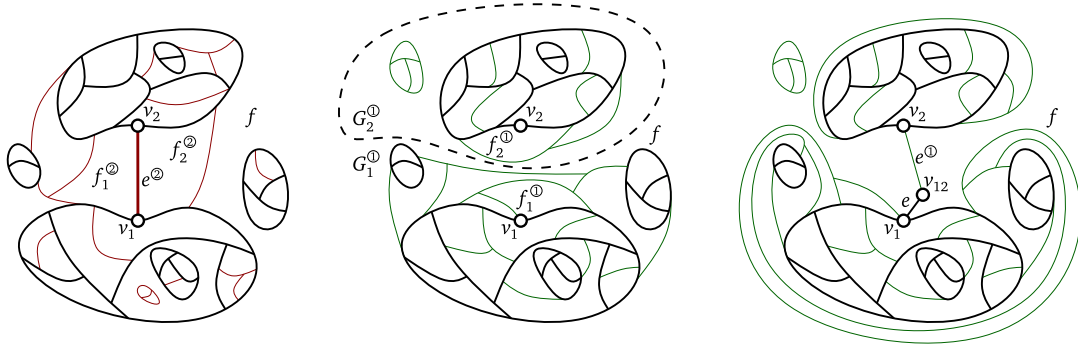
Fig. 2. Illustration of Lemma 1, the common graph is depicted black. The graph $G^{②}$ with the edge $e^{②} = \{v_1, v_2\}$ lying in the common face f, which is the outer face of G (left). The graph $G^{①}$ with the faces $f_1^{①}, f_2^{①} \in \mathcal{F}^{①}(f)$ incident to v_1 and v_2, respectively, partitioned into $G_1^{①}$ and $G_2^{①}$ (middle). The resulting graph $G^{①}$ after choosing $f_i^{①}$ as outer face of $G_i^{①}$ (for $i = 1, 2$) and inserting the vertex v_{12} and the edges e and $e^{①}$ (right).

face of $\mathcal{E}^{①}$ separating $G_1^{①}$ and $G_2^{①}$. Thus, we can change the embeddings of $\mathcal{E}_1^{①}$ and $\mathcal{E}_2^{①}$ independently. In particular, we choose the faces $f_1^{①}$ and $f_2^{①}$ to be the new outer faces, yielding the changed embeddings $\mathcal{E}_1'^{①}$ and $\mathcal{E}_2'^{①}$, respectively. When combining these to embeddings by putting $G_1^{①}$ into the outer face of $G_2^{①}$ and vice versa, we obtain a new embedding $\mathcal{E}'^{①}$ of $G^{①}$ with the following two properties. First, the embedding induced for the common graph does not change since both faces $f_1^{①}$ and $f_2^{①}$ belong to the outer face f of the embedding $\mathcal{E}$ of the common graph G. Second, the vertices v_1 and v_2 both lie on the outer face of the embedding $\mathcal{E}'^{①}$. Hence, $(\mathcal{E}'^{①}, \mathcal{E}^{②})$ is still a SEFE of the instance $(G^{①}, G^{②})$ and the vertex v_{12} together with the two edges e and $e^{①}$ can be added easily, which concludes the proof. $\square$

With this construction we can reduce the number of connected components of $G^{①}$ and $G^{②}$ and thus finally obtain an equivalent instance of SEFE in which both graphs are connected. We obtain the following theorem.

Theorem 1. *For every instance $(G^{①}, G^{②})$ of* SEFE *there exits an equivalent instance $(G_{++}^{①}, G_{++}^{②})$ such that $G_{++}^{①}$ and $G_{++}^{②}$ are connected. Such an instance can be computed in linear time.*

Proof. Lemma 1 directly implies that an equivalent instance $(G_{++}^{①}, G_{++}^{②})$ in which both graphs are connected exists. It remains to show that it can be computed in linear time. To connect all the connected components of $G^{①}$, we contract each of them to a single vertex in the graph $G^{②}$. Then an arbitrary spanning tree yields a set of edges $e_1^{②}, \ldots, e_k^{②} \in E^{②}$, such that augmenting the instance with respect to these edges yields a connected graph $G_{++}^{①}$. This works symmetrically for $G^{②}$ and can obviously be done in linear time. $\square$

3. Disjoint cycles

In this section, we consider the problem SEFE for the case that the common graph consists of a set of disjoint cycles. Due to Theorem 1, we can assume without loss of generality that both graphs are connected. In Section 3.1 we show how to solve this special case of SEFE in polynomial time. In Section 3.2 we introduce a tree-like data structure, the *CC-tree*, representing all planar embeddings of a set of cycles contained in a single graph that can be induced by an embedding of the whole graph. We additionally show that the intersection of the set of embeddings represented by two CC-trees can again be represented by a CC-tree, yielding a solution for SEFE even for the case of more than two graphs if all graphs have the same common graph, which consists of a set of disjoint cycles. In Section 3.3 we show how to compute the CC-tree and the intersection of two CC-trees in linear time. Before we start, we fix some definitions.

Embeddings of disjoint cycles. Let $\mathcal{C} = \{C_1, \ldots, C_k\}$ be a set of disjoint simple cycles. We consider embeddings of these cycles on the sphere. Since a single cycle has a unique embedding on the sphere only their relative positions to one another are of interest. To be able to use the terms "left" and "right" we consider the cycles to be directed. We denote the relative position of a cycle C_j with respect to a cycle C_i by $\text{pos}_{C_i}(C_j)$. More precisely, we have $\text{pos}_{C_i}(C_j) = $ "left" and $\text{pos}_{C_i}(C_j) = $ "right", if C_j lies on the left and right side of C_i, respectively. We call an assignment of a value "left" or "right" to each of these relative positions a *semi-embedding* of the cycles $\mathcal{C} = \{C_1, \ldots, C_k\}$. Note that not every semi-embedding yields an embedding of the cycles. For example if $\text{pos}_{C_i}(C_j) = \text{pos}_{C_j}(C_k) = $ "left" and $\text{pos}_{C_j}(C_i) = $ "right", then $\text{pos}_{C_i}(C_k)$ also needs to have the value "left"; see Fig. 3. However, two embeddings yielding the same semi-embedding are the same.
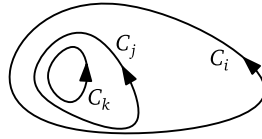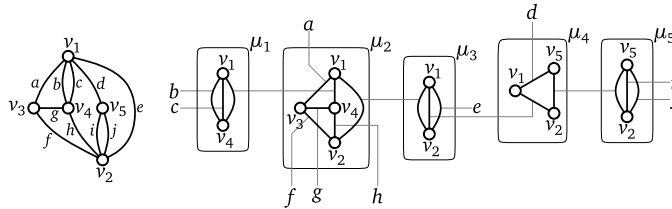
Fig. 3. Three nested cycles.

Fig. 4. The unrooted SPQR-tree of a biconnected planar graph. The nodes μ_1, μ_3 and μ_5 are P-nodes, μ_2 is an R-node and μ_4 is an S-node. The Q-nodes are not shown explicitly.

Sometimes we do not only consider the relative position of cycles but also of some other disjoint subgraph. We extend our notation to this case. For example the relative position of a single vertex v with respect to a cycle C is denoted by $\text{pos}_C(v)$.

SPQR- and BC-trees. The *block-cutvertex tree (BC-tree)* $\mathcal{B}$ of a connected graph is a tree whose nodes are the blocks and cutvertices of the graph, called *B-nodes* and *C-nodes*, respectively. In the BC-tree a block B and a cutvertex v are joined by an edge if v belongs to B. If an embedding is chosen for each block, these embeddings can be combined to an embedding of the whole graph if and only if $\mathcal{B}$ can be rooted at a B-node such that the parent of every other block B in $\mathcal{B}$, which is a cutvertex, lies on the outer face of B.

We use the *SPQR-tree* introduced by Di Battista and Tamassia [11,12] to represent all planar embeddings of a biconnected planar graph G. The SPQR-tree $\mathcal{T}$ of G is a decomposition of G into triconnected components along its *split pairs*, where a split pair is either a separation pair or an edge. We define the SPQR-tree to be unrooted, representing embeddings on the sphere, that is planar embeddings without a designated outer face. Let $\{s, t\}$ be a split pair and let H_1 and H_2 be two subgraphs of G such that $H_1 \cup H_2 = G$ and $H_1 \cap H_2 = \{s, t\}$. Consider the tree containing the two nodes μ_1 and μ_2 associated with the graphs $H_1 + \{s, t\}$ and $H_2 + \{s, t\}$, respectively. These graphs are called *skeletons* of the nodes μ_i, denoted by $\text{skel}(\mu_i)$ and the special edge $\{s, t\}$ is said to be a *virtual edge*. The two nodes μ_1 and μ_2 are connected by an edge or, more precisely, the occurrence of the virtual edges $\{s, t\}$ in both skeletons are linked by this edge. The *expansion graph* $\exp(\{s, t\})$ of a virtual edge $\{s, t\}$ is the subgraph of G it represents, that is in $\text{skel}(\mu_1)$ and $\text{skel}(\mu_2)$ the expansion graphs of $\{s, t\}$ are H_2 and H_1, respectively. Now a combinatorial embedding of G uniquely induces a combinatorial embedding of $\text{skel}(\mu_1)$ and $\text{skel}(\mu_2)$. Furthermore, arbitrary and independently chosen embeddings for the two skeletons determine an embedding of G, thus the resulting tree can be used to represent all embeddings of G by the combination of all embeddings of two smaller planar graphs. This replacement can of course be applied iteratively to the skeletons yielding a tree with more nodes but smaller skeletons associated with the nodes.

Applying this kind of decomposition in a systematic way yields the SPQR-tree as introduced by Di Battista and Tamassia [11,12]. The SPQR-tree $\mathcal{T}$ of a biconnected planar graph G contains four types of nodes. First, the skeleton of a P-node consists of a bundle of at least three parallel edges. Embedding the skeleton of a P-node corresponds to choosing an order for the parallel edges. Second, the skeleton of an R-node is triconnected, thus having exactly two embeddings [13], and third, S-nodes have a simple cycle as skeleton without any choice for the embedding. Finally, every edge in a skeleton representing only a single edge in the original graph G is formally also considered to be a virtual edge linked to a Q-node in $\mathcal{T}$ representing this single edge. Note that all leaves of the SPQR-tree $\mathcal{T}$ are Q-nodes. Besides from being a nice way to represent all embeddings of a biconnected planar graph, the SPQR-tree has size only linear in the size of G and Gutwenger and Mutzel [14] show that it can be computed in linear time. Fig. 4 shows a biconnected planar graph together with its SPQR-tree.

3.1. A polynomial-time algorithm

Let $(G^{\textcircled{1}}, G^{\textcircled{2}})$ be an instance of SEFE with common graph G consisting of pairwise disjoint simple cycles $\mathcal{C} = \{C_1, \ldots, C_k\}$. We first assume that $G^{\textcircled{1}}$ and $G^{\textcircled{2}}$ are biconnected and show later how to remove this restriction. Our approach is to formulate constraints on the relative positions of the cycles to one another ensuring that $G^{\textcircled{1}}$ and $G^{\textcircled{2}}$ induce the same semi-embedding on the common graph G. We show implicitly that the resulting semi-embedding is really an embedding by showing that the graphs $G^{\textcircled{1}}$ and $G^{\textcircled{2}}$ have planar embeddings inducing this semi-embedding. Note that this only works for the case that $G^{\textcircled{1}}$ and $G^{\textcircled{2}}$ are connected. Thus, our approach crucially relies on the result provided in Section 2.

Biconnected graphs

Before considering two graphs, we determine for a single graph the possible embeddings it may induce on a set of disjoint cycles contained in it. Let $G = (V, E)$ be a biconnected graph with SPQR-tree $\mathcal{T}$, let C be a simple directed cycle in G and let μ be a node in $\mathcal{T}$. Obviously, C is either completely contained in the expansion graph of a single virtual edge of μ or C induces a simple directed cycle of virtual edges in $skel(\mu)$. We say that C is *contracted* in $skel(\mu)$ in the first case and that C is *a cycle* in $skel(\mu)$ in the second case. If C is a cycle in $skel(\mu)$, we also say that $skel(\mu)$ *contains* C as a cycle. Consider the case where C is a cycle in $skel(\mu)$ and let κ denote this cycle. By fixing the embedding of $skel(\mu)$ the virtual edges in $skel(\mu)$ not contained in κ split into two groups, some lie to the left and some to the right of κ. Obviously, a vertex $v \in V \setminus V(C)$ in the expansion graph of a virtual edge that lies to the left (to the right) of κ lies to the left (to the right) of C in G, no matter which embedding is chosen for the skeletons of other nodes. In other words, the value of $pos_C(v)$ is completely determined by this single node μ. We show that for every vertex $v \in V \setminus V(C)$ there is a node μ in $\mathcal{T}$ containing C as a cycle such that the virtual edge in $skel(\mu)$ containing v in its expansion graph is not contained in the cycle κ induced by C. Hence such a node $\mu \in \mathcal{T}$ determining $pos_C(v)$ always exists. Extending this to a pair of cycles yields the following lemma.

Lemma 2. *Let G be a biconnected planar graph with SPQR-tree $\mathcal{T}$ and let C_1 and C_2 be two disjoint simple cycles in G. There is exactly one node μ in $\mathcal{T}$ determining $pos_{C_1}(C_2)$. Moreover, μ contains C_1 as cycle κ_1 and C_2 either as a cycle or contracted in an edge not contained in κ_1.*

Proof. We choose some vertex $v \in V(C_2)$ as representative for the whole cycle. Consider a Q-node μ_1 in the SPQR-tree $\mathcal{T}$ corresponding to an edge contained in C_1. Moreover, let μ_k be a Q-node corresponding to an edge incident to v. We claim that the desired node μ lies somewhere on the path $\mu_1, \ldots, \mu_k$ in the SPQR-tree $\mathcal{T}$.

Obviously C_1 is a cycle in μ_1 and the vertex v belongs to the virtual edge in $skel(\mu_1)$. In μ_k the vertex v is a pole and C_1 is contracted in the virtual edge of $skel(\mu_k)$ since $v \notin V(C_1)$. Assume we are navigating from μ_1 to μ_k and let μ_i be the current node. If $skel(\mu_i)$ does not contain the vertex v, it belongs to a single virtual edge in $skel(\mu_i)$. In this case μ_{i+1} is obviously the node corresponding to this virtual edge. If v is a vertex of $skel(\mu_i)$, then μ_{i+1} corresponds to one of the virtual edges incident to v in $skel(\mu_i)$. As long as C_1 is a cycle in the current node and v belongs to a virtual edge in this cycle, the next node in the path corresponds to this virtual edge and thus C_1 remains a cycle. Since C_1 is contracted in μ_k, we somewhere need to follow a virtual edge not contained in the cycle induced by C_1; let μ be this node. By definition μ contains C_1 as cycle κ and the next node on the path belongs to a virtual edge that is not contained in κ but contains v in its expansion graph. Thus $pos_{C_1}(v)$ is determined by this node μ. Since v is a node of the second cycle C_2 also $pos_{C_1}(C_2)$ is completely determined by this node. Moreover, μ contains C_1 as cycle κ and C_2 either as a cycle or contracted in a virtual edge not belonging to κ. $\square$

Now consider a set of pairwise disjoint cycles $\mathcal{C} = \{C_1, \ldots, C_k\}$ in G. Let μ be an arbitrary node in the SPQR-tree $\mathcal{T}$. If μ is an S- or a Q-node it clearly does not determine any of the relative positions since either every cycle is contracted in $skel(\mu)$ or a single cycle is a cycle in $skel(\mu)$ containing all the virtual edges. In the following, we consider the two interesting cases namely that μ is an R- or a P-node containing at least one cycle as a cycle.

Let μ be a **P-node** in $\mathcal{T}$ with $skel(\mu)$ consisting of two vertices s and t with parallel virtual edges $\varepsilon_1, \ldots, \varepsilon_\ell$ between them. If $C \in \mathcal{C}$ is contained as a cycle in $skel(\mu)$, it induces a cycle κ in $skel(\mu)$ consisting of two of the parallel virtual edges. Let without loss of generality ε_1 and ε_2 be these virtual edges. Obviously, no other cycle $C' \in \mathcal{C}$ is a cycle in $skel(\mu)$ since such a cycle would need to contain s and t, which is a contradiction to the assumption that C and C' are disjoint. Thus, every other cycle C' is contracted in $skel(\mu)$, belonging to one of the virtual edges $\varepsilon_1, \ldots, \varepsilon_\ell$. If it belongs to ε_1 or ε_2, which are contained in κ, then $pos_C(C')$ is not determined by μ. If C' belongs to one of the virtual edges $\varepsilon_3, \ldots, \varepsilon_\ell$, the relative position $pos_C(C')$ is determined by the relative position of this virtual edge with respect to the cycle κ. This relative position can be chosen for every virtual edge $\varepsilon_3, \ldots, \varepsilon_\ell$ arbitrarily and independently. Hence, if there are two cycles C_i and C_j belonging to different virtual edges in μ, the positions $pos_C(C_i)$ and $pos_C(C_j)$ can be chosen independently. Furthermore, if the two cycles C_i and C_j belong to the same virtual edge $\varepsilon \in \{\varepsilon_3, \ldots, \varepsilon_\ell\}$, their relative position with respect to C is the same, that is $pos_C(C_i) = pos_C(C_j)$, for every embedding of G.

Let μ be an **R-node** in $\mathcal{T}$. For the moment, we consider that the embedding of $skel(\mu)$ is fixed by choosing one of the two orientations. Let C be a cycle inducing the cycle κ in $skel(\mu)$. Then the relative position $pos_C(C')$ of a cycle $C' \neq C$ is determined by μ if and only if C' is a cycle in $skel(\mu)$ or if it is contracted belonging to a virtual edge not contained in κ. Since we consider only one of the two embeddings of $skel(\mu)$ at the moment, $pos_C(C')$ is fixed to one of the two values "left" or "right" in this case. The same can be done for all other cycles that are cycles in $skel(\mu)$ yielding a fixed value for all relative positions that are determined by μ. Finally, we have a partition of all positions determined by μ into the set of positions $\mathcal{P}_1 = \{pos_{C_{a(1)}}(C_{b(1)}), \ldots, pos_{C_{a(r)}}(C_{b(r)})\}$ all having the value "left" and the set of positions $\mathcal{P}_2 = \{pos_{C_{c(1)}}(C_{d(1)}), \ldots, pos_{C_{c(s)}}(C_{d(s)})\}$ having the value "right". Now if the embedding of $skel(\mu)$ is not fixed anymore, we have only the possibility to flip it. By flipping, all the positions in $\mathcal{P}_1$ change to "right" and all positions in $\mathcal{P}_2$ change to "left". Hence, we obtain that the equation $pos_{C_{a(1)}}(C_{b(1)}) = \cdots = pos_{C_{a(r)}}(C_{b(r)}) \neq pos_{C_{c(1)}}(C_{d(1)}) = \cdots = pos_{C_{c(s)}}(C_{d(s)})$ is satisfied for every embedding of the cycles $\mathcal{C} = \{C_1, \ldots, C_k\}$ induced by an embedding of G.

To sum up, we obtain a set of (in)equalities relating the relative positions of cycles to one another. We call these constraints the *PR-node constraints* with respect to the biconnected graph *G*. Obviously the PR-node constraints are necessary in the sense that every embedding of *G* induces an embedding of the cycles $\mathcal{C} = \{C_1, \ldots, C_k\}$ satisfying these constraints. The following lemma additionally states the sufficiency of the PR-node constraints.

Lemma 3. *Let G be a biconnected planar graph containing the disjoint cycles $\mathcal{C} = \{C_1, \ldots, C_k\}$. Let further $\mathcal{E}_\mathcal{C}$ be a semi-embedding of these cycles. There is an embedding $\mathcal{E}$ of G inducing $\mathcal{E}_\mathcal{C}$ if and only if $\mathcal{E}_\mathcal{C}$ satisfies the PR-node constraints.*

Proof. The "only if"-part of the proof is obvious, as mentioned above. It remains to show the "if"-part. Let $\mathcal{E}_\mathcal{C}$ be a semi-embedding of $\mathcal{C} = \{C_1, \ldots, C_k\}$ satisfying the PR-node constraints given by *G*. We show how to construct an embedding $\mathcal{E}$ of *G* inducing the embedding $\mathcal{E}_\mathcal{C}$ on the cycles $\mathcal{C} = \{C_1, \ldots, C_k\}$. We simply process the nodes of the SPQR-tree one by one and choose an embedding for the skeleton of every node. Let μ be a node in $\mathcal{T}$. If μ is an S- or a Q-node, there is nothing to do, since there is no choice for the embedding of skel(μ). If μ is a P-node several relative positions may be determined by the embedding of skel(μ). However, these positions satisfy the PR-node constraints stemming from μ, hence we can choose an embedding for skel(μ) determining these positions as given by $\mathcal{E}_\mathcal{C}$. Obviously, the same holds for the case where μ is an R-node. Hence, we finally obtain an embedding $\mathcal{E}$ of *G* determining the positions that are determined by a node in $\mathcal{T}$ as required by $\mathcal{E}_\mathcal{C}$. Due to Lemma 2 every pair of relative positions is determined by exactly one node in $\mathcal{T}$, yielding that the resulting embedding $\mathcal{E}$ induces $\mathcal{E}_\mathcal{C}$ on the cycles. Note that this shows implicitly that $\mathcal{E}_\mathcal{C}$ is not only a semi-embedding but also an embedding. □

Now let $G^{①}$ and $G^{②}$ be two biconnected planar graphs with the common graph *G* consisting of pairwise disjoint simple cycles $\mathcal{C} = \{C_1, \ldots, C_k\}$. If we find a semi-embedding $\mathcal{E}$ of the cycles that satisfies the PR-node constraints with respect to $G^{①}$ and $G^{②}$ simultaneously, we can use Lemma 3 to find embeddings $\mathcal{E}^{①}$ and $\mathcal{E}^{②}$ for $G^{①}$ and $G^{②}$ both inducing the embedding $\mathcal{E}$ on the common graph *G*. Thus, satisfying the PR-node constraints with respect to both $G^{①}$ and $G^{②}$, is sufficient to find a SEFE. Conversely, given a pair of embeddings $\mathcal{E}^{①}$ and $\mathcal{E}^{②}$ inducing the same embedding $\mathcal{E}$ on *G*, this embedding $\mathcal{E}$ needs to satisfy the PR-node constraints with respect to both, $G^{①}$ and $G^{②}$, which is again due to Lemma 3. Since the PR-node constraints form a set of boolean (in)equalities we can express them as an instance of 2-SAT. As this instance has polynomial size and can easily be computed in polynomial time, we obtain the following theorem.

Theorem 2. SIMULTANEOUS EMBEDDING WITH FIXED EDGES *can be solved in quadratic time for biconnected graphs whose common graph is a set of disjoint cycles.*

Proof. It remains to show that the PR-node constraints can be computed in quadratic time, yielding an instance of 2-SAT with quadratic size. As this 2-SAT instance can be solved consuming time linear in its size [15,16], we obtain a quadratic-time algorithm.

We show how to process each node μ of the SPQR-tree in $\mathcal{O}(n \cdot |\text{skel}(\mu)|)$ time, computing the PR-node constraints stemming from μ. For each virtual edge ε we compute a list of cycles in $\mathcal{C}$ that contain edges in the expansion graph $\exp(\varepsilon)$ by traversing all leaves in the corresponding subtree, consuming $\mathcal{O}(n)$ time for each virtual edge. Then the list of cycles that occur as cycles in skel(μ) can be computed in linear time. For each of these cycles *C* all constraints on relative positions with respect to *C* determined by μ can be easily computed in $\mathcal{O}(n)$ time. As only $\mathcal{O}(|\text{skel}(\mu)|)$ cycles can be contained as cycles in skel(μ), this yields the claimed $\mathcal{O}(n \cdot |\text{skel}(\mu)|)$ time for each skeleton. Since the total size of the skeletons is linear in the size of the graph, this yields an overall $\mathcal{O}(n^2)$-time algorithm. □

Allowing cutvertices

In this section we consider the case where the graphs may contain cutvertices. As before, we consider a single graph *G* containing a set of disjoint cycles $\mathcal{C} = \{C_1, \ldots, C_k\}$ first. Let $C \in \mathcal{C}$ be one of the cycles and let *v* be a cutvertex contained in the same block *B* that contains *C*. The cutvertex *v* splits *G* into ℓ cut components $H_1, \ldots, H_\ell$. Assume without loss of generality that *B* (and with it also *C*) is contained in H_1. We distinguish between the cases that *v* is contained in *C* and that it is not.

If ***v* is not contained in *C***, then the relative position $\text{pos}_C(v)$ is determined by the embedding of the block *B* and it follows that all the subgraphs $H_2, \ldots, H_\ell$ lie on the same side of *C* as *v* does. It follows from the biconnected case (Lemma 2) that $\text{pos}_C(v)$ is determined by the embedding of the skeleton of exactly one node μ in the SPQR-tree of *B*. Obviously, the conditions that all cycles in $H_2, \ldots, H_\ell$ are on the same side of *C* as *v* can be easily added to the PR-node constraints stemming from the node μ; call the resulting constraints the *extended PR-node constraints*. These constraints are clearly necessary. On the other hand, if $\mathcal{E}_\mathcal{C}$ is a semi-embedding of the cycles satisfying the extended PR-node constraints, we can find an embedding $\mathcal{E}_B$ of the block *B* such that all relative positions of cycles that are determined by single nodes in the SPQR-tree of *B* are compatible with $\mathcal{E}_\mathcal{C}$.

If ***v* is contained in *C***, the relative position $\text{pos}_C(v)$ does not exist. Assume the embedding of each block is already chosen. Then for each subgraph $H \in \{H_2, \ldots, H_\ell\}$, the positions $\text{pos}_C(H)$ can be chosen arbitrarily and independently. In this case we say for a cycle C' in *H* that its relative position $\text{pos}_C(C')$ is *determined by the embedding chosen for the cutvertex v*.

$pos_C(v) =$ "right"　　　　$pos_C(v) =$ "left"　　　　$pos_C(v) =$ "right"　　　　$pos_C(v) =$ "left"
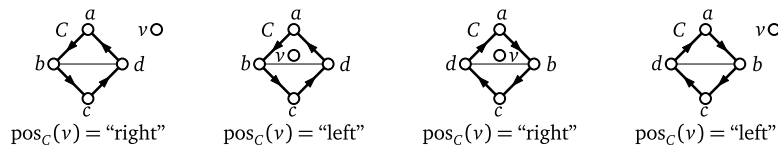
Fig. 5. One component containing C (bold) and another consisting only of the vertex v. Changing the face in which v lies may change the relative position $pos_C(v)$. Moreover, changing the embedding of the component containing C (in this case flipping it) also changes $pos_C(v)$.

Obviously, in every embedding of G, a pair of cycles C_i and C_j both belonging to the same subgraph $H \in \{H_2, \ldots, H_\ell\}$ lie on the same side of C yielding the equation $pos_C(C_i) = pos_C(C_j)$. This equation can be set up for every pair of cycles in each of the subgraphs, yielding the *cutvertex constraints* with respect to v. Again we have that, given a semi-embedding $\mathcal{E}_\mathcal{C}$ of the cycles satisfying the cutvertex constraints with respect to v, we can simply choose an embedding of the graph such that the relative positions determined by the embedding around the cutvertex are compatible with $\mathcal{E}_\mathcal{C}$.

To sum up, a semi-embedding $\mathcal{E}_\mathcal{C}$ on the cycles $\mathcal{C} = \{C_1, \ldots, C_k\}$ that is induced by an embedding $\mathcal{E}$ of the whole graph always satisfies the extended PR-node and cutvertex constraints. Moreover, given a semi-embedding $\mathcal{E}_\mathcal{C}$ satisfying these constraints, we can find an embedding $\mathcal{E}$ of G inducing compatible relative positions for each relative position that is determined by a single node in the SPQR-tree of a block or by a cutvertex. Obviously, the relative position of every pair of cycles is determined by such a node or a cutvertex. Thus the extended PR-node and cutvertex constraints together are sufficient, that is, given a semi-embedding of the cycles satisfying these constraints, we can find an embedding of G inducing this semi-embedding. This shows implicitly that the given semi-embedding is an embedding. This result is stated again in the following lemma.

Lemma 4. *Let G be a connected planar graph containing the disjoint cycles $\mathcal{C} = \{C_1, \ldots, C_k\}$. Let further $\mathcal{E}_\mathcal{C}$ be a semi-embedding of these cycles. There is an embedding $\mathcal{E}$ of G inducing $\mathcal{E}_\mathcal{C}$ if and only if $\mathcal{E}_\mathcal{C}$ satisfies the extended PR-node and cutvertex constraints with respect to G.*

This result again directly yields a polynomial-time algorithm to solve SEFE for the case that both graphs $G^{①}$ and $G^{②}$ are connected and their common graph G consists of a set of disjoint cycles. Moreover, requiring both graphs to be connected is not really a restriction due to Theorem 1. The extended PR-node and cutvertex constraints can be computed similarly as in the proof of Theorem 2, yielding the following theorem.

Theorem 3. Simultaneous Embedding with Fixed Edges *can be solved in quadratic time if the common graph consists of disjoint cycles.*

Note that we really need to use Theorem 1 to ensure that the graphs are connected since our approach does not extend to the case where the graphs are allowed to be disconnected. In this case it would still be easy to formulate necessary conditions in terms of boolean equations. However, these conditions would only be sufficient if it is additionally ensured that the given semi-embedding actually is an embedding. The reason why this is not directly ensured by the embedding of the graph (as it is in the connected case) is that the relative position of cycles to one another is not determined by exactly one choice that can be made independently from the other choices; see Fig. 5.

3.2. A compact representation of all simultaneous embeddings

In the previous section we showed that SEFE can be solved in polynomial time for the case that the common graph consists of disjoint cycles. In this section we describe a data structure, the *CC-tree*, representing all embeddings of a set of disjoint cycles that can be induced by an embedding of a connected graph containing them. Afterwards, we show that the intersection of the sets of embeddings represented by two CC-trees can again be represented by a CC-tree. In Section 3.3 we then show that the CC-tree and the intersection of two CC-trees can be computed in linear time, yielding an optimal linear-time algorithm for SEFE for the case that the common graph consists of disjoint cycles. Note that this algorithm obviously extends to the case where k graphs $G^{①}, \ldots, G^{⑱}$ are given such that they all intersect in the same common graph G consisting of a set of disjoint cycles.

C-trees and CC-trees

Let $\mathcal{C} = \{C_1, \ldots, C_k\}$ be a set of disjoint cycles. A *cycle-tree (C-tree)* $\mathcal{T}_\mathcal{C}$ on these cycles is a minimal connected graph containing $\mathcal{C}$; see Fig. 6. Obviously, every embedding of $\mathcal{T}_\mathcal{C}$ induces an embedding of the cycles. We say that two embeddings of $\mathcal{T}_\mathcal{C}$ are equivalent if they induce the same embedding of $\mathcal{C}$ and we are only interested in the equivalence classes with respect to this equivalence relation. An embedding $\mathcal{E}$ of the cycles in $\mathcal{C}$ is *represented* by $\mathcal{T}_\mathcal{C}$ if it admits an embedding inducing $\mathcal{E}$. Note that contracting each of the cycles $\mathcal{C} = \{C_1, \ldots, C_k\}$ in a C-tree to a single vertex yields a spanning tree on these vertices. In most cases we implicitly assume the cycles to be contracted such that $\mathcal{T}_\mathcal{C}$ can be treated like a tree.
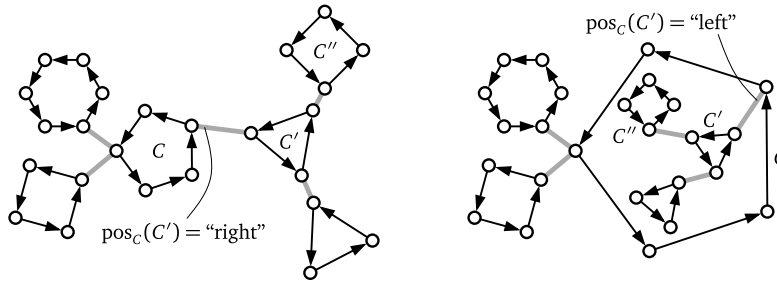
Fig. 6. Two embeddings of the same CC-tree. The only difference between the embeddings is that different values are chosen for the crucial relative position $\text{pos}_C(C')$. Note that the tree structure enforces the (non-crucial) relative position $\text{pos}_C(C'')$ to be equal to $\text{pos}_C(C')$.

The embedding choices that can be made for $\mathcal{T}_{\mathcal{C}}$ are of the following kind. For every edge $e = \{C, C'\}$ in $\mathcal{T}_{\mathcal{C}}$, we can decide to put all cycles in the subtree attached to C via e either to the left or to the right of C. In particular, we can assign a value "left" or "right" to the relative position $\text{pos}_C(C')$. Moreover, by fixing the relative positions $\text{pos}_C(C')$ and $\text{pos}_{C'}(C)$ for every pair of cycles C and C' that are adjacent in $\mathcal{T}_{\mathcal{C}}$, the embedding represented by $\mathcal{T}_{\mathcal{C}}$ is completely determined. Thus, given a C-tree $\mathcal{T}_{\mathcal{C}}$, we call the relative positions $\text{pos}_C(C')$ and $\text{pos}_{C'}(C)$ with $C, C' \in \mathcal{C}$ *crucial* if C and C' are adjacent in $\mathcal{T}_{\mathcal{C}}$; see Fig. 6. We note that, when determining an embedding of $\mathcal{T}_{\mathcal{C}}$, the crucial relative positions can be chosen independently from one another.

Since the crucial relative positions with respect to a C-tree $\mathcal{T}_{\mathcal{C}}$ are binary variables, we can use (in)equalities between them to further constrain the embeddings represented by $\mathcal{T}_{\mathcal{C}}$. We call a C-tree with such additional constraints on its crucial relative positions a *constrained cycle-tree (CC-tree)* on the set of cycles $\mathcal{C}$. In this way, there is a bijection between the embeddings of $\mathcal{C}$ represented by a CC-tree and the solutions of an instance of 2-Sat given by the constraints on the crucial relative positions of $\mathcal{T}_{\mathcal{C}}$. We essentially prove two things. First, for every connected graph G containing the cycles $\mathcal{C}$, there exists a CC-tree representing exactly the embeddings of $\mathcal{C}$ that can be induced by embeddings of G. Essentially, we have to restrict the extended PR-node and cutvertex constraints to the crucial relative positions of a C-tree compatible with G. Second, for a pair of CC-trees $\mathcal{T}_{\mathcal{C}}^{\textcircled{1}}$ and $\mathcal{T}_{\mathcal{C}}^{\textcircled{2}}$ on the same set $\mathcal{C}$ of cycles, there exists a CC-tree $\mathcal{T}_{\mathcal{C}}$ representing exactly the embeddings of $\mathcal{C}$ that are represented by $\mathcal{T}_{\mathcal{C}}^{\textcircled{1}}$ and $\mathcal{T}_{\mathcal{C}}^{\textcircled{2}}$.

Let G be a connected planar graph containing a set $\mathcal{C}$ of disjoint cycles. We say that a C-tree $\mathcal{T}_{\mathcal{C}}$ is *compatible* with G if it is a minor of G, that is if it can be obtained by contracting edges in a subgraph of G. The corresponding *compatible CC-tree* is obtained from $\mathcal{T}_{\mathcal{C}}$ by adding the subset of the extended PR-node and cutvertex constraints that only involve crucial relative positions of $\mathcal{T}_{\mathcal{C}}$. Note that there may be many compatible CC-trees for a single graph G. However, in the following we arbitrarily fix one of them and speak about *the* CC-tree of G.

Theorem 4. *Let G be a connected planar graph containing the disjoint cycles $\mathcal{C} = \{C_1, \ldots, C_k\}$. The CC-tree $\mathcal{T}_{\mathcal{C}}$ of G represents exactly the embeddings of $\mathcal{C}$ that can be induced by an embedding of G.*

Proof. Let $\mathcal{E}$ be an embedding of G and let $\mathcal{E}_{\mathcal{C}}$ be the embedding induced on the cycles $\mathcal{C} = \{C_1, \ldots, C_k\}$. Obviously, the CC-tree $\mathcal{T}_{\mathcal{C}}$ can be obtained from G by contracting the cycles $\mathcal{C}$ to single vertices, choosing a spanning tree, expanding the cycles and contracting edges incident to non-cycle vertices. Since we essentially only pick a subgraph of G containing all cycles $\mathcal{C} = \{C_1, \ldots, C_k\}$ and contract edges, the embedding $\mathcal{E}_{\mathcal{C}}$ is preserved. Moreover, by Lemma 4, it satisfies the extended PR-node and cutvertex constraints since it is induced by the embedding $\mathcal{E}$ of G. Hence, $\mathcal{E}_{\mathcal{C}}$ is represented by the CC-tree $\mathcal{T}_{\mathcal{C}}$.

Conversely, let $\mathcal{E}_{\mathcal{C}}$ be an embedding on the cycles represented by the CC-tree $\mathcal{T}_{\mathcal{C}}$. By definition, the extended PR-node and cutvertex constraints are satisfied for the crucial relative positions. We show that the tree-like structure of $\mathcal{T}_{\mathcal{C}}$ ensures that they are also satisfied for the remaining relative positions, yielding that an embedding $\mathcal{E}$ of G inducing $\mathcal{E}_{\mathcal{C}}$ exists due to Lemma 4. We start with the PR-node constraints. Let B be a block of G with SPQR-tree $\mathcal{T}(B)$. In a P-node μ containing a cycle C as cycle κ every other cycle in B is contracted, belonging to a single virtual edge. Let C_i and C_j be two cycles in B belonging to the same virtual edge ε not contained in κ. In this case the PR-node constraints stemming from μ require $\text{pos}_C(C_i) = \text{pos}_C(C_j)$, and we show that this equation is implied if the extended PR-node constraints are satisfied for the crucial relative positions. Let C_i' and C_j' be the first cycles on the paths from C to C_i and C_j in $\mathcal{T}_{\mathcal{C}}$, respectively. Note that C_i' and C_j' are not necessarily contained in the block B. However, we first consider the case where both are contained in B. Then C_i' and C_j' are both contracted in the same virtual edge ε as C_i and C_j since a path from a cycle belonging to ε to a cycle belonging to a different virtual edge would necessarily contain a pole of $\text{skel}(\mu)$ and thus a vertex in C. Thus, the PR-node constraints restricted to the crucial relative positions enforce $\text{pos}_C(C_i') = \text{pos}_C(C_j')$. Furthermore, the tree structure of $\mathcal{T}_{\mathcal{C}}$ enforces $\text{pos}_C(C_i) = \text{pos}_C(C_i')$ and $\text{pos}_C(C_j) = \text{pos}_C(C_j')$. Hence, in this case the PR-node constraints stemming from μ are implied by their restriction to the crucial relative positions. For the case that C_i' or C_j' are contained in a different block,

they are connected to B via cutvertices v_i or v_j, which must belong to $\exp(\varepsilon)$ by the same argument as above, namely that every path from C_i or C_j to a vertex that is contained in the expansion graph of another virtual edge needs to contain one of the poles. Thus, the extended PR-node constraints enforce $\mathrm{pos}_C(C_i') = \mathrm{pos}_C(C_j')$ yielding the same situation as above. In total, the extended PR-node constraints stemming from a P-node μ restricted to the crucial relative positions enforce that the PR-node constraints stemming from μ are satisfied for all relative positions.

For the case that μ is an R-node a similar argument holds. If C is a cycle κ in $\mathrm{skel}(\mu)$ and two cycles C_i and C_j lie contracted or as cycles on the same side (on different sides) of κ, then the first cycles C_i' and C_j' on the path from C to C_i and C_j in the CC-tree $\mathcal{T}_C$ lie on the same side (on different sides) of κ or the cutvertices connecting C_i' and C_j' to the block B lie on the same side (on different sides) of κ. Thus, the extended PR-node constraints restricted to the crucial relative positions enforce $\mathrm{pos}_C(C_i') = \mathrm{pos}_C(C_j')$ ($\mathrm{pos}_C(C_i') \neq \mathrm{pos}_C(C_j')$) and the tree structure of $\mathcal{T}_C$ yields $\mathrm{pos}_C(C_i) = \mathrm{pos}_C(C_i')$ and $\mathrm{pos}_C(C_j) = \mathrm{pos}_C(C_j')$. Obviously, these arguments extend to the case of extended PR-node constraints since a cutvertex not contained in C can be treated like a disjoint cycle.

It remains to deal with the cutvertex constraints stemming from the case where C is a cycle containing a cutvertex v splitting G into the cut components $H_1, \ldots, H_\ell$. Let without loss of generality H_1 be the subgraph containing C. The cutvertex constraints ensure that a pair of cycles C_i and C_j belonging to the same subgraph $H \in \{H_2, \ldots, H_\ell\}$ are located on the same side of C. Let C_i' and C_j' be the first cycles on the path from C to C_i and C_j in the CC-tree $\mathcal{T}_C$, respectively. Obviously C_i' and C_j' belong to the same subgraph H and hence the cutvertex constraints restricted to the crucial relative positions enforce $\mathrm{pos}_C(C_i') = \mathrm{pos}_C(C_j')$. Moreover, the tree structure of $\mathcal{T}_C$ again ensures that the equations $\mathrm{pos}_C(C_i) = \mathrm{pos}_C(C_i')$ and $\mathrm{pos}_C(C_j) = \mathrm{pos}_C(C_j')$ hold, which concludes the proof. $\quad\square$

Intersecting CC-trees

In this section we consider two CC-trees $\mathcal{T}_C^{\textcircled{1}}$ and $\mathcal{T}_C^{\textcircled{2}}$ on the same set of cycles $\mathcal{C}$. We show that the set of embeddings that are represented by both $\mathcal{T}_C^{\textcircled{1}}$ and $\mathcal{T}_C^{\textcircled{2}}$ can again be represented by a single CC-tree. We will show this by constructing a new CC-tree, which we call the *intersection* of $\mathcal{T}_C^{\textcircled{1}}$ and $\mathcal{T}_C^{\textcircled{2}}$, showing afterwards that this CC-tree has the desired property. The intersection $\mathcal{T}_C$ is a copy of $\mathcal{T}_C^{\textcircled{1}}$ with some additional constraints given by the second CC-tree $\mathcal{T}_C^{\textcircled{2}}$. We essentially have to formulate two types of constraints. First, constraints stemming from the structure of the underlying C-tree of $\mathcal{T}_C^{\textcircled{2}}$. Second, the constraints given by the (in)equalities on the relative positions that are crucial with respect to $\mathcal{T}_C^{\textcircled{2}}$. We show that both kinds of constraints can be formulated as (in)equalities on the relative positions that are crucial with respect to $\mathcal{T}_C^{\textcircled{1}}$.

Let C_1 and C_2 be two cycles joined by an edge in $\mathcal{T}_C^{\textcircled{2}}$. Obviously, C_1 and C_2 are contained in the boundary of a common face in every embedding $\mathcal{E}^{\textcircled{2}}$ represented by $\mathcal{T}_C^{\textcircled{2}}$. It is easy to formulate constraints on the relative positions that are crucial with respect to $\mathcal{T}_C^{\textcircled{1}}$ such that C_1 and C_2 are contained in the boundary of a common face for every embedding represented by $\mathcal{T}_C^{\textcircled{1}}$. Consider the path π from C_1 to C_2 in $\mathcal{T}_C^{\textcircled{1}}$. For every three cycles C, C' and C'' appearing consecutively on π it is necessary that $\mathrm{pos}_{C'}(C) = \mathrm{pos}_{C'}(C'')$ holds. Otherwise C_1 and C_2 would be separated by C'. Conversely, if this equation holds for every triple of consecutive cycles on π, then C_1 and C_2 always lie on a common face. We call the resulting equations the *common-face constraints*. Note that all relative positions involved in such constraints are crucial with respect to $\mathcal{T}_C^{\textcircled{1}}$.

To formulate the constraints given on the crucial relative positions of $\mathcal{T}_C^{\textcircled{2}}$, we essentially find, for each of these crucial relative positions $\mathrm{pos}_{C_1}(C_2)$, a relative position $\mathrm{pos}_{C_1}(C_2')$ that is crucial with respect to $\mathcal{T}_C^{\textcircled{1}}$ such that $\mathrm{pos}_{C_1}(C_2)$ is determined by fixing $\mathrm{pos}_{C_1}(C_2')$ in $\mathcal{T}_C^{\textcircled{1}}$. More precisely, for every relative position $\mathrm{pos}_{C_1}(C_2)$ that is crucial with respect to $\mathcal{T}_C^{\textcircled{2}}$ we define its *representative* in $\mathcal{T}_C^{\textcircled{1}}$ to be the crucial relative position $\mathrm{pos}_{C_1}(C_2')$, where C_2' is the first cycle in $\mathcal{T}_C^{\textcircled{1}}$ on the path from C_1 to C_2. We obtain the *crucial-position constraints* on the crucial relative positions of $\mathcal{T}_C^{\textcircled{1}}$ by replacing every relative position in the constraints given for $\mathcal{T}_C^{\textcircled{2}}$ by its representative. The resulting set of (in)equalities on the crucial relative positions of $\mathcal{T}_C^{\textcircled{1}}$ is obviously necessary.

We can now formally define the *intersection* $\mathcal{T}_C$ of two CC-trees $\mathcal{T}_C^{\textcircled{1}}$ and $\mathcal{T}_C^{\textcircled{2}}$ to be $\mathcal{T}_C^{\textcircled{1}}$ with the common-face and crucial-position constraints additionally restricting its crucial relative positions. We obtain the following theorem, justifying the name "intersection".

Theorem 5. *The intersection of two CC-trees represents exactly the embeddings that are represented by both CC-trees.*

Proof. Let $\mathcal{T}_C^{\textcircled{1}}$ and $\mathcal{T}_C^{\textcircled{2}}$ be two CC-trees and let $\mathcal{T}_C$ be their intersection. Let further $\mathcal{E}$ be an embedding represented by $\mathcal{T}_C^{\textcircled{1}}$ and $\mathcal{T}_C^{\textcircled{2}}$. Then $\mathcal{T}_C$ also represents $\mathcal{E}$ since the common-face and crucial-position constraints are obviously necessary.
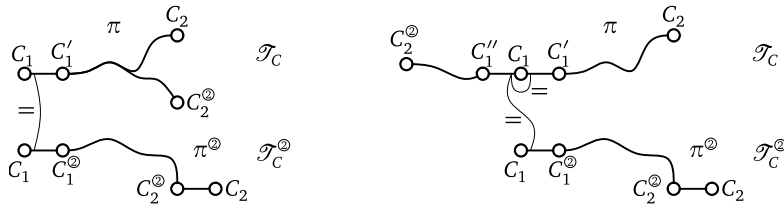
Fig. 7. The two cases arising in the proof of Theorem 5. If the path from C_1 to $C_2^{\textcircled{2}}$ starts with the edge $\{C_1, C_1'\}$ (left) the equation $\mathrm{pos}_{C_1}(C_1') = \mathrm{pos}_{C_1}^{\textcircled{2}}(C_1^{\textcircled{2}})$ follows by induction. Otherwise (right) $\mathrm{pos}_{C_1}(C_1'') = \mathrm{pos}_{C_1}^{\textcircled{2}}(C_1^{\textcircled{2}})$ follows by induction and the equation $\mathrm{pos}_{C_1}(C_1') = \mathrm{pos}_{C_1}(C_1'')$ holds due to the common-face constraint stemming from $\{C_2, C_2^{\textcircled{2}}\}$.

Now let $\mathcal{E}$ be an embedding represented by $\mathcal{T}_C$. It is clearly also represented by $\mathcal{T}_C^{\textcircled{1}}$ since $\mathcal{T}_C$ is the same tree with some additional constraints. It remains to show that $\mathcal{E}$ is represented by $\mathcal{T}_C^{\textcircled{2}}$. The embedding $\mathcal{E}$ induces a value for every relative position. In particular, it induces a value for every relative position that is crucial with respect to $\mathcal{T}_C^{\textcircled{2}}$. The crucial-position constraints ensure that these values satisfy the constraints given for the crucial relative positions in the CC-tree $\mathcal{T}_C^{\textcircled{2}}$. Thus we can simply take these positions, apply them to $\mathcal{T}_C^{\textcircled{2}}$ and obtain an embedding $\mathcal{E}^{\textcircled{2}}$ that is represented by $\mathcal{T}_C^{\textcircled{2}}$. It remains to show that $\mathcal{E} = \mathcal{E}^{\textcircled{2}}$. To this end, we consider an arbitrary pair of cycles C_1 and C_2 and show the following equation, where $\mathrm{pos}_{C_1}(C_2)$ and $\mathrm{pos}_{C_1}^{\textcircled{2}}(C_2)$ denote the relative positions of C_2 with respect to C_1 in the embeddings $\mathcal{E}$ and $\mathcal{E}^{\textcircled{2}}$, respectively.

$$\mathrm{pos}_{C_1}(C_2) = \mathrm{pos}_{C_1}^{\textcircled{2}}(C_2) \tag{1}$$

Consider the paths π and $\pi^{\textcircled{2}}$ from C_1 to C_2 in $\mathcal{T}_C$ and $\mathcal{T}_C^{\textcircled{2}}$, respectively. We use induction on the length of $\pi^{\textcircled{2}}$, illustrated in Fig. 7, with Eq. (1) as induction hypothesis. If $|\pi^{\textcircled{2}}| = 1$, then $\mathrm{pos}_{C_1}(C_2)$ is crucial with respect to $\mathcal{T}_C^{\textcircled{2}}$ and thus equal in both embeddings $\mathcal{E}$ and $\mathcal{E}^{\textcircled{2}}$ by construction of $\mathcal{E}^{\textcircled{2}}$. For the case $|\pi^{\textcircled{2}}| > 1$ let C_1' and $C_1^{\textcircled{2}}$ be the neighbors of C_1 in π and $\pi^{\textcircled{2}}$, respectively. Since C_1' and $C_1^{\textcircled{2}}$ lie on the path between C_1 and C_2 in $\mathcal{T}_C$ and $\mathcal{T}_C^{\textcircled{2}}$, the following two equations hold.

$$\mathrm{pos}_{C_1}(C_2) = \mathrm{pos}_{C_1}(C_1') \tag{2}$$

$$\mathrm{pos}_{C_1}^{\textcircled{2}}(C_2) = \mathrm{pos}_{C_1}^{\textcircled{2}}(C_1^{\textcircled{2}}) \tag{3}$$

Thus, it suffices to show that $\mathrm{pos}_{C_1}(C_1') = \mathrm{pos}_{C_1}^{\textcircled{2}}(C_1^{\textcircled{2}})$ holds to obtain Eq. (1). Let $C_2^{\textcircled{2}}$ be the neighbor of C_2 on the path $\pi^{\textcircled{2}}$. Since the path from C_1 to $C_2^{\textcircled{2}}$ is shorter than $\pi^{\textcircled{2}}$ the equation $\mathrm{pos}_{C_1}(C_2^{\textcircled{2}}) = \mathrm{pos}_{C_1}^{\textcircled{2}}(C_2^{\textcircled{2}})$ follows from the induction hypothesis stated in Eq. (1). There are two possibilities. The path from C_1 to $C_2^{\textcircled{2}}$ in $\mathcal{T}_C$ has either $\{C_1, C_1'\}$ or $\{C_1, C_1''\}$ for some other cycle C_1'' as first edge. In the former case the equation

$$\mathrm{pos}_{C_1}(C_1') = \mathrm{pos}_{C_1}^{\textcircled{2}}(C_1^{\textcircled{2}}) \tag{4}$$

obviously follows. Together with Eqs. (2) and (3), this yields the induction hypothesis (Eq. (1)). In the latter case we have the following equation.

$$\mathrm{pos}_{C_1}(C_1'') = \mathrm{pos}_{C_1}^{\textcircled{2}}(C_1^{\textcircled{2}}) \tag{5}$$

Moreover, the common-face constraints stemming from the edge $\{C_2^{\textcircled{2}}, C_2\}$ in $\mathcal{T}_C^{\textcircled{2}}$ enforce

$$\mathrm{pos}_{C_1}(C_1'') = \mathrm{pos}_{C_1}(C_1'), \tag{6}$$

again yielding the induction hypothesis stated in Eq. (1). This concludes the proof. $\quad\square$

3.3. Linear-time algorithm

In this section we first show how to compute the CC-tree of a given graph containing a set of disjoint cycles in linear time. Afterwards, we show that the intersection of two CC-trees can be computed in linear time. Together, this yields a linear-time algorithm for the variant of SEFE we consider.

Table 1

Data that is computed to compute the PR-node constraints restricted to the crucial relative positions.

Data	Description
$\mathrm{cyc}(\mu)$	For a node μ in the SPQR-tree the list of cycles in $\mathcal{C}$ that are cycles in $\mathrm{skel}(\mu)$
$\mathrm{bel}(\varepsilon)$	For a virtual edge ε in $\mathrm{skel}(\mu)$ either a cycle $C \in \mathcal{C}$ if C induces a cycle in $\mathrm{skel}(\mu)$ containing ε or $\bot$ denoting that ε is not contained in such a cycle
$\mathrm{root}(\mathcal{T}\vert_C)$	The root for the induced tree $\mathcal{T}\vert_C$ with respect to a chosen root for the SPQR-tree $\mathcal{T}$
$\mathrm{high}(\mu)$	For a node μ in the SPQR-tree $\mathcal{T}$ the highest edge in $\mathcal{T}$ on the path from μ to the root that is reachable without using an edge in any of the induced subtrees $\mathcal{T}\vert_C$
$\mathrm{det}(\mathrm{pos}_C(C'))$	The node in the SPQR-tree determining the relative position $\mathrm{pos}_C(C')$ of the cycle C' with respect to another cycle C.
$\mathrm{contr}(\varepsilon)$	For a virtual edge ε in $\mathrm{skel}(\mu)$ a list of relative positions containing $\mathrm{pos}_C(C')$ if and only if it is crucial, determined by μ and C' is contracted in ε.
$\mathrm{detcyc}(\mu)$	For every R-node μ a list of crucial relative positions containing $\mathrm{pos}_C(C')$ if and only if C and C' are cycles in $\mathrm{skel}(\mu)$.

Computing the CC-tree in linear time

The first step of computing the CC-tree $\mathcal{T}_{\mathcal{C}}$ of a graph G is to compute the underlying C-tree. Obviously, this can be easily done in linear time. Thus, the focus of this section lies on computing the extended PR-node and cutvertex constraints restricted to the crucial relative positions. To simplify notation we first consider the case where G is biconnected. Before we start computing the PR-node constraints we need one more definition. For each cycle C there is a set of inner nodes in the SPQR-tree $\mathcal{T}$ containing C as a cycle. We denote the subgraph of $\mathcal{T}$ induced by these nodes by $\mathcal{T}\vert_C$ and call it the *induced subtree* with respect to C. To justify the term "subtree" we prove the following lemma.

Lemma 5. *Let G be a biconnected planar graph with SPQR-tree $\mathcal{T}$ containing the disjoint cycles $\mathcal{C} = \{C_1, \ldots, C_k\}$. The induced subtrees $\mathcal{T}\vert_{C_1}, \ldots, \mathcal{T}\vert_{C_k}$ with respect to $C_1, \ldots, C_k$ are pairwise edge-disjoint trees.*

Proof. We first show that the induced tree with respect to a single cycle is really a tree. Afterwards, we show that two disjoint cycles induce edge-disjoint trees, yielding that they have linear size in total.

Let C be a cycle in G and let $\mathcal{T}\vert_C$ be its induced tree. A Q-node in $\mathcal{T}$ contains C as a cycle if and only if the corresponding edge is contained in C. For each pair of these Q-nodes all nodes on the path between them are contained in $\mathcal{T}\vert_C$, thus the Q-nodes are in the same connected component in the induced subtree. Moreover, an internal node in $\mathcal{T}$ cannot be a leaf in $\mathcal{T}\vert_C$, implying that it contains only one connected component.

Assume there are two cycles C_i and C_j inducing trees $\mathcal{T}\vert_{C_i}$ and $\mathcal{T}\vert_{C_j}$ that are not edge-disjoint. Let $\{\mu, \mu'\}$ be an edge in $\mathcal{T}$ belonging to both. Let further κ_i and κ_j be the cycles in $\mathrm{skel}(\mu)$ induced by C_i and C_j, respectively. Since the neighbor μ' of μ also contains C_i as a cycle, it corresponds to a virtual edge ε in μ that is contained in κ_i. Similarly, ε is also contained in κ_j, which is a contradiction since C_i and C_j are disjoint. $\quad\square$

Our algorithm computing the PR-node constraints consists of four phases, each of them consuming linear time. In each phase we compute data we then use in the next phase. Table 1 gives an overview about the data we compute. During all phases we assume the SPQR-tree $\mathcal{T}$ to be rooted at a Q-node corresponding to an edge in G that is not contained in any cycle in $\mathcal{C}$. In the first phase we essentially compute the induced trees $\mathcal{T}\vert_C$. More precisely, for every node μ in the SPQR-tree we compute a list $\mathrm{cyc}(\mu)$ containing a cycle C if and only if C is a cycle in $\mathrm{skel}(\mu)$, that is if and only if μ is contained in $\mathcal{T}\vert_C$. Moreover, we say a virtual edge ε in $\mathrm{skel}(\mu)$ *belongs* to a cycle C if C induces a cycle in $\mathrm{skel}(\mu)$ containing ε. Note that ε belongs to at most one cycle. If ε belongs to C, we set $\mathrm{bel}(\varepsilon) = C$; if ε does not belong to any cycle, we set $\mathrm{bel}(\varepsilon) = \bot$. Finally, the root of an induced tree $\mathcal{T}\vert_C$ with respect to the root chosen for $\mathcal{T}$ is denoted by $\mathrm{root}(\mathcal{T}\vert_C)$. To sum up, in the first phase we compute $\mathrm{cyc}(\mu)$ for every node μ, $\mathrm{bel}(\varepsilon)$ for every virtual edge ε and $\mathrm{root}(\mathcal{T}\vert_C)$ for every induced subtree $\mathcal{T}\vert_C$. In the second phase, we compute $\mathrm{high}(\mu)$ as the highest edge in the SPQR-tree $\mathcal{T}$ on the path from μ to the root whose endpoints are both reachable from μ without using edges contained in any of the induced subtrees $\mathcal{T}\vert_C$. Note that $\mathrm{high}(\mu)$ is the edge in $\mathcal{T}$ incident to the root if no edge on the path from μ to the root is contained in one of the induced subtrees. For the special case that the edge from μ to its parent itself is already contained in one of the induced trees, the edge $\mathrm{high}(\mu)$ is not defined and we set $\mathrm{high}(\mu) = \bot$. In the third phase we compute for every crucial relative position $\mathrm{pos}_C(C')$ the node in the SPQR-tree determining it, denoted by $\mathrm{det}(\mathrm{pos}_C(C'))$. Moreover, for every virtual edge ε in $\mathrm{skel}(\mu)$ we compute a list $\mathrm{contr}(\varepsilon)$ of relative positions. A relative position $\mathrm{pos}_C(C')$ is contained in $\mathrm{contr}(\varepsilon)$ if and only if it is crucial, determined by μ and C' is contracted in ε. Similarly, the list $\mathrm{detcyc}(\mu)$ for an R-node μ contains the crucial relative position $\mathrm{pos}_C(C')$ if and only if C and C' are both cycles in $\mathrm{skel}(\mu)$, implying that $\mathrm{pos}_C(C')$ is determined by μ. Finally, in the fourth phase, we compute the PR-node constraints restricted to the crucial relative positions. The next lemma states that the first phase can be implemented in linear time.

Lemma 6. *Let G be a biconnected planar graph with SPQR-tree $\mathcal{T}$ containing the disjoint cycles $\mathcal{C}$. The data $\mathrm{cyc}(\mu)$ for every node μ, $\mathrm{bel}(\varepsilon)$ for every virtual edge ε, and $\mathrm{root}(\mathcal{T}\vert_{C_i})$ for every cycle C_i can be computed in overall linear time.*

Proof. We process the SPQR-tree $\mathcal{T}$ bottom-up, starting with the Q-nodes. If a Q-node μ corresponds to an edge belonging to a cycle C, then $\text{cyc}(\mu)$ contains only C and $\text{bel}(\varepsilon) = C$ for the virtual edge in μ. If the edge corresponding to μ is not contained in a cycle, then $\text{cyc}(\mu)$ is empty and $\text{bel}(\varepsilon) = \bot$. Furthermore, a Q-node cannot be the root of any induced subtree $\mathcal{T}|_C$ as we chose as the root of $\mathcal{T}$ a Q-node corresponding to an edge not contained in any of the cycles. Now consider an inner node μ. We first process the virtual edges in $\text{skel}(\mu)$ not belonging to the parent of μ. Let ε be such a virtual edge corresponding to the child μ' of μ and let ε' be the virtual edge in $\text{skel}(\mu')$ corresponding to its parent μ. Then ε belongs to a cycle induced by C if and only if ε' does, thus we set $\text{bel}(\varepsilon) = \text{bel}(\varepsilon')$. Moreover, if $\text{bel}(\varepsilon) \neq \bot$ we need to add the cycle $\text{bel}(\varepsilon)$ to $\text{cyc}(\mu)$ if it was not already added. Whether $\text{bel}(\varepsilon)$ is already contained in $\text{cyc}(\mu)$ can be tested in constant time as follows. We define a timestamp t, increase t every time we go to the next node in $\mathcal{T}$ and we store the current value of t for a cycle added to $\text{cyc}(\mu)$. Then a cycle C was already added to $\text{cyc}(\mu)$ if and only if the timestamp stored for C is equal to the current timestamp t. Thus, processing all virtual edges in $\text{skel}(\mu)$ not corresponding to the parent of μ takes time linear in the size of $\text{skel}(\mu)$. Let now $\varepsilon = \{s, t\}$ be the virtual edge corresponding to the parent of μ. If $\text{bel}(\varepsilon') = \bot$ for all virtual edges ε' incident to s, then ε cannot be contained in a cycle induced by any of the cycles in $\mathcal{C}$. Otherwise, there are two possibilities. There is a cycle $C \in \mathcal{C}$ such that $\text{bel}(\varepsilon_1) = C$ for exactly one virtual edge ε_1 incident to s or there are two such edges ε_1 and ε_2 with $\text{bel}(\varepsilon_1) = \text{bel}(\varepsilon_2) = C$. In the former case the edges belonging to C in $\text{skel}(\mu)$ form a path from s to t, thus the edge ε corresponding to the parent also belongs to C and we set $\text{bel}(\varepsilon) = C$. In the latter case s is contained in the cycle C but the virtual edge does not belong to C and we set $\text{bel}(\varepsilon) = \bot$. This takes time linear in the degree of s in $\text{skel}(\mu)$ and hence lies in $\mathcal{O}(|\text{skel}(\mu)|)$. It remains to set $\text{root}(\mathcal{T}|_C) = \mu$ for every cycle C inducing the subtree $\mathcal{T}|_C$ having μ as root. The tree $\mathcal{T}|_C$ has μ as root if and only if C is contained as cycle κ in μ but the virtual edge ε in $\text{skel}(\mu)$ corresponding to the parent of μ is not contained in κ. Thus we have to set $\text{root}(\mathcal{T}|_C) = \mu$ for all cycles C in $\text{cyc}(\mu)$ except for $\text{bel}(\varepsilon)$. Note that this again consumes time linear in the size of $\text{skel}(\mu)$ since the number of cycles that are cycles in μ is in $\mathcal{O}(|\text{skel}(\mu)|)$. Due to the fact that the SPQR-tree $\mathcal{T}$ has linear size this yields an overall linear running time. $\quad\square$

In the second phase we want to compute $\text{high}(\mu)$ for each of the nodes in $\mathcal{T}$. We obtain the following lemma.

Lemma 7. *Let G be a biconnected planar graph with SPQR-tree $\mathcal{T}$ containing the disjoint cycles $\mathcal{C}$. For every node μ in $\mathcal{T}$ the edge $\text{high}(\mu)$ can be computed in linear time.*

Proof. We make use of the fact that $\text{bel}(\varepsilon)$ is already computed for every virtual edge ε in each of the skeletons, which can be done in linear time due to Lemma 6. Note that an edge $\{\mu, \mu'\}$ in the SPQR-tree $\mathcal{T}$ (where μ is the parent of μ') belongs to the induced subtree $\mathcal{T}|_C$ with respect to the cycle $C \in \mathcal{C}$ if and only if $\text{bel}(\varepsilon) = C$ for the virtual edge ε in $\text{skel}(\mu)$ corresponding to the child μ'. In this case we also have $\text{bel}(\varepsilon') = \text{bel}(\varepsilon) = C$ where ε' is the virtual edge in $\text{skel}(\mu')$ corresponding to the parent. Hence, we can compute $\text{high}(\mu)$ for every node μ in $\mathcal{T}$ by processing $\mathcal{T}$ top–down remembering the latest processed edge not belonging to any of the induced subtrees. This can easily be done in linear time. $\quad\square$

In the third phase we compute $\det(\text{pos}_C(C'))$ for every crucial relative position in linear time. Moreover, we compute $\text{contr}(\varepsilon)$ for every virtual edge ε and $\text{detcyc}(\mu)$ for every R-node μ. We show the following lemma.

Lemma 8. *Let G be a biconnected planar graph with SPQR-tree $\mathcal{T}$ containing the disjoint cycles $\mathcal{C}$. The node $\det(\text{pos}_C(C'))$ for each crucial relative position $\text{pos}_C(C')$, the list $\text{contr}(\varepsilon)$ for each virtual edges ε and the list $\text{detcyc}(\mu)$ for each R-nodes μ can be computed in overall linear time.*

Proof. Let C and C' be two cycles such that $\text{pos}_C(C')$ is a crucial relative position. We show how to compute the node $\det(\text{pos}_C(C'))$ determining this relative position in constant time. Moreover, if C' is contracted in a virtual edge ε in $\det(\text{pos}_C(C'))$, we append the relative position $\text{pos}_C(C')$ to $\text{contr}(\varepsilon)$. Otherwise, $\det(\text{pos}_C(C'))$ is an R-node containing C and C' as cycles and we add $\text{pos}_C(C')$ to $\text{detcyc}(\det(\text{pos}_C(C')))$. Since there are only linearly many crucial relative positions this takes only linear time. Let $\mu = \text{root}(\mathcal{T}|_C)$ and $\mu' = \text{root}(\mathcal{T}|_{C'})$ be the roots of the induced trees with respect to C and C', respectively, which are already computed due to Lemma 6. We use that the lowest common ancestor of a pair of nodes can be computed in constant time after a linear-time preprocessing [17,18]. In particular, let $\text{LCA}(\mu, \mu')$ be the lowest common ancestor of the two roots. There are three possibilities. First, $\text{LCA}(\mu, \mu')$ is above μ (Fig. 8(a)). Second, $\text{LCA}(\mu, \mu') = \mu = \mu'$ (Fig. 8(b)). And third, $\text{LCA}(\mu, \mu') = \mu$ lies above μ' (Fig. 8(c–f)). Note that the first case includes the situation where $\mu' = \text{LCA}(\mu, \mu')$ lies above μ.

In the first case the cycle C' is contracted in μ in the virtual edge ε corresponding to the parent of μ, while μ contains C as cycle κ not containing the virtual edge ε corresponding to the parent. Hence, μ determines $\text{pos}_C(C')$. We set $\det(\text{pos}_C(C')) = \mu$ and insert $\text{pos}_C(C')$ into $\text{contr}(\varepsilon)$. In the second case C and C' are both cycles in $\mu = \mu'$, hence μ determines $\text{pos}_C(C')$. We set $\det(\text{pos}_C(C')) = \mu$ and insert $\text{pos}_C(C')$ into $\text{detcyc}(\mu)$ since $\text{skel}(\mu)$ contains C and C' as cycles.

In the third case the node determining $\text{pos}_C(C')$ lies somewhere on the path from μ down to μ'. In this situation $\text{high}(\mu')$ comes into play and we distinguish several cases. We first assume that $\text{high}(\mu') \neq \bot$. Let $\{\eta, \eta'\} = \text{high}(\mu')$ be the highest edge in $\mathcal{T}$ on the path from μ' to the root that is reachable without using an edge in any of the induced trees,
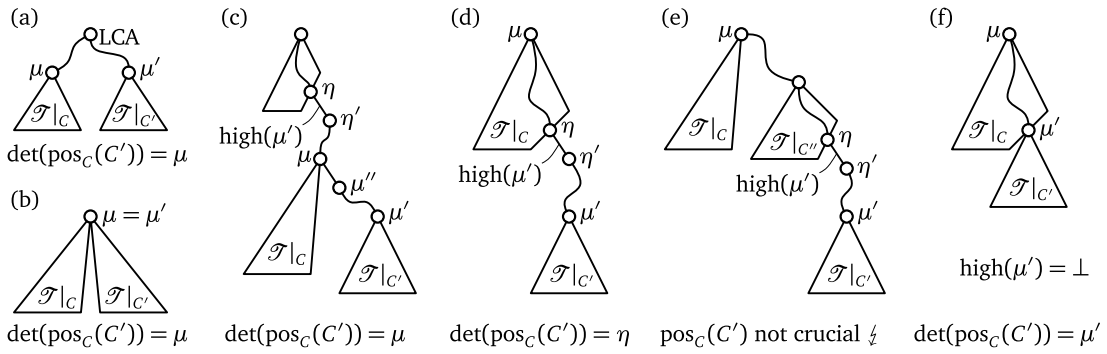
Fig. 8. The cases that occur in the proof of Lemma 8.

as computed by Lemma 7. Let η be the parent of η'. We claim that either μ or η determines the crucial relative position $pos_C(C')$.

More precisely, if η lies above or is equal to μ (Fig. 8(c)), then the child μ'' of μ on the path from μ' to μ does not contain C as a cycle. Otherwise the edge $\{\mu, \mu''\}$ would have been contained in $\mathcal{T}|_C$, which is a contradiction to the definition of $high(\mu')$. Thus, C' is contracted in the virtual edge ε in $skel(\mu)$ corresponding to the child μ'' that is not contained in the cycle induced by C, implying that μ determines $pos_C(C')$. In this case we set $det(pos_C(C')) = \mu$. Moreover, we want to insert the crucial relative position $pos_C(C')$ into $contr(\varepsilon)$. Unfortunately, we cannot determine the virtual edge ε belonging to the child μ'' in constant time. We handle that problem by storing a temporary list $temp(\mu)$ for the node μ and insert $pos_C(C')$ into this list. After we have processed all crucial relative positions, we process $\mathcal{T}$ bottom-up, building a union-find data structure by taking the union of μ with all its children after processing μ. Thus, when processing μ, we can simply traverse the list $temp(\mu)$ once, find for every crucial relative position $pos_C(C')$ the virtual edge ε containing C' by finding $root(\mathcal{T}|_{C'})$ in the union-find data structure and then add $pos_C(C')$ to $contr(\varepsilon)$. Note that this takes overall linear time, because the union-find data structure consumes amortized constant time per operation since the union-operations we apply are known in advance [19].

In the second case η lies below μ, where $high(\mu') = \{\eta, \eta'\}$. We claim that η contains C as a cycle and C' contracted in the virtual edge ε' in $skel(\eta)$ corresponding to the child η', as depicted in Fig. 8(d). By definition of $high(\mu')$ there is a cycle C'' that is contained as a cycle in η and in the parent of η but not in η'. We show that $C'' = C$ or $pos_C(C')$ is not a crucial relative position. Assume $C'' \neq C$; see Fig. 8(e). In $skel(\eta)$ the cycle C' is contracted in the virtual edge ε' corresponding to the child η', whereas C is contracted in the virtual edge ε corresponding to the parent of η. Since C'' is a cycle in η and in its parent, it induces a cycle in $skel(\eta)$ containing ε. Consider a path π from C' to C in the graph G. Then π contains one of the poles of $skel(\eta)$ and hence contains a vertex in C''. Thus the relative position $pos_C(C')$ is not crucial, which is a contradiction. Hence we can simply set $det(pos_C(C')) = \eta$ and append $pos_C(C')$ to the list $contr(\varepsilon')$.

Finally, $high(\mu')$ may be not defined, that is $high(\mu') = \perp$ since the edge connecting μ' to its parent is already contained in one of the induced cycle trees. With a similar argument as before, this induced tree is $\mathcal{T}|_C$, belonging to the cycle C, as depicted in Fig. 8(f). Thus μ' contains C and C' as cycles and we set $det(pos_C(C')) = \mu'$ and add $pos_C(C')$ to the list $detcyc(\mu')$. This concludes the proof. □

In the fourth and last phase we process the SPQR-tree $\mathcal{T}$ once more to finally compute the PR-node constraints restricted to the crucial relative positions. We obtain the following lemma.

Lemma 9. *Let G be a biconnected planar graph. The PR-node constraints restricted to the crucial relative positions can be computed in linear time.*

Proof. We process each node in the SPQR-tree $\mathcal{T}$ of G once, consuming time linear in the size of its skeleton plus some additional costs that sum up to the number of crucial relative positions in total. Let μ be a node in $\mathcal{T}$. If μ is not contained in any induced tree $\mathcal{T}|_C$, it does not determine any relative position at all. Thus assume there is at least one cycle that is a cycle in μ. If μ is a P-node, $skel(\mu)$ consists of ℓ parallel virtual edges $\varepsilon_1, \ldots, \varepsilon_\ell$ and we can assume without loss of generality that the cycle C induces in $skel(\mu)$ the cycle κ consisting of the two virtual edges ε_1 and ε_2. For every crucial relative position $pos_C(C')$ that is determined by μ there is a virtual edge $\varepsilon \in \{\varepsilon_3, \ldots, \varepsilon_\ell\}$ containing C' in the list $contr(\varepsilon)$, which is already computed due to Lemma 8. Hence, the PR-node constraints stemming from μ can be computed by processing each of these lists $contr(\varepsilon)$, setting $pos_C(C') = pos_C(C'')$ for any two cycles C' and C'' appearing consecutively in $contr(\varepsilon)$. The time-consumption is linear in the size of $skel(\mu)$ plus the number of crucial relative positions determined by μ.

If μ is an R-node, it may contain several cycles as a cycle, all of them stored in the list $cyc(\mu)$ due to Lemma 6. Every crucial relative position $pos_C(C')$ determined by μ is either contained in the lists $contr(\varepsilon)$ for a virtual edge ε in $skel(\mu)$

or in detcyc(μ) if C and C' are both cycles in μ (Lemma 8). We first carry the relative positions in detcyc(μ) over to the corresponding cycles. More precisely, we define a list detcyc(C') for every cycle C' in cyc(μ) and insert a crucial relative position pos$_C(C')$ into it, if it is contained in detcyc(μ). This can obviously be done consuming time linear in the size of detcyc(μ). Afterwards, we start by fixing the embedding of skel(μ) and pick an arbitrary vertex v_0 in skel(μ). For each cycle C contained as cycle κ in skel(μ) we define a variable side(C) and initialize it with the value "left" or "right", depending on which side v_0 lies with respect to κ in the chosen embedding of skel(μ), or with the value "on" if v_0 is contained in C. Due to Lemma 6 we know for every edge ε to which cycle it belongs (or that it does not belong to a cycle at all). Thus side(C) can be easily computed for every cycle C that is a cycle in skel(μ) consuming time linear in $|\text{skel}(\mu)|$ by traversing skel(μ) once, starting at v_0.

To sum up, each crucial relative position pos$_C(C')$ determined by μ is either contained in contr(ε) if C' is contracted in ε or in detcyc(C') if C' is a cycle in skel(μ). Moreover, for each cycle C the value of side(C) describes on which side of C the chosen start-vertex v_0 lies with respect to a chosen orientation of skel(μ). We now want to divide the crucial relative positions determined by μ into two lists LEFT and RIGHT depending on which value they have with respect to the chosen embedding. If this is done, the PR-node constraints stemming from μ restricted to the crucial relative positions can be computed by simply processing these two lists once. To construct the lists LEFT and RIGHT, we make a DFS-traversal in skel(μ) such that each virtual edge is processed once. More precisely, when we visit an edge $\{u, v\}$ (starting at u), then v is either an unvisited vertex and we continue the traversal from v or v was already visited, then we go back to u. If all virtual edges incident to the current vertex u were already visited, we do a back-tracking step, i.e., we go back to the vertex from which we moved to u. Essentially, a normal step consists of three phases, leaving the current vertex u, traveling along the virtual edge $\{u, v\}$, and finally arriving at v or back at u. In a back-tracking step we have only two phases, namely leaving the current vertex u and arriving at its predecessor. During the whole traversal we keep track of the sides side($\cdot$). More precisely, when leaving a vertex u that was contained in a cycle C we may have to update side(C) if the target-vertex v is not also contained in C. On the other hand, when arriving at a vertex v contained in a cycle C we have to set side(C) = "on". Since such an update has to be done for at most one cycle we can keep track of the sides in constant time per operation and thus in overall linear time. Now it is easy to compute the values of the crucial relative positions determined by μ with respect to the currently chosen embedding. While traveling along a virtual edge $\varepsilon = \{u, v\}$ we process contr(ε). For a crucial relative position pos$_C(C')$ contained in contr(ε) we know that C' is contracted in ε. Thus, in the chosen embedding the value of pos$_C(C')$ is the current value of side(C) and we can insert pos$_C(C')$ into the list LEFT or RIGHT depending on the value of side(C). This takes linear time in the number of crucial relative positions contained in contr(ε). We deal with the crucial relative positions contained in one of the lists detcyc(C') in a similar way. Every time we reach a vertex v contained in a cycle C' we check whether this is the first time we visit the cycle C'. If it is the first time, we insert every crucial relative positions pos$_C(C')$ contained in detcyc(C') into one of the lists LEFT or RIGHT, depending on the current value of side(C). Clearly the whole traversal takes linear time in the size of skel(μ) plus linear time in the number of crucial relative positions determined by μ. Moreover, we obviously obtain the PR-node constraints restricted to the crucial relative positions stemming form μ by processing each of the lists LEFT and RIGHT once, obtaining an equality constraint for positions that are adjacent in the lists and additionally a single inequality for a pair of positions, one contained in LEFT and the other in RIGHT, unless one of them is empty. $\quad\square$

Corollary 1. *The CC-tree $\mathcal{T}_C$ of a biconnected planar graph G can be computed in linear time.*

It remains to extend the described algorithm to the case where G is not necessarily biconnected. More precisely, we need to show how to compute the extended PR-node constraints and the cutvertex constraints in linear time. This is done in the proof of the following theorem.

Theorem 6. *The CC-tree $\mathcal{T}_C$ of a connected planar graph G can be computed in linear time.*

Proof. As before, the underlying C-tree can be easily computed in linear time. For a fixed block B we have the SPQR-tree $\mathcal{T}$ and for a cycle C in B the induced tree $\mathcal{T}|_C$ can be defined as before. Obviously, Lemma 6 can be used as before to compute cyc(μ) for every node μ, bel(ε) for every virtual edge and root($\mathcal{T}|_C$) for every induced subtree in linear time. Moreover, the edge high(μ) in $\mathcal{T}$ can be computed for every node μ as in Lemma 7. For the computation of det(pos$_C(C')$) for every crucial relative position pos$_C(C')$ and contr(ε) for every virtual edge ε, we cannot directly apply Lemma 8 since the cycles C and C' may be contained in different blocks. Thus, before we can compute det(pos$_C(C')$), we need to find out whether C and C' are in the same block, which can be done by simply storing for every cycle a pointer to the block containing it. For the case that C and C' are contained in the same block det(pos$_C(C')$) can be computed as before and pos$_C(C')$ can be inserted into the list contr(ε) for some ε if necessary. For the case that C and C' are contained in different blocks B and B', we need to find the unique cutvertex v in B that separates B and B'. This can be done in overall linear time by computing the BC-tree and using an approach combining the lowest common ancestor and union-find data structure similar as in the proof of Lemma 8.

If the resulting cutvertex v is not contained in C, we can treat the cutvertex v as if it was the cycle C' and use the same algorithm as in Lemma 8 to compute det(pos$_C(C')$) and append pos$_C(C')$ to contr(ε) for some ε if necessary. If v is

contained in C, then the crucial relative position $\mathrm{pos}_C(C')$ is not determined by any node in any SPQR-tree at all, but by the embedding of the blocks around the cutvertex v. Thus there are no extended PR-node constraints restricting $\mathrm{pos}_C(C')$. Finally, $\det(\mathrm{pos}_C(C'))$ can be computed in overall linear time for every crucial relative position $\mathrm{pos}_C(C')$ that is determined by a node in the SPQR-tree of the block containing C. Moreover, for a node μ in the SPQR-tree of the block B containing C every virtual edge ε has a list $\mathrm{contr}(\varepsilon)$ containing all crucial relative positions $\mathrm{pos}_C(C')$ that are determined by μ and for which either C' is contracted in ε or belongs to a different block B' and is connected to B via a cutvertex contained in the expansion graph $\exp(\varepsilon)$. With these information the extended PR-node constraints can be computed exactly the same as the PR-node constraints are computed in Lemma 9.

It remains to compute the cutvertex constraints restricted to the crucial relative positions. As mentioned above, we can compute a list of crucial relative positions $\mathrm{pos}_C(C_1), \ldots, \mathrm{pos}_C(C_\ell)$ determined by the embedding of the blocks around a cutvertex v contained in C in linear time. We then process this list once, starting with $\mathrm{pos}_C(C_1)$. We store $\mathrm{pos}_C(C_1)$ as reference position for the block B_1 containing C_1. Now, when processing $\mathrm{pos}_C(C_i)$, we check whether the block B_i containing C_i already has a reference position $\mathrm{pos}_C(C_j)$ assigned to it. In this case we set $\mathrm{pos}_C(C_i) = \mathrm{pos}_C(C_j)$, otherwise we set $\mathrm{pos}_C(C_i)$ to be the reference position. This obviously consumes overall linear time and computes the cutvertex constraints restricted to the crucial relative positions. □

Intersecting CC-trees in linear time

Due to Theorem 5 we can test whether two graphs $G^{①}$ and $G^{②}$ with common graph C consisting of a set of disjoint cycles have a SEFE by computing the CC-trees $\mathcal{T}_C^{①}$ and $\mathcal{T}_C^{②}$ of $G^{①}$ and $G^{②}$, respectively, which can be done in linear time due to Theorem 6. Then the intersection $\mathcal{T}_C$ of $\mathcal{T}_C^{①}$ and $\mathcal{T}_C^{②}$ represents exactly the possible embeddings of the common graph G in a SEFE. It remains to show that the intersection can be computed in linear time.

Theorem 7. *The intersection of two CC-trees can be computed in linear time.*

Proof. Let $\mathcal{T}_C^{①}$ and $\mathcal{T}_C^{②}$ be two CC-trees on a set $\mathcal{C}$ of cycles. We start with $\mathcal{T}_C = \mathcal{T}_C^{①}$ and show how to compute the common-face and crucial-position constraints in overall linear time. For the crucial-position constraints we essentially only show how to find for each crucial relative position in $\mathcal{T}_C^{②}$ a crucial relative position in $\mathcal{T}_C$ corresponding to it. Computing the crucial-position constraints is then easy. We root $\mathcal{T}_C$ at an arbitrary vertex and again use that the lowest common ancestor of two vertices in $\mathcal{T}_C$ can be computed in constant time [17,18]. For every edge $e^{②} = \{C_1, C_2\}$ in $\mathcal{T}_C^{②}$ we obtain a path in $\mathcal{T}_C$ from C_1 to the lowest common ancestor of C_1 and C_2 and further to C_2. We essentially process these two parts of the path separately with some additional computation for the lowest common ancestor. We say that the parts of the paths *belong* to the *half-edges* $e_1^{②}$ and $e_2^{②}$, respectively. We use the following data structure. For every cycle C there is a list $\mathrm{end}(C)$ containing all edges in $\mathcal{T}_C^{②}$ whose endpoints have C in $\mathcal{T}_C$ as lowest common ancestor. This list can be computed for every cycle in overall linear time. We then process $\mathcal{T}_C$ bottom up, saving for the cycle C we currently process a second list $\mathrm{curr}(C)$ containing all the half-edges in $\mathcal{T}_C^{②}$ whose paths contain C. This can be done in overall linear time by ensuring that every half-edge $e_i^{②}$ ($i = 1, 2$) is contained in at most one list $\mathrm{curr}(C)$ at the same time. Then $e_i^{②}$ can be removed from this list in constant time by storing for $e_i^{②}$ pointers to the previous and to the next element in that list, denoted by $\mathrm{prev}(e_i^{②})$ and $\mathrm{next}(e_i^{②})$. Additionally, we build up the following union-find data structure. Every time we have processed a cycle C, we union C with all its children in $\mathcal{T}_C$. Thus, when processing C, this data structure can be used to find for every cycle in the subtree below C the child of C it belongs to. Note that again this version of the union-find data structure consumes amortized constant time per operation since the sequence of union operations is known in advance [19]. Before starting to process $\mathcal{T}_C$, we process $\mathcal{T}_C^{②}$ once and for every edge $e^{②} = \{C_1, C_2\}$ we insert the half-edges $e_1^{②}$ and $e_2^{②}$ to the lists $\mathrm{curr}(C_1)$ and $\mathrm{curr}(C_2)$, respectively. While processing $\mathcal{T}_C$ bottom up the following invariants hold at the moment we start to process C.

1. The list $\mathrm{curr}(C)$ contains all half-edges starting at C.
2. For every child C' of C the list $\mathrm{curr}(C')$ contains the half-edge $e_i^{②}$ if and only if the path belonging to it contains C and C'.
3. Every half-edge $e_i^{②}$ is contained in at most one list $\mathrm{curr}(C)$, and $\mathrm{prev}(e_i^{②})$ and $\mathrm{next}(e_i^{②})$ contain the previous and next element in that list, respectively.

When we start processing a leaf C the invariants are obviously true. To satisfy invariant 2. for the parent of C we have to ensure that all half-edges ending at C are removed form the list $\mathrm{curr}(C)$. Since there are no half-edges ending in a leaf, we simply do nothing. Invariants 1. and 3. obviously also hold for the parent of C.

Let C be an arbitrary cycle and assume that the invariants are satisfied. To ensure that invariant 2. holds for the parent of C, we need to build a list of all half-edges whose paths contain C and do not end at C. Since invariants 1. and 2. hold for C this are exactly the half-edges contained in $\mathrm{curr}(C)$ plus the half-edges contained in $\mathrm{curr}(C')$ for each of the children C'

of C that are not ending at C. Note that a half-edge may also start at C and end at C. This is the case if the corresponding edge connects C with another cycle C'' such that the lowest common ancestor of C and C'' is C. We first process the list end(C) containing the edges ending at C; let $e^{②}$ be an edge in end(C). The two half-edges $e_1^{②}$ and $e_2^{②}$ belonging to $e^{②}$ are contained in the lists curr(C_1) and curr(C_2), where C_1 and C_2 are different cycles and each of them is either C or a child of C. We remove $e_i^{②}$ form curr(C_i) for $i = 1, 2$. This can be done by setting the pointers next$(\text{prev}(e_i^{②})) = \text{next}(e_i^{②})$ and prev$(\text{next}(e_i^{②})) = \text{prev}(e_i^{②})$, taking constant time per edge since each half-edge is contained in at most one list due to invariant 3. Afterwards, for every child C' of C, we append curr(C') to curr(C) and empty the list curr(C') afterwards, to ensure that invariant 3 remains satisfied. This takes constant time per child and thus overall time linear in the degree of C. Obviously this satisfies all invariants for the parent of C. Furthermore, we consume time linear in the degree of C plus time linear in the number of half-edges ending at C. However, every half-edge ends exactly once yielding overall linear time.

Now it is easy to compute the common-face and crucial-position constraints while processing $\mathcal{T}_C$ as described above. Essentially, when processing C, we compute all the constraints concerning the relative position of other cycles with respect to C. In particular, we need to add common-face constraints if two half-edges end at C and if the path belonging to a half-edge contains C in its interior. Furthermore, we find a corresponding crucial relative position for every half-edge starting at C. Let $e^{②} = \{C_1, C_2\}$ be an edge whose half-edges end at C. There are two different cases. First, one of the cycles C_i (for $i = 1, 2$) is C (its half edge starts and ends at C). Then the other cycle (whose half-edge only ends at C) is contained in a subtree with root C', where C' is a child of C. Second, C_1 and C_2 are contained in the subtrees with roots C_1' and C_2', respectively, where C_1' and C_2' are different children of C. In this case, both half-edges end at C. We consider the second case first. Then C_1' and C_2' can be found in amortized constant time by finding C_1 and C_2 in the union-find data structure. The equation $\text{pos}_C(C_1') = \text{pos}_C(C_2')$ is exactly the common-face constraint at the cycle C stemming from the edge $e^{②}$. In the second case we can again find the child C' in constant time. Assume without loss of generality that $C_1 = C$ and C_2 is contained in the subtree having C' as root. Then $\text{pos}_C(C')$ is the crucial relative position in $\mathcal{T}_C$ corresponding to the crucial relative position $\text{pos}_C(C_2)$ in $\mathcal{T}_C^{②}$. The half-edges containing C in its interior are exactly the half-edges contained in one of the lists curr(C') for a child C' of C whose path does not end at C. Thus, for the parent C'' of C we have to add the common-face constraint $\text{pos}_C(C') = \text{pos}_C(C'')$ if and only if the list curr(C') is not empty after deleting all half-edges in end(C). These additional computations obviously do not increase the running time and hence the common-face and crucial-position constraints can be computed in overall linear running time. $\square$

Theorems 4, 5, 6 and 7 directly yield the following results.

Theorem 8. Simultaneous Embedding with Fixed Edges *can be solved in linear time if the common graph consists of disjoint cycles.*

Theorem 9. SEFE *can be solved in linear time for the case of k graphs $G^{①}, \ldots, G^{⑱}$ all intersecting in the same common graph G consisting of disjoint cycles.*

4. Connected components with fixed embedding

In this section we show how the previous results can be extended to the case that the common graph has several connected components, each of them with a fixed planar embedding. Again, we first consider the case of a single graph G containing $\mathcal{C}$ as a subgraph, where in this case $\mathcal{C}$ is a set of connected components instead of a set of disjoint cycles. First note that the relative position $\text{pos}_C(C')$ of a component C' with respect to another component C can be an arbitrary face of C. Thus, the choice of the relative positions is no longer binary and a set of inequalities on the relative positions would lead to a coloring problem in the conflict graph, which is $\mathcal{NP}$-hard in general. However, most of the constraints between relative positions are equations, in fact, all inequalities stem from R-nodes in the SPQR-tree of G (or of the SPQR-tree of one of the blocks in G). Fortunately, if a relative position is determined by an R-node, there are only two possibilities to embed this R-node. Thus, the possible values for the relative position is restricted to two faces, yielding a binary decision. Note that in general the possible values for $\text{pos}_C(C')$ are not all faces of C, even if $\text{pos}_C(C')$ is not determined by an R-node but by a P-node or by the embedding around a cutvertex.

Thus, we obtain for each relative position a set of possible faces as values and additionally several equations and inequalities, where inequalities only occur between relative positions with a binary choice. These conditions can be modeled as a conflict graph where each node represents a relative position with some allowed colors (faces) and edges in this conflict graph enforce both endvertices to be either colored the same or differently. In the case of the problem SEFE each of the graphs yields such a conflict graph. These conflict graphs can be easily merged by intersecting for each relative position the sets of allowed colors (faces). Then a simultaneous embedding can be constructed by first iteratively contracting edges requiring equality, intersecting the possible colors of the involved nodes. If the resulting graph contains a node with the empty set as choice for the color, then no simultaneous embedding exists. Otherwise, we have to test whether each connected component in the remaining graph can be colored such that adjacent nodes have different colors, which can be done efficiently since such a component either consists of a single node or there are only up to two possible colors for each connected component left due to the considerations above.

Moreover, the CC-tree can be adapted to work for the case of connected components with fixed embeddings instead of disjoint cycles, as the extended PR-node and cutvertex constraints on the crucial relative positions are still sufficient to imply them on all relative positions. We call this tree on connected components the CC$^\oplus$-tree, standing for *constrained component-tree*. In the following we quickly go through the steps we did before in the case of disjoint cycles and describe the changes when considering connected components instead.

PR-node constraints. Let G be a biconnected planar graph and let $\mathcal{C}$ be a subgraph of G consisting of several connected components, each with a fixed planar embedding. Let further $C \in \mathcal{C}$ be one of the connected components and let μ be a node in the SPQR-tree $\mathcal{T}$ of G. The virtual edges in skel(μ) whose expansion graphs contain parts of the component C induce a connected subgraph in skel(μ). In the previous case, where the subgraph consisted of disjoint cycles, this induced subgraph was either a single edge or a cycle. In the case that C is an arbitrary component the induced graph can be an arbitrary connected subgraph of skel(μ). If it is a single edge, we say that C is *contracted* in μ, otherwise C is a *component* in μ.

We obviously obtain that the relative position $\mathrm{pos}_C(C')$ of another component C' with respect to C is determined by the embedding of skel(μ) if and only if C is a component in μ and C' is not contracted in one of the virtual edges belonging to the subgraph induced by C. Moreover, the embedding of skel(μ) is partially (or completely) fixed by the embedding of C if the induced graph in skel(μ) contains a vertex with degree greater than 2. More precisely, consider μ to be a P-node containing C as a component. Then the virtual edges belonging to C have a fixed planar embedding and each face in this induced graph represents a face in C. These faces are the possible values for the relative positions with respect to C that are determined by μ. The remaining virtual edges not belonging to C can be added arbitrarily and thus components contracted in these edges can be put into one of the possible faces with the restriction that two components contracted in the same virtual edge have to lie in the same face, that is they have the same relative position with respect to C. To sum up, we obtain a set of possible faces of C with respect to μ and a set of equations between relative positions of components with respect to C.

For the case that μ is an R-node, either the embedding of skel(μ) is fixed due to the fact that there exists a component whose induced graph in skel(μ) contains a vertex with degree greater than 2. Otherwise, each component is either contracted in μ or the induced subgraph is a cycle or a path. No matter which case arises, the relative positions determined by μ are either completely fixed or there are only two possibilities. If the embedding is fixed, the relative positions determined by μ are fixed and thus there is no need for additional constraints. Otherwise, a crucial relative position with respect to C is fixed if C induces a path in skel(μ) and it changes by flipping skel(μ) if C induces a cycle. For two components C and C' both inducing a cycle in skel(μ) this yields a bijection between the two possible values for relative positions with respect to C determined by μ and the two possible values for positions with respect to C'. Thus we can add the equations and inequalities as in the case of disjoint cycles.

The resulting constraints are again called PR-node constraints. As for disjoint cycles we obtain that an embedding of the components $\mathcal{C}$ respecting the fixed embeddings for each component can be induced by an embedding chosen for G if and only if the PR-node constraints are satisfied. This directly yields a polynomial-time algorithm to solve SEFE for the case that both graphs are biconnected and the common graph consists of several connected components, each having a fixed planar embedding.

Extended PR-node and cutvertex constraints. As for cycles the considerations above can be easily extended to the case that the graph G containing the components $\mathcal{C}$ is allowed to contain cutvertices. For a cutvertex v not contained in a component C, the relative position of v with respect to C determines the relative positions of components attached via v, which again yields the extended PR-node constraints. If v is contained in C, then the relative position of another component C' with respect to C is determined by the embedding around v if and only if v splits C from C'. In this case C' can obviously lie in one of the faces of C incident to v. Fortunately, the cutvertex constraints do not contain inequalities as they only ensure that components attached to v via the same block lie in the same face of C. With these considerations all results form Section 3.1 can be extended to the case of components with fixed embedding instead of cycles. In particular, SEFE can be solved in polynomial time if the common graph consists of connected components, each with a fixed planar embedding.

CC$^\oplus$-trees. As mentioned before, the CC-tree can be adapted to represent all embeddings that can be induced on the set of components $\mathcal{C}$ by an embedding of the graph G, yielding the CC$^\oplus$-tree. To this end, each node in the tree represents a component $C \in \mathcal{C}$ and the incidence to C of an edge $\{C, C'\}$ in the CC$^\oplus$-tree represents the choice for the crucial relative position $\mathrm{pos}_C(C')$. The possible values are restricted to a subset of faces of C as described before and there may be some equations between crucial relative positions with respect to C. Moreover, there may be inequalities between crucial relative positions even with respect to different components. However, if this is the case, then there are at most two possible choices and we have a bijection between the possible faces of different components. As in the proof of Theorem 4, it follows from the structure of the underlying C-tree, that relative positions that are not crucial are determined by a crucial relative position that is determined by the same P- or R-node or by the same embedding choice around a cutvertex. The proof can be easily adapted to the case of components instead of cycles yielding that satisfying the constraints and restrictions to a subset of faces for the crucial relative positions automatically satisfies these conditions for all relative positions.

To be able to solve SEFE with the help of $CC^{\oplus}$-trees, we need to intersect two $CC^{\oplus}$-trees such that the result is again a $CC^{\oplus}$-tree. Assume as in the case of cycles that we have the two $CC^{\oplus}$-trees $\mathcal{T}_{\mathcal{C}}^{①}$ and $\mathcal{T}_{\mathcal{C}}^{②}$. As before we start with $\mathcal{T}_{\mathcal{C}}^{①}$ and add the restrictions given by $\mathcal{T}_{\mathcal{C}}^{②}$. More precisely, for every pair $\{C, C'\}$ of adjacent nodes in $\mathcal{T}_{\mathcal{C}}^{②}$ we have to add the common-face constraints to $\mathcal{T}_{\mathcal{C}}^{①}$, that is equations between crucial relative positions on the path between C and C' in $\mathcal{T}_{\mathcal{C}}^{①}$ enforcing C and C' to share a face. Moreover, for every relative position $\text{pos}_C(C')$ that is crucial with respect to $\mathcal{T}_{\mathcal{C}}^{②}$ we have to add the equations and inequalities it is involved in to the $CC^{\oplus}$-tree $\mathcal{T}_{\mathcal{C}}^{①}$. As for cycles $\text{pos}_C(C')$ is in $\mathcal{T}_{\mathcal{C}}^{①}$ determined by the crucial relative position $\text{pos}_C(C'')$, where C'' is the first node on the path from C to C'. We have to do two things. First, we have to restrict the possible choices for $\text{pos}_C(C'')$ to those that are possible for $\text{pos}_C(C')$, which can easily be done by intersecting the two sets. Second, the equations and inequalities $\text{pos}_C(C')$ is involved in have to be carried over to $\text{pos}_C(C'')$. This can be done as before by choosing for each crucial relative position in $\mathcal{T}_{\mathcal{C}}^{②}$ the representative in $\mathcal{T}_{\mathcal{C}}^{①}$. For the resulting intersection $\mathcal{T}_{\mathcal{C}}$ it remains to show that every embedding represented by it is also represented by $\mathcal{T}_{\mathcal{C}}^{①}$ and $\mathcal{T}_{\mathcal{C}}^{②}$. The former is clear, the latter can be shown as in the proof of Theorem 5.

Efficient implementation. Unfortunately, the constrained component-tree may have quadratic size in contrast to the constrained cycle-tree, which has linear size. This comes from the fact that a node C in the $CC^{\oplus}$-tree may have linearly many neighbors. Moreover, each relative position $\text{pos}_C(C')$ of a neighbor C' of C may have linearly many possible values, as C may have that many faces. As these possible values need to be stored for the edge $\{C, C'\}$ in the CC-tree it has quadratic size. On the other hand, it is easy to see that the CC-tree can be computed in quadratic time. Moreover, the proof of Theorem 7 providing a linear-time algorithm to intersect $CC^{\oplus}$-trees can be adapted almost literally. The only thing that changes is that additionally the possible values for $\text{pos}_C(C'')$ and $\text{pos}_C(C')$ need to be intersected, where $\text{pos}_C(C')$ is a relative position that is crucial with respect to $\mathcal{T}_{\mathcal{C}}^{②}$ and $\text{pos}_C(C'')$ is the representative in $\mathcal{T}_{\mathcal{C}}^{①}$. Thus, two $CC^{\oplus}$-trees can be intersected consuming time linear in the size of the $CC^{\oplus}$-trees, that is quadratic time in the size of the input graphs. We finally obtain the following theorem.

Theorem 10. SIMULTANEOUS EMBEDDING WITH FIXED EDGES *can be solved in quadratic time, if the embedding of each connected component of the common graph is fixed.*

Let $\mathcal{T}_{\mathcal{C}}$ be the $CC^{\oplus}$-tree representing all embeddings of the components $\mathcal{C}$ that can be induced by the graph G. It is worth noting that, although the explicit representation of $\mathcal{T}_{\mathcal{C}}$ may have quadratic size, it also admits a compact representation of linear size. The key idea is the following. In case there are more than two possible values for a crucial relative position $\text{pos}_C(C')$, this position is determined by a P-node or a cutvertex. Then we can encode the possible values for $\text{pos}_C(C')$ by pointing to a list that is stored at that P-node or cutvertex, respectively. Since this set of values is independent of C' it is sufficient to store one list for each P-node or cutvertex. It is not hard to see that the total size of these lists is linear. Moreover, the fast algorithm for computing CC-trees can be applied with obvious modifications to compute this compact representation in linear time. It is, however, unclear whether the intersection of two or more $CC^{\oplus}$-trees still admits a compact representation and whether it can be computed in linear time from the given compact representations.

5. Conclusion

Contrary to the previous results on simultaneous embeddings we focused on the case where the embedding choice does not consist of ordering edges around vertices but of placing connected components in relative positions to one another. We first showed that generally both input graphs of an instance of SIMULTANEOUS EMBEDDING WITH FIXED EDGES can be always assumed to be connected. We then showed how to solve SIMULTANEOUS EMBEDDING WITH FIXED EDGES in linear time for the case that the common graph consists of simple disjoint cycles (or more generally has maximum degree 2). We further extended the result to a quadratic-time algorithm solving the more general case where the embedding of each connected component of the common graph is fixed. These solutions include a compact and easy to handle data structure, the CC-tree and $CC^{\oplus}$-tree, representing all possible simultaneous embeddings. Thus, there is hope that the CC-tree and the $CC^{\oplus}$-tree are also useful when relaxing the restriction of a fixed embedding for each component.

References

[1] T. Bläsius, S.G. Kobourov, I. Rutter, Simultaneous embedding of planar graphs, in: Handbook of Graph Drawing and Visualization, Chapman and Hall/CRC, 2013, pp. 349–381.
[2] E. Gassner, M. Jünger, M. Percan, M. Schaefer, M. Schulz, Simultaneous graph embeddings with fixed edges, in: F. Fomin (Ed.), Proceedings of the 32nd Workshop on Graph-Theoretic Concepts in Computer Science (WG'06), in: Lecture Notes in Computer Science, vol. 4271, Springer, Berlin/Heidelberg, 2006, pp. 325–335.
[3] J.J. Fowler, C. Gutwenger, M. Jünger, P. Mutzel, M. Schulz, An SPQR-tree approach to decide special cases of simultaneous embedding with fixed edges, in: I. Tollis, M. Patrignani (Eds.), Proceedings of the 16th International Symposium on Graph Drawing (GD'08), in: Lecture Notes in Computer Science, vol. 5417, Springer, Berlin/Heidelberg, 2009, pp. 157–168.

[4] J.J. Fowler, M. Jünger, S.G. Kobourov, M. Schulz, Characterizations of restricted pairs of planar graphs allowing simultaneous embedding with fixed edges, Comput. Geom. 44 (8) (2011) 385–398.

[5] P. Angelini, G. Di Battista, F. Frati, V. Jelínek, J. Kratochvíl, M. Patrignani, I. Rutter, Testing planarity of partially embedded graphs, in: Proceedings of the 21st Annual ACM–SIAM Symposium on Discrete Algorithms (SODA'10), SIAM, 2010, pp. 202–221.

[6] B. Haeupler, K. Jampani, A. Lubiw, Testing simultaneous planarity when the common graph is 2-connected, in: O. Cheong, K.-Y. Chwa, K. Park (Eds.), Proceedings of the 21st International Symposium on Algorithms and Computation (ISAAC'10), in: Lecture Notes in Computer Science, vol. 6507, Springer, Berlin/Heidelberg, 2010, pp. 410–421.

[7] P. Angelini, G. Di Battista, F. Frati, M. Patrignani, I. Rutter, Testing the simultaneous embeddability of two graphs whose intersection is a biconnected or a connected graph, J. Discrete Algorithms 14 (2012) 150–172.

[8] T. Bläsius, I. Rutter, Simultaneous PQ-ordering with applications to constrained embedding problems, in: Proceedings of the Twenty-Fourth Annual ACM–SIAM Symposium on Discrete Algorithm, SODA '13, ACM, 2013, pp. 1030–1043.

[9] M. Jünger, M. Schulz, Intersection graphs in simultaneous embedding with fixed edges, J. Graph Algorithms Appl. 13 (2) (2009) 205–218.

[10] M. Schaefer, Toward a theory of planarity: Hanani–Tutte and planarity variants, in: W. Didimo, M. Patrignani (Eds.), Graph Drawing, in: Lecture Notes in Computer Science, vol. 7704, Springer, Berlin/Heidelberg, 2013, pp. 162–173.

[11] G. Di Battista, R. Tamassia, On-line maintenance of triconnected components with SPQR-trees, Algorithmica 15 (4) (1996) 302–318.

[12] G. Di Battista, R. Tamassia, On-line planarity testing, SIAM J. Comput. 25 (5) (1996) 956–997.

[13] H. Whitney, Congruent graphs and the connectivity of graphs, Am. J. Math. 54 (1) (1932) 150–168.

[14] C. Gutwenger, P. Mutzel, A linear time implementation of SPQR-trees, in: Proc. 8th Internat. Sympos. Graph Drawing (GD'00), in: Lecture Notes in Computer Science, vol. 1984, Springer-Verlag, 2001, pp. 77–90.

[15] S. Even, A. Itai, A. Shamir, On the complexity of timetable and multicommodity flow problems, SIAM J. Comput. 5 (4) (1976) 691–703.

[16] B. Aspvall, M.F. Plass, R.E. Tarjan, A linear-time algorithm for testing the truth of certain quantified Boolean formulas, Inf. Process. Lett. 8 (3) (1979) 121–123.

[17] D. Harel, R.E. Tarjan, Fast algorithms for finding nearest common ancestors, SIAM J. Comput. 13 (2) (1984) 338–355.

[18] M.A. Bender, M. Farach-Colton, The LCA problem revisited, in: Proceedings of the 4th Latin American Symposium on Theoretical Informatics, LATIN '00, Springer-Verlag, 2000, pp. 88–94.

[19] H.N. Gabow, R.E. Tarjan, A linear-time algorithm for a special case of disjoint set union, J. Comput. Syst. Sci. 30 (2) (1985) 209–221.