Testing Planarity of Partially Embedded Graphs

PATRIZIO ANGELINI and GIUSEPPE DI BATTISTA, Roma Tre University, Italy
FABRIZIO FRATI, University of Sydney, Australia
VÍT JELÍNEK and JAN KRATOCHVÍL, Charles University, Prague, Czech Republic
MAURIZIO PATRIGNANI, Roma Tre University, Italy
IGNAZ RUTTER, Karlsruhe Institute of Technology (KIT), Germany and Charles University, Prague

We study the following problem: given a planar graph G and a planar drawing (embedding) of a subgraph of G, can such a drawing be extended to a planar drawing of the entire graph G? This problem fits the paradigm of extending a partial solution for a problem to a complete one, which has been studied before in many different settings. Unlike many cases, in which the presence of a partial solution in the input makes an otherwise easy problem hard, we show that the planarity question remains polynomial-time solvable. Our algorithm is based on several combinatorial lemmas, which show that the planarity of partially embedded graphs exhibits the 'TONCAS' behavior "the obvious necessary conditions for planarity are also sufficient." These conditions are expressed in terms of the interplay between (1) the rotation system and containment relationships between cycles and (2) the decomposition of a graph into its connected, biconnected, and triconnected components. This implies that no dynamic programming is needed for a decision algorithm and that the elements of the decomposition can be processed independently.

Further, by equipping the components of the decomposition with suitable data structures and by carefully splitting the problem into simpler subproblems, we make our algorithm run in linear time.

Finally, we consider several generalizations of the problem, such as minimizing the number of edges of the partial embedding that need to be rerouted to extend it, and argue that they are NP-hard. We also apply our algorithm to the simultaneous graph drawing problem Simultaneous Embedding with Fixed Edges (Sefe). There we obtain a linear-time algorithm for the case that one of the input graphs or the common graph has a fixed planar embedding.

Categories and Subject Descriptors: G.2.2 [**Algorithms**]

General Terms: Algorithms, Theory

Additional Key Words and Phrases: Planarity, partial embedding, simultaneous embedding, algorithm

---

A preliminary version of this article appeared as "Testing Planarity of Partially Embedded Graphs" in *Proceedings of the 21st ACM-SIAM Symposium on Discrete Algorithms (SODA'10)*, pages 202–221.
Work on the journal version of this article was supported by ESF EuroGIGA GraDR as Czech Research grant GACR GIG-11-E023 for V. Jelínek, J. Kratochvíl, and I. Rutter (partial support). I. Rutter was supported by a fellowship within the postdoctoral program of the German Academic Exchange Service (DAAD). F. Frati acknowledges support from the Australian Research Council (grant DE140100708). Research was supported in part by the MIUR project AMANDA (Algorithmics for Massive and Networked Data) protocol 2012C4E3KT_001.
Authors' addresses: P. Angelini, G. Di Battista, and M. Patrignani, Dipartimento di Ingegneria, Universita' Roma Tre, Via della Vasca Navale 79, 00146, Rome, Italy; emails: {angelini, gdb, patrigna}@dia.uniroma3.it; F. Frati, School of Information Technologies, The University of Sydney, NSW 2006, Sydney, Australia; email: fabrizio.frati@sydney.edu.au; V. Jelínek, IUUK MFF UK, Malostranske nam 25, 11800 Praha 1, Czech; email: jelinek@iuuk.mff.cuni.cz; J. Kratochvíl, KAM MFF UK, Malostranske nam 25, 11800 Praha 1, Czech Republic; email: honza@kam.mff.cuni.cz; I. Rutter, Karlsruhe Institute of Technology (KIT), Institute of Theoretical Informatics, Box 6980, D-76128, Karlsruhe, Germany; email: rutter@kit.edu.

DOI: http://dx.doi.org/10.1145/2629341

ACM Reference Format:
Patrizio Angelini, Giuseppe Di Battista, Fabrizio Frati, Vít Jelínek, Jan Kratochvíl, Maurizio Patrignani, and Ignaz Rutter. 2015. Testing planarity of partially embedded graphs. ACM Trans. Algor. 11, 4, Article 32 (April 2015), 42 pages.
DOI: http://dx.doi.org/10.1145/2629341

1. INTRODUCTION

Planarity is one of the central concepts not only in graph drawing but in graph theory as a whole. The characterization of planar graphs proved by Kuratowski [1930] represents a fundamental result in modern graph theory. This characterization, based on two forbidden topological subgraphs—K_5 and $K_{3,3}$—makes planarity a finite problem and leads to a polynomial-time recognition algorithm. Planarity is thus "simple" from the computational point of view (this, of course, does not mean that algorithms for testing planarity are trivial) in the strongest possible way, as several linear-time algorithms for testing planarity are known [Boyer and Myrvold 2004; de Fraysseix et al. 2006; Hopcroft and Tarjan 1974].

In this article, we pose and study the question of planarity testing in a constrained setting, namely when part of the input graph is already drawn and cannot be changed. Practical motivation for this question comes from the visualization of large networks in which certain patterns are required to be drawn in a standard way. The known planarity testing algorithms, even those that build a drawing incrementally, are of no help here, as they are allowed to redraw at each step the part of the graph processed so far. For similar reasons, online planar embedding and planarity testing algorithms, such as those of Di Battista and Tamassia [1996], Poutré [1994], Tamassia [1996], and Westbrook [1992], are not suitable to be used in this context.

Related work. The question of testing the planarity of partially drawn graphs fits into the general paradigm of extending a partial solution for a problem to a full one. This has been studied in various settings, and often the extendability problem is more difficult than the unconstrained one. As an example, graph coloring is NP-complete for perfect graphs even if only four vertices are already colored [Kratochvíl and Sebo 1997], whereas the chromatic number of a perfect graph can be determined in polynomial time [Grötschel et al. 1988]. Another example is provided by edge colorings—deciding 3-edge-colorability of cubic bipartite graphs if some edges are already colored is NP-complete [Fiala 2003], whereas it follows from the famous Kőnig-Hall theorem that cubic bipartite graphs are always 3-edge colorable. In view of these hardness results, it is somewhat surprising that the planarity of partially drawn graphs can be tested in polynomial time, in fact in linear time, as we show in this article. This is all the more so, considering that this problem is known to be NP-hard for drawings where edges are constrained to be straight-line segments [Patrignani 2006].

Specific constraints on planar graph drawings have been studied by several authors (e.g., see Dornheim [2002], Gutwenger et al. [2008], Tamassia [1998], and Tamassia et al. [1988]). However, none of those results can be exploited to solve the question that we pose in this article. The work in Juvan and Mohar [2005] and Mohar [1999] give algorithms for extending 2-cell embeddings on the torus and surfaces of higher genus. Their results show that even for arbitrary surfaces, the problem of extending an embedding of a graph $H \subseteq G$ to an embedding of G is fixed-parameter tractable with respect to the branch size of H. The *branch size* of a graph H is the size of the smallest graph H' from which H can be obtained as a subdivision. However, the approach of Juvan and Mohar is not applicable to our problem, as our goal is to find algorithms that are polynomial in the size of H as well as G. Moreover, the algorithm

by Juvan and Mohar assumes that either each component of $G - H$ has at most two allowed embeddings, which are given as part of the input, or H has a *closed* 2-cell embedding—that is, H is biconnected.

Contribution and outline. To solve the general problem, we allow disconnected graphs or graphs with low connectivity to be part of the input. It is readily seen that in this case the rotation system (i.e., the cyclic orderings of the edges incident to the vertices of the graph) does not fully describe the input. In fact, the relative position of vertices against cycles in the graph must also be considered. (These concepts and their technical details are discussed later.) Further, we make use of the fact that drawing graphs on the plane and on the sphere are equivalent concepts. The advantage of considering embeddings on the sphere lies in the fact that we do not need to distinguish between the outer face and the inner faces.

Many known planarity testing algorithms work by incrementally extending a partial drawing constructed in previous steps. The main idea of our algorithm is to look at the problem from the "opposite" perspective. Namely, we do not try to directly extend the input partial embedding (which seems much harder than one would expect). Instead, we look at the possible embeddings of the entire graph and decide if any of them contains the embedding of the subgraph prescribed by the input.

Our algorithm is based on several combinatorial lemmas, relating the problem to the connectivity of the graph. Most of them exhibit the TONCAS property—"the obvious necessary conditions are also sufficient." This is particularly elegant in the case of 2-connected graphs, in which we exploit the SPQR-tree decomposition of the graph. This notion was introduced by Di Battista and Tamassia [1996] to describe all possible embeddings of 2-connected planar graphs in a succinct way and has been used in various situations when asking for planar embeddings with special properties. A survey on the use of this technique in planar graphs is given by Mutzel [2003]. It is indeed obvious that if a 2-connected graph admits an embedding extending a given partial embedding, then the skeleton of each node of the SPQR-tree has a drawing *compatible* (a precise definition of compatibility will come later) with the partial embedding. We prove that the converse is also true. Hence, if we only aim at polynomial running time, we do not need to perform *any* dynamic programming on the SPQR-tree and could process its nodes independently. However, for the ultimate goal of linear running time, we must refine the approach and pass several pieces of information through the SPQR-tree. Then, dynamic programming becomes very useful. In addition, the SPQR-trees are exploited at two levels of abstraction, both for decomposing an entire block and for computing the embedding of the subgraph induced by each face of the constrained part of the drawing.

The article is organized as follows. We first describe the terminology and list auxiliary topological lemmas in Section 2. In particular, the combinatorial invariants of equivalent embeddings are introduced. In Section 3, we state the combinatorial characterization theorems for 2-connected, connected, and disconnected cases. These theorems yield a simple polynomial-time algorithm outlined at the end of the section. Section 4 is devoted to the technical details of the linear-time algorithm. Section 5 discusses several possible generalizations of the partially embedded planarity concept leading to NP-hard problems and shows how our techniques can be used to solve other graph drawing problems. We summarize our results and discuss some directions for further research in Section 6.

2. NOTATION AND PRELIMINARIES

In this section, we introduce some notations and preliminaries that we use throughout the article. In particular, we give a detailed description of how planar embeddings of

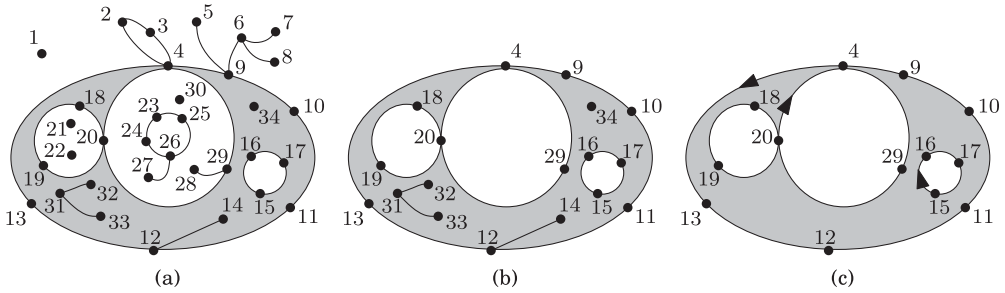

Fig. 1. (a) A planar drawing of a graph G. The shaded region represents a face f of the drawing. (b) The boundary of f. The circular lists defining the boundary of f are [15, 16, 17], [33, 31, 32, 31], [13, 12, 14, 12, 11, 10, 9, 4, 29, 20, 19, 18, 20, 4], [34]. (c) The facial cycles of f.

not necessarily connected graphs can be handled, and we give a first characterization of the embeddings extending given partial embeddings. We conclude with an overview of data structures and their efficient construction, which will be particularly important for the linear-time implementation of our algorithm.

The definitions listed in this section are standard and can be found in most graph theory textbooks. We are listing them for the sake of completeness. Perhaps less standard is the notion of H-bridges (first introduced by Demoucron et al. [1964] under the name *fragment*) and the definition of SPQR-trees (introduced in Di Battista and Tamassia [1996], we have followed an alternative wording used in the conference version of this paper [Angelini et al. 2010]).

2.1. Drawings, Embeddings, and the Problem Definition

A *drawing* of a graph is a mapping of each vertex to a distinct point of the plane and of each edge to a simple curve connecting its endpoints. A drawing is *planar* if the curves representing its edges do not intersect, except, possibly, at common endpoints. A graph is *planar* if it admits a planar drawing. A planar drawing Γ determines a subdivision of the plane into connected regions, called *faces*, and a circular ordering of the edges incident to each vertex, called *rotation system*. The circular ordering of the edges incident to a vertex x is the *(local) rotation of* x.

Visiting the (not necessarily connected) border of a face f of Γ in such a way to keep f to the left, we determine a set of circular lists of vertices. Such a set is the *boundary* of f. Two drawings are *equivalent* if they have the same rotation system and the same face boundaries. A *planar embedding* is an equivalence class of planar drawings. Note that equivalent planar drawings need not have the same outer face, and that a planar embedding does not determine which face is the outer face. This loss of information is harmless, as for the purposes of extending a partial embedding, the choice of the outer face is irrelevant. It is therefore convenient to imagine planar graphs as being embedded on a sphere, where no face plays the special role of outer face.

For connected graphs in the plane, an embedding is uniquely determined by the rotation system. For disconnected graphs, on the other hand, this information is not sufficient, as it does not determine the relative positions of the connected components. However, this additional information is encoded in the face boundaries, which, together with the rotation system, completely describe planar embeddings, even for disconnected graphs; Figure 1(a) and (b) provide an example.

Our initial motivation was to extend a given drawing of a subgraph of a planar graph to a planar drawing of the entire graph; however, it is not hard to see that this is equivalent to an embedding problem, where we wish to extend a planar embedding of a subgraph to a planar embedding of the whole graph.

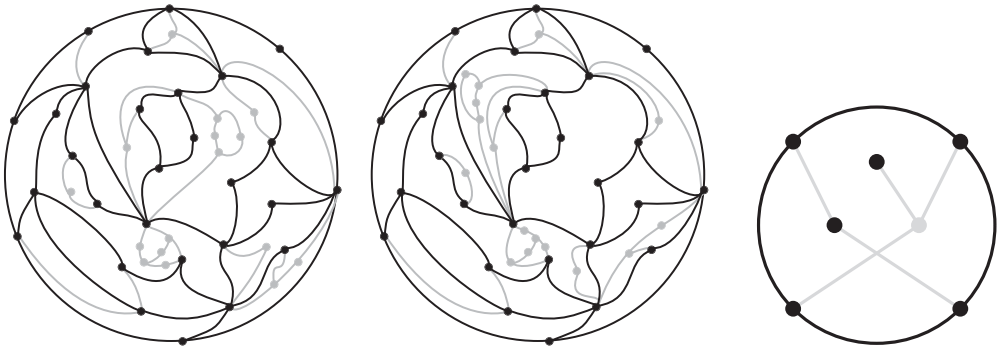

Fig. 2. Two different planar embeddings of a graph G whose restrictions to H (black vertices and edges) coincide with $\mathcal{H}$ (a, b). An instance that does not admit an embedding extension (c). Vertices and edges in $G \backslash H$ are grey.

A *partially embedded graph*, or PEG for short, is a triplet $(G, H, \mathcal{H})$, where G is a graph, H is a subgraph of G, and $\mathcal{H}$ is a planar embedding of H. We say that the vertices and edges of H are *prescribed*. The problem PARTIALLY EMBEDDED PLANARITY (PEP) asks whether a given PEG $(G, H, \mathcal{H})$ admits a planar embedding $\mathcal{G}$ of G whose restriction to H is $\mathcal{H}$. In this case, we say that the PEG $(G, H, \mathcal{H})$ is *planar*. We say that $\mathcal{G}$ is an *extension* of an embedding $\mathcal{H}$ of H if the restriction of $\mathcal{G}$ to H is $\mathcal{H}$. Figure 2 presents an example of a PEG that admits several different embedding extensions and an example that does not admit any.

2.2. Facial Cycles

Let Γ be a planar drawing of a graph H (see Figure 1(a)). Let $\vec{C}$ be a simple cycle in H with an arbitrary orientation. The oriented cycle $\vec{C}$ splits the plane into two connected parts. Denote by $V_\Gamma^{\text{left}}(\vec{C})$ and $V_\Gamma^{\text{right}}(\vec{C})$ the sets of vertices of the graph that are to the left and to the right of $\vec{C}$ in Γ, respectively. The boundary of each face f of Γ can be uniquely decomposed into simple edge-disjoint cycles, *bridges* (edges that are not part of any cycle), and isolated vertices see Figure 1(b)). Orient the cycles in such a way that f is to the left when walking along the cycle according to the orientation. Call these oriented cycles the *facial cycles* of f (see Figure 1(c)). Observe that the sets $V_\Gamma^{\text{left}}(\vec{C})$, $V_\Gamma^{\text{right}}(\vec{C})$, and the notion of facial cycles only depend on the embedding $\mathcal{H}$ of Γ. Hence, it makes sense to write $V_\mathcal{H}^{\text{left}}(\vec{C})$ and $V_\mathcal{H}^{\text{right}}(\vec{C})$, and to define the facial cycles of $\mathcal{H}$ as the facial cycles of the faces of $\mathcal{H}$.

For a vertex x of a graph G with embedding $\mathcal{G}$, we denote by $E_G(x)$ the set of edges incident to x and by $\sigma_\mathcal{G}(x)$ the (local) rotation of x in $\mathcal{G}$. The following lemma characterizes the planar embeddings of a PEG $(G, H, \mathcal{H})$ that extend $\mathcal{H}$ in terms of the rotation system and relative cycle–vertex positions with respect to the facial cycles of $\mathcal{H}$.

LEMMA 2.1. *Let $(G, H, \mathcal{H})$ be a* PEG *and let $\mathcal{G}$ be a planar embedding of G. The restriction of $\mathcal{G}$ to H is $\mathcal{H}$ if and only if the following conditions hold:*

(*1*) *for every vertex $x \in V(H)$, $\sigma_\mathcal{G}(x)$ restricted to $E_H(x)$ coincides with $\sigma_\mathcal{H}(x)$, and*
(*2*) *for every facial cycle $\vec{C}$ of each face of $\mathcal{H}$, we have that $V_\mathcal{H}^{\text{left}}(\vec{C}) \subseteq V_\mathcal{G}^{\text{left}}(\vec{C})$ and $V_\mathcal{H}^{\text{right}}(\vec{C}) \subseteq V_\mathcal{G}^{\text{right}}(\vec{C})$.*

PROOF. The proof follows easily from the following statement. Let Γ_1 and Γ_2 be two drawings of the same graph G such that for every vertex $x \in V(G)$, $\sigma_{\Gamma_1}(x) = \sigma_{\Gamma_2}(x)$ holds. Assume that each facial cycle $\vec{C}$ for any face f in Γ_1 or in Γ_2 is oriented in such a way that f is to the left of $\vec{C}$. Drawings Γ_1 and Γ_2 are equivalent if and only if they have the same oriented facial cycles and, for each oriented facial cycle $\vec{C}$, it holds $V^{\text{left}}_{\Gamma_1}(\vec{C}) = V^{\text{left}}_{\Gamma_2}(\vec{C})$.

We need to prove this statement in both directions: (i) if Γ_1 and Γ_2 have the same embedding, then they have the same oriented facial cycles and, for each facial cycle $\vec{C}$, we have $V^{\text{left}}_{\Gamma_1}(\vec{C}) = V^{\text{left}}_{\Gamma_2}(\vec{C})$, and (ii) if Γ_1 and Γ_2 have the same oriented facial cycles and, for each facial cycle $\vec{C}$, we have $V^{\text{left}}_{\Gamma_1}(\vec{C}) = V^{\text{left}}_{\Gamma_2}(\vec{C})$, then Γ_1 and Γ_2 have the same embedding.

We start with direction (i). By definition, drawings with the same embedding have the same facial boundaries and hence the same oriented facial cycles. Suppose for a contradiction that for some facial cycle $\vec{C}$, $V^{\text{left}}_{\Gamma_1}(\vec{C}) \neq V^{\text{left}}_{\Gamma_2}(\vec{C})$. Then, at least one vertex v is to the left of $\vec{C}$ in Γ_1 and to the right of $\vec{C}$ in Γ_2 (the opposite case being analogous). Hence, v is part of the boundary of a face that is to the left of $\vec{C}$ in Γ_1 and part of the boundary of a face that is to the right of $\vec{C}$ in Γ_2, contradicting the hypothesis that Γ_1 and Γ_2 have the same facial boundaries.

We now come to the proof of direction (ii). First, suppose that G is connected and has at least one vertex of degree 3. In this case, the fact that Γ_1 and Γ_2 have the same rotation system implies that they also have the same face boundaries and, hence, the same embedding. Second, suppose that G is connected and has maximum degree 2. Then, G is either a path or a cycle. In both cases, the face boundaries of Γ_1 and Γ_2 are the same (recall that G is drawn on the sphere). Finally, suppose that G has several connected components $C_1, C_2, \ldots, C_k$. We say that two components C_i and C_j *share a face* f if there exists a vertex of C_i and a vertex of C_j on the boundary of f. The drawings Γ_1 and Γ_2 have the same face boundaries if (a) for each C_i, with $i = 1, \ldots, k$, the embedding $\mathcal{G}_1$ of G in Γ_1 restricted to C_i is the same as the embedding $\mathcal{G}_2$ of G in Γ_2 restricted to C_i, and (b) each pair of connected components C_i and C_j, with $i, j \in \{1, \ldots, k\}$ and $i \neq j$, either do not share a face both in $\mathcal{G}_1$ and in $\mathcal{G}_2$ or they contribute with the same circular lists to the boundary of the same face f in $\mathcal{G}_1$ and in $\mathcal{G}_2$.

It can be seen that condition (a) is satisfied by applying the argument we made for a connected graph to each connected component of G.

Condition (b) follows from the hypothesis that for each oriented facial cycle $\vec{C}$, we have $V^{\text{left}}_{\Gamma_1}(\vec{C}) = V^{\text{left}}_{\Gamma_2}(\vec{C})$. Suppose, for a contradiction, that two connected components C_x and C_y of G share a face f in $\mathcal{G}_1$ and no face in $\mathcal{G}_2$. Since C_x and C_y share a face in $\mathcal{G}_1$, they are on the same side of any facial cycle $\vec{C}$ belonging to any other component C_z of G (more intuitively, C_x and C_y are not separated by any cycle and in particular by any facial cycle in Γ_1). On the other hand, since C_x and C_y do not share a face in Γ_2, there exists a component C_z of G containing a facial cycle $\vec{C}$ separating C_x from C_y, thus contradicting the hypothesis that $V^{\text{left}}_{\Gamma_1}(\vec{C}) = V^{\text{left}}_{\Gamma_2}(\vec{C})$.

Next suppose, for a contradiction, that two connected components C_x and C_y contribute with circular lists L^x_1 and L^y_1 to the boundary of the same face f_1 of $\mathcal{G}_1$ and with circular lists L^x_2 and L^y_2 to the boundary of the same face f_2 of $\mathcal{G}_2$ and suppose that $L^x_1 \neq L^x_2$. In particular, assume, without loss of generality, that there exists a facial cycle $\vec{C}'$ of f_2 that is part of C_x and that is not a facial cycle of f_1. The boundary of f_1 is

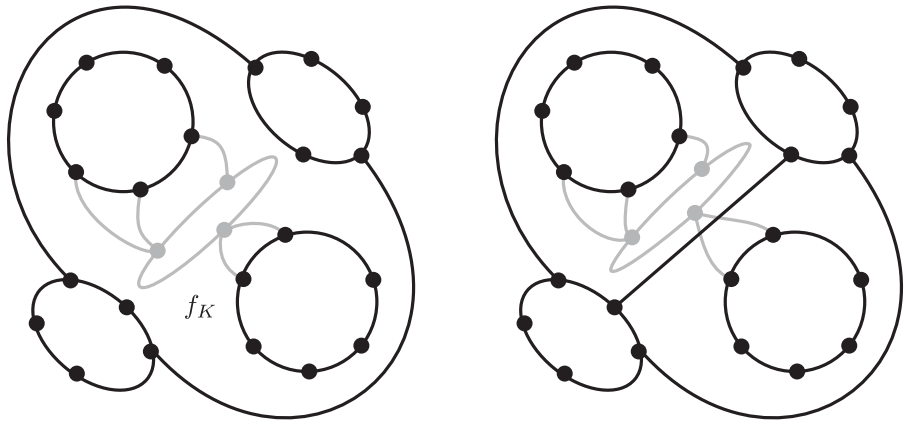

Fig. 3. A nonlocal bridge is either necessarily contained in a face f_K (a) or causes a nonplanarity (b).

oriented in such a way that every facial cycle of f_1 has f_1 to its left. Then, every facial cycle of f_1 obtained from L_1^x has C_y to its left. Further, there exists a facial cycle $\vec{C}$ of f_1 obtained from L_1^x that has $\vec{C}'$ to its right (part of $\vec{C}$ and of $\vec{C}'$ may coincide). As $\mathcal{G}_1$ and $\mathcal{G}_2$ restricted to C_x give the same embedding, the last statement is true both in $\mathcal{G}_1$ and in $\mathcal{G}_2$. Since $\vec{C}'$ is incident to f_2 and since C_y is incident to f_2, such a component is to the right of $\vec{C}$, contradicting the hypothesis that $V_{\Gamma_1}^{\text{left}}(\vec{C}) = V_{\Gamma_2}^{\text{left}}(\vec{C})$. □

2.3. Connectivity, *H*-Bridges, and Data Structures

A graph is *connected* if every pair of vertices is connected by a path. A *k-connected* graph G is such that removing any at most $k-1$ vertices leaves G connected; 3-connected, 2-connected, and 1-connected graphs are also called *triconnected*, *biconnected*, and *simply connected* graphs, respectively. By convention, the complete graph on k vertices is $(k-1)$-connected but not k-connected. A *separating k-set* is a set of k vertices whose removal disconnects the graph. Separating 1- and 2-sets are called *cutvertices* and *separation pairs*, respectively. Hence, a connected graph is biconnected if it has no cutvertices, and it is triconnected if it has no separation pairs and no cutvertices. A *block* of G is either a maximal biconnected subgraph of G or a bridge in G. Each edge of G falls into a single block of G, whereas cutvertices are shared by different blocks. We extend the notion of k-connectivity to PEGs by saying that a PEG $(G, H, \mathcal{H})$ is k-connected if and only if G is k-connected.

Let G be a graph and let H be a subgraph of G. An *H-bridge* K of G is a subgraph of G formed either by a single edge $e \in E(G)\backslash E(H)$ whose end-vertices belong to H or by a connected component K^- of $G - V(H)$, together with all edges (and their end-vertices) that connect a vertex in K^- to vertices in H. In the first case, the H-bridge is *trivial*. A vertex that belongs to $V(H) \cap V(K)$ is called an *attachment vertex* (or *attachment*) of K. Note that the edge sets of the H-bridges form a partition of $E(G)\backslash E(H)$.

An H-bridge K is *local* to a block B of H if all attachments of K belong to B. Notice that an H-bridge with a single attachment can be local to more than one block, whereas an H-bridge with at least two attachments is local to at most one block. An H-bridge that is not local to any block of H is *nonlocal*.

Note that for a nonlocal H-bridge K, there exists at most one face of $\mathcal{H}$ containing all attachments of K (Figure 3(a)). Namely, if all attachments of K were contained on the boundaries of two distinct faces of $\mathcal{H}$, then K would necessarily be local. In addition, if

there is no face of $\mathcal{H}$ incident to all attachments of K, then G clearly has no embedding extension (see Figure 3(b)).

Let $(G, H, \mathcal{H})$ be a PEG. In the following, we define some data structures that are widely used throughout the article. All of these data structures can easily be computed in time linear in the number of edges of the graph or of the embedding to which they refer. We use the decomposition of a graph G into its connected, biconnected, and triconnected components. To further relate these decompositions with the embedding of $\mathcal{H}$, we make use of several auxiliary data structures.

The *component–face tree* $\mathcal{CF}$ of $\mathcal{H}$ is a tree whose nodes are the connected components of H and the faces of $\mathcal{H}$. A face f and a component C are joined by an edge if a vertex of C is incident to f. The *block–face tree* $\mathcal{BF}$ of $\mathcal{H}$ is a tree whose nodes are the blocks of H and the faces of $\mathcal{H}$. A face f and a block B are joined by an edge if B contains an edge incident to f. The *vertex–face incidence graph* $\mathcal{VF}$ of $\mathcal{H}$ is a graph whose nodes are the vertices of H and the faces of $\mathcal{H}$. A vertex v and a face f are joined by an edge if v appears on the boundary of f.

To handle the decomposition of a graph into biconnected components, we use the *block-cutvertex tree*. The block-cutvertex tree of a connected graph G is a tree whose nodes are the blocks and the cutvertices of G. Edges in the block-cutvertex tree join each cutvertex to the blocks to which it belongs. The *enriched block-cutvertex tree* of G is a tree obtained by adding to the block-cutvertex tree of G each vertex v of G that is not a cutvertex and by connecting v to the unique block to which it belongs.

To handle the decomposition of a graph into its triconnected components, we use the SPQR-tree $\mathcal{T}$ of G, which we describe in the following.

Let G be a graph. A *split pair* of G is either a separation pair or a pair of adjacent vertices. A *maximal split component*[1] of G with respect to a split pair $\{u, v\}$ (or, simply, a maximal split component of $\{u, v\}$) is either an edge (u, v) or a maximal subgraph G' of G such that G' contains u and v, and $\{u, v\}$ is not a split pair of G'. A vertex $w \neq u, v$ belongs to exactly one maximal split component of $\{u, v\}$. A *split component* of $\{u, v\}$ is the union of any number of maximal split components of $\{u, v\}$.

The SPQR-tree $\mathcal{T}$ of a biconnected graph G is a data structure that describes a recursive decomposition of G induced by its split pairs. The nodes of $\mathcal{T}$ are of four types: S, P, Q, and R. The Q-nodes are the leaves of the tree $\mathcal{T}$. Each Q-node represents a unique edge of the graph G.

Each node μ of $\mathcal{T}$ has an associated biconnected multigraph, called the *skeleton of* μ and denoted by skel(μ). The skeleton describes a decomposition of G into edge-disjoint split components. The edges of the skeleton are called *virtual edges*.

For an internal node μ of $\mathcal{T}$ of degree d, the skeleton skel(μ) has d virtual edges $e_1, e_2, \ldots, e_d$, which correspond bijectively to the connected components of $\mathcal{T} - \mu$. Let $\mathcal{T}_i$ be the component of $\mathcal{T} - \mu$ corresponding to e_i, and let G_i be the subgraph of G formed by all edges of G whose Q-nodes belong to $\mathcal{T}_i$. The graph G_i is called *the expansion graph* of e_i. Each expansion graph G_i is a split component of G, and by replacing each virtual edge e_i in skel(μ) with its expansion graph G_i, we obtain the graph G.

For algorithmic purposes, it is often convenient to treat $\mathcal{T}$ as a rooted tree. In such a case, we choose an arbitrary Q-node as a root of $\mathcal{T}$. In the skeleton of any nonroot node μ, there is a unique virtual edge representing the component of $\mathcal{T} - \mu$ that contains the root. This virtual edge is the *parent edge* of skel(μ). Supposing that μ has k children, let $e_1, \ldots, e_k$ be the nonparent edges of skel(μ), and let $G_1, \ldots, G_k$ be their expansion graphs. The graph $G^* = G_1 \cup G_2 \cup \cdots \cup G_k$ is the *pertinent graph of* μ, denoted by pert(μ).

[1]Note that "maximal" refers to the splitting (the component cannot be split further by $\{u, v\}$), not the size, of the component. We use the term for consistency with existing literature.

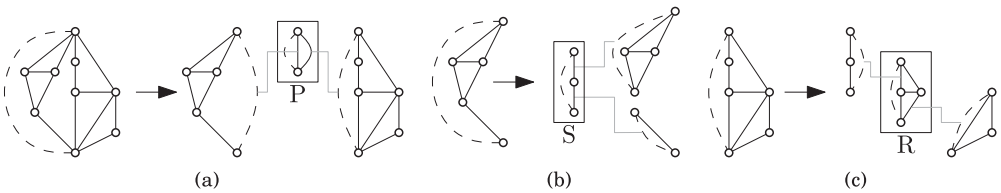

Fig. 4. Different cases in the construction of an SPQR-tree: parallel case (a), series case (b), and rigid case (c). The skeletons are shown in a box with the parent edge dashed. The dashed curves in the graphs represent the split pair with respect to which the graphs are decomposed; the procedure decomposes the graph without the dashed edge. The grey curves show the correspondence between the virtual edges of the skeleton and the corresponding split pairs in the children.

If μ has no children—that is, μ is a nonroot Q-node—then pert(μ) consists of the single edge represented by the node μ.

Note that the expansion graph of the parent edge of skel(μ) contains precisely those edges of G that do not belong to pert(μ). Note also that if ν is the parent of μ, then skel(ν) has a virtual edge whose expansion graph is pert(μ).

To give a precise definition of the SPQR-tree, we present an algorithm that recursively builds $\mathcal{T}$, starting from the root and then at each step adding a new child to a previously constructed node. Given a biconnected graph G and an edge $e = (u', v')$ of G, the algorithm proceeds as follows. At each recursive step, a split pair $\{u, v\}$ of G, a split component G^* of $\{u, v\}$, and a previously constructed node ν of $\mathcal{T}$ are given. A new node μ with pertinent graph G^* is added to $\mathcal{T}$ and attached as a child to ν. The vertices u and v are the *poles* of μ and are denoted by $u(\mu)$ and $v(\mu)$, respectively. The algorithm then possibly recurs on some split components of G^*.

At the beginning, the Q-node representing the edge $e = \{u', v'\}$ is designated as the root of $\mathcal{T}$. The algorithm then recurs with $G^* = G - \{e\}$, $\{u, v\} = \{u', v'\}$, and ν being the root. The recursive step distinguishes some cases, which are illustrated in Figure 4:

Base case: If G^* consists of exactly one edge between u and v, then μ is a Q-node whose skeleton is a cycle of length 2.

Parallel case: If G^* is composed of at least two maximal split components $G_1, \ldots, G_k$ ($k \geq 2$) of G with respect to $\{u, v\}$, then μ is a P-node. The graph skel(μ) consists of k parallel virtual edges between u and v, denoted by $e_1, \ldots, e_k$ and corresponding to $G_1, \ldots, G_k$, respectively, plus an additional parent edge (u, v). The decomposition recurs on $G_1, \ldots, G_k$, with $\{u, v\}$ as poles for every graph, and with μ as parent node.

Series case: If G^* is composed of exactly one maximal split component of G with respect to $\{u, v\}$ and if G^* has cutvertices $c_1, \ldots, c_{k-1}$ ($k \geq 2$), appearing in this order on a path from u to v, then μ is an S-node. The graph skel(μ) is the cycle composed of a path $e_1, \ldots, e_k$ of virtual edges, where e_1 connects u with c_1, e_i connects c_{i-1} with c_i ($i = 2, \ldots, k-1$), and e_k connects c_{k-1} with v, plus a parent edge (u, v). The decomposition recurs on the split components corresponding to each of $e_1, e_2, \ldots, e_{k-1}, e_k$ with μ as parent node and with $\{u, c_1\}, \{c_1, c_2\}, \ldots, \{c_{k-2}, c_{k-1}\}, \{c_{k-1}, v\}$ as poles, respectively.

Rigid case: If none of the preceding cases applies, the purpose of the decomposition step is that of partitioning G^* into the minimum number of split components and recurring on each of them. We need some further definitions. Given a maximal split component G' of a split pair $\{s, t\}$ of G^*, a vertex $w \in G'$ *properly belongs* to G' if $w \neq s, t$. Given a split pair $\{s, t\}$ of G^*, a maximal split component G' of $\{s, t\}$ is *internal* if neither u nor v (the poles of G^*) properly belongs to G', *external* otherwise. A *maximal split pair* $\{s, t\}$ of G^* is a split pair of G^* that is not

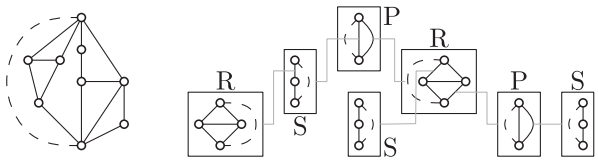

Fig. 5. The example graph of Figure 4 and its SPQR-tree with respect to the dashed reference edge. For clarity, Q-nodes are omitted. The parent edge of each skeleton is dashed, and each skeleton is connected by a grey curve to the virtual edge in the parent that represents it.

contained in an internal maximal split component of any other split pair $\{s', t'\}$ of G^*. Let $\{u_1, v_1\}, \ldots, \{u_k, v_k\}$ be the maximal split pairs of G^* ($k \geq 1$) that have at least one internal split component, and for $i = 1, \ldots, k$, let G_i be the union of all internal maximal split components of $\{u_i, v_i\}$. Observe that each vertex of G^* either properly belongs to exactly one G_i or belongs to some maximal split pair $\{u_i, v_i\}$. Node μ is an R-node. Graph skel(μ) is the graph obtained from G^* by replacing each subgraph G_i with the virtual edge e_i between u_i and v_i and by adding a parent edge (u, v). The decomposition recurs on each G_i with μ as parent node and with $\{u_i, v_i\}$ as poles. It can be shown that the skeleton of an R-node is a triconnected graph.

Figure 5 illustrates an example of the construction of the SPQR-tree of a biconnected graph G. The SPQR-tree $\mathcal{T}$ of an n-vertex biconnected graph G is well suited for the implementation of efficient algorithms, as $\mathcal{T}$ can be computed in linear time [Gutwenger and Mutzel 2000], it has $O(n)$ nodes, and the total size of all skeletons of all nodes of $\mathcal{T}$ is also $O(n)$ [Bertolazzi et al. 2000]. We say that an edge e of G *projects* to a virtual edge e' (or *belongs* to e') of skel(μ), for some node μ in $\mathcal{T}$, if e belongs to the expansion graph of e'.

The SPQR-tree $\mathcal{T}$ can be used to represent all planar embeddings of G. In the first part of the article, for our characterization, we will use the unrooted version of the SPQR-tree. In the second part of the article, for the linear-time implementation, we will then root the tree to allow for dynamic programming on the tree in a bottom-up fashion.

We emphasize the following properties, which are implicitly exploited throughout the article.

PROPERTY 1. *A planar embedding of the skeleton of every node of $\mathcal{T}$ determines a planar embedding of G and vice versa.*

PROPERTY 2. *Let C be a cycle of G and let μ be any node of $\mathcal{T}$. Then, either the edges of C belong to a single virtual edge of* skel(μ) *or they belong to a set of virtual edges that induce a cycle in* skel(μ).

Efficient computation of data structures. We now briefly discuss the time complexity of constructing the introduced data structures. We further show how to implement the basic queries used in our algorithms in constant time per operation.

First, observe that linear-time preprocessing can associate each edge of a planar graph with the unique connected component to which it belongs, with the unique block to which it belongs, and (given a planar embedding of the graph) with the (at most) two faces to which it is incident. Additionally, we can associate each vertex of a graph with the unique connected component to which it belongs.

The block-cutvertex tree of a connected planar graph [Tarjan 1972] and the SPQR-tree of a biconnected planar graph [Gutwenger and Mutzel 2000] can be constructed in linear time. The enriched block-cutvertex tree of a connected planar graph G can be

constructed starting from the block-cutvertex tree of G by adding to the tree (i) each vertex v that is not a cutvertex of G, and (ii) an edge between v and the only block to which it belongs.

The block-face tree $\mathcal{BF}$ of a planar embedding $\mathcal{G}$ of a planar graph G can be constructed in linear time. Namely, for each edge e of G, let B_e be the unique block of G containing e and let f'_e and f''_e be the two faces of $\mathcal{G}$ adjacent to e (possibly $f'_e = f''_e$). Add edges (f'_e, B_e) and (f''_e, B_e) to $\mathcal{BF}$. When all edges of G have been considered, the resulting multigraph $\mathcal{BF}$ has a linear number of edges. Remove multiple edges as follows. Root $\mathcal{BF}$ at any node and orient $\mathcal{BF}$ so that all edges point toward the root. Remove all edges exiting from each node, except for one, thus obtaining the block-face tree $\mathcal{BF}$ of $\mathcal{G}$. The component-face tree $\mathcal{CF}$ of a planar embedding $\mathcal{G}$ of a planar graph G can be constructed analogously in linear time.

The vertex-face incidence graph $\mathcal{VF}$ of a planar embedding $\mathcal{G}$ of a planar graph G can be constructed in linear time by processing faces of $\mathcal{G}$ one by one, where for each face f we walk along the boundary of f and add to $\mathcal{VF}$ edges between f and the vertices on the boundary. To avoid adding multiple edges, we remember, for each vertex x, the last face f that has been connected to x in $\mathcal{VF}$. Note that $\mathcal{VF}$ is a planar graph.

Kowalik and Kurowski [2003] have shown that for a given planar graph F and a fixed integer k, it is possible to build in linear time a "short-path" data structure that allows checking in constant time whether two given vertices of F are connected by a path of length at most k and returning such a path if it exists. We will employ this data structure to search for paths of lengths 1 and 2 in our auxiliary graphs. Using this data structure, we can, for example, determine in constant time whether two vertices share a common face in $\mathcal{H}$ (by finding a path of length two in the vertex-face incidence graph $\mathcal{VF}$) or whether they share the same block (by finding a path of length 2 in the enriched block-cutvertex tree).

Efficient computation of local and nonlocal H-bridges. We now describe how these data structures can be used to solve the following problem. Given an instance $(G, H, \mathcal{H})$ of Pep and an H-bridge K of G, determine whether K is local or not and, in the latter case, compute the unique face f of $\mathcal{H}$ in which K has to be embedded in any solution of $(G, H, \mathcal{H})$. In the following lemma, we show how to solve this problem in time linear in the size of K.

Lemma 2.2. *Let $(G, H, \mathcal{H})$ be any instance of* Pep. *Assume that we are given the component-face tree $\mathcal{CF}$ of $\mathcal{H}$, the vertex-face incidence graph $\mathcal{VF}$ of $\mathcal{H}$, the block-face tree $\mathcal{BF}$ of $\mathcal{H}$, and, for each connected component C_i of H, the enriched block-cutvertex tree $\mathcal{B}_i^+$ of C_i. Suppose further that all of these graphs are endowed with short-path data structures to check for paths of length 1 and 2. Let K be an H-bridge of G. There is an algorithm that checks whether K is local to any block of H in time linear in the size of K. Furthermore, if K is nonlocal, the algorithm computes the only face of $\mathcal{H}$ incident to all attachment vertices of K, if such a face exists, in time linear in the size of K.*

Proof. Consider the attachment vertices $a_1, a_2, \ldots, a_h$ of K. If $h = 1$, then K is local. Otherwise, $h \geq 2$. To decide whether K is local for some block of H, we perform the following check. Consider the attachment vertices a_1 and a_2. If a_1 and a_2 belong to distinct connected components, then K is not local to any block. Otherwise, they belong to the same connected component C_i. Check whether a_1 and a_2 have distance 2 in $\mathcal{B}_i^+$—that is, whether they belong to the same block B. This is done in constant time by querying the short-path structure. If the check fails, then K is not local to any block. Otherwise, B contains both a_1 and a_2. In the latter case, check whether B is also adjacent in $\mathcal{B}_i^+$ to all other attachment vertices $a_3, \ldots, a_h$ of K. Again, each such a check is performed in constant time. If the test succeeds, then K is local to block B.

Otherwise, there exists a vertex a_j, with $3 \leq j \leq h$, that is not incident to B, and K is not local to any block.

If K is nonlocal, we compute the unique face f of $\mathcal{H}$ to which all attachment vertices of K are incident. First, we choose two attachment vertices a_p and a_q, with $1 \leq p, q \leq h$, that do not belong to the same block. If a_1 and a_2 do not belong to the same block, then we take $a_p = a_1$ and $a_q = a_2$. If the check failed on an attachment vertex a_j in $a_3, \ldots, a_h$, then either a_1 and a_j or a_2 and a_j do not belong to the same block. In the former case, set $a_p = a_1$ and $a_q = a_j$; in the latter one, set $a_p = a_2$ and $a_q = a_j$. By querying the short-path data structure, we determine in constant time whether a_p and a_q are connected by a path of length 2 in $\mathcal{V}F$ and find the middle vertex of such a path. This middle vertex corresponds to the unique common face f of a_p and a_q. We then check whether all attachments of K are adjacent to f in $\mathcal{V}F$. If the test fails, then no face of $\mathcal{H}$ contains all attachments of K. Otherwise, f is the only face of $\mathcal{H}$ whose boundary contains all attachments of K. □

3. COMBINATORIAL CHARACTERIZATION

We first present a combinatorial characterization of planar PEGs. This not only forms a basis of our algorithm but also is interesting in its own right, as it shows that a PEG has an embedding extension if and only if it satisfies simple conditions that are obviously necessary for an embedding extension to exist.

Our characterization is based on a decomposition of the graph G of a PEG $(G, H, \mathcal{H})$ into its connected, biconnected and triconnected components. For triconnected PEGs, the problem is particularly easy. For a triconnected PEG $(G, H, \mathcal{H})$, the graph G has only two distinct planar embeddings: $\mathcal{G}_1$ and $\mathcal{G}_2$. The PEG is thus planar if and only if either $\mathcal{G}_1$ or $\mathcal{G}_2$ extends $\mathcal{H}$. Clearly, for a disconnected PEG to admit an embedding extension, it is a necessary condition that each of its connected components admits an embedding extension. Similarly, it is a necessary condition for a connected PEG that each biconnected component admits an embedding extension. We start with the most specific case—the case where G is biconnected—and then extend the characterization to the cases where G is connected or even disconnected.

3.1. Planarity of Biconnected PEGs

In this section, we focus on biconnected PEGs $(G, H, \mathcal{H})$. This assumption allows us to use the SPQR-tree $\mathcal{T}$ of G as the main tool of our characterization, which is based on the two necessary and sufficient conditions of Lemma 2.1. We show that they can be individually translated to constraints on the embeddings of the skeletons of $\mathcal{T}$.

Definition 3.1. A planar embedding of the skeleton of a node μ of the SPQR-tree of G is *edge-compatible with* $\mathcal{H}$ if, for every vertex x of skel(μ), and for every three edges of $E_H(x)$ belonging to different virtual edges of skel(μ), their clockwise order determined by the embedding of skel(μ) is a suborder of $\sigma_{\mathcal{H}}(x)$.

LEMMA 3.2. *Let* $(G, H, \mathcal{H})$ *be a biconnected* PEG. *Let* $\mathcal{T}$ *be the SPQR-tree of* G. *An embedding* $\mathcal{G}$ *of* G *satisfies condition 1 of Lemma 2.1 if and only if for each node* μ *of* $\mathcal{T}$, *the corresponding embedding of* skel(μ) *is edge-compatible with* $\mathcal{H}$.

PROOF. Obviously, if G has an embedding satisfying condition 1 of Lemma 2.1, then the corresponding embedding of skel(μ) is edge-compatible with $\mathcal{H}$ for each node μ of $\mathcal{T}$.

To prove the converse, assume that the skeleton of every node of $\mathcal{T}$ has an embedding that is edge-compatible with $\mathcal{H}$, and let $\mathcal{G}$ be the embedding of G determined by all such skeleton embeddings. We claim that $\mathcal{G}$ satisfies condition 1 of Lemma 2.1. To prove the claim, it suffices to show that any three edges e, f, and g of $\mathcal{H}$ that share a common

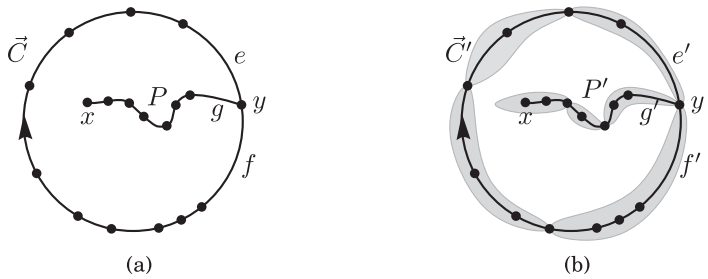

Fig. 6. Illustration for the proof of Lemma 3.4. (a) Path P, vertices x and y, and edges e, f, and g in $\mathcal{G}$. (b) Path or cycle P' and edges e', f', and g' of skel(μ). Grey regions represent virtual edges of the skeleton of a node of $\mathcal{T}$.

vertex x appear in the same clockwise order around x in $\mathcal{H}$ and in $\mathcal{G}$. Assume that the triple (e, f, g) is embedded in clockwise order around x in $\mathcal{H}$. Let μ be the node of $\mathcal{T}$ with the property that the Q-nodes representing e, f, and g appear in distinct components of $\mathcal{T} - \mu$. Note that such a node μ exists and is unique. The three edges e, f, and g project into three distinct virtual edges e', f', and g' of skel(μ). Since the embedding of skel(μ) is assumed to be edge-compatible with $\mathcal{H}$, the triple (e', f', g') is embedded in clockwise order in skel(μ), and hence the triple (e, f, g) is embedded in clockwise order in $\mathcal{G}$. □

Lemma 3.2 settles the translation of condition 1 of Lemma 2.1 to conditions on the embeddings of the skeletons of the SPQR-tree of G. Next, we deal with condition 2. Consider a simple cycle $\vec{C}$ of G with an arbitrary orientation and a node μ of the SPQR-tree of G. By Property 2, either all edges of $\vec{C}$ belong to the expansion graph of a single virtual edge of skel(μ) or the virtual edges whose expansion graphs contain the edges of $\vec{C}$ form a simple cycle in skel(μ). Such a cycle in skel(μ) inherits the orientation of $\vec{C}$ in a natural way.

Definition 3.3. A planar embedding of the skeleton of a node μ of the SPQR-tree of G is *cycle-compatible with* $\mathcal{H}$ if for every facial cycle $\vec{C}$ of $\mathcal{H}$ whose edges project to a simple cycle $\vec{C}'$ in skel(μ), all vertices of skel(μ) that belong to $V_{\mathcal{H}}^{\text{left}}(\vec{C})$ and all virtual edges that contain vertices of $V_{\mathcal{H}}^{\text{left}}(\vec{C})$ (except for the virtual edges of $\vec{C}'$ itself) are embedded to the left of $\vec{C}'$, and analogously for $V_{\mathcal{H}}^{\text{right}}(\vec{C})$.

LEMMA 3.4. *Let* $(G, H, \mathcal{H})$ *be a biconnected* PEG. *Let* $\mathcal{T}$ *be the SPQR-tree of* G. *An embedding* $\mathcal{G}$ *of* G *satisfies condition 2 of Lemma 2.1 if and only if for each node* μ *of* $\mathcal{T}$, *the corresponding embedding of* skel(μ) *is cycle-compatible with* $\mathcal{H}$.

PROOF. Obviously, if $\mathcal{G}$ is an embedding of G that satisfies condition 2 of Lemma 2.1, then the corresponding embedding of skel(μ) is cycle-compatible with $\mathcal{H}$ for each node μ of $\mathcal{T}$.

To prove the converse, assume that skel(μ) has an embedding that is cycle-compatible with $\mathcal{H}$ for each node μ of $\mathcal{T}$, and let $\mathcal{G}$ be the resulting embedding of G.

Our goal is to show that for every facial cycle $\vec{C}$ of $\mathcal{H}$ and for every vertex x of $H - V(\vec{C})$, the left/right position of x with respect to $\vec{C}$ is the same in $\mathcal{H}$ as in $\mathcal{G}$.

Refer to Figure 6(a). Assume that x is to the right of $\vec{C}$ in $\mathcal{G}$ (the other case being analogous). Let P be a shortest path in G that connects x to a vertex of $\vec{C}$. Such a path exists since G is connected. Let y be the vertex of $\vec{C} \cap P$, and let e and f be the two edges of $\vec{C}$ adjacent to y, where e directly precedes f in the orientation of $\vec{C}$. By the

minimality of P, all vertices of $P - y$ avoid $\vec{C}$, and hence all vertices of $P - y$ are to the right of $\vec{C}$ in $\mathcal{G}$. Let g be the edge of P adjacent to y. In $\mathcal{G}$, the triple (e, f, g) appears in clockwise order around y.

Refer to Figure 6(b). Let μ be the (unique) internal node of $\mathcal{T}$ in which e, f, and g project to distinct edges e', f', and g' of skel(μ). Let $\vec{C}'$ be the projection of $\vec{C}$ into skel(μ) (i.e., $\vec{C}'$ is the subgraph of skel(μ) formed by edges that contain the projection of at least one edge of $\vec{C}$), and let P' be the projection of P. It is easy to see that $\vec{C}'$ is a cycle of length at least 2, whereas P' is either a path or a cycle. Note that the latter case only happens when P' has at least two edges and the vertex x properly belongs to a virtual edge d' of skel(μ) incident with y. Assume that the edges of $\vec{C}'$ are oriented consistently with the orientation of $\vec{C}$ and that the edges of P' form an ordered sequence, where the edge containing x is the first and g' is the last.

Both the endpoints of an edge of $\vec{C}'$ are vertices of $\vec{C}$. Analogously, both the endpoints of an edge of P' are vertices of P, with the possible exception of the first vertex of P'. It follows that no vertex of P' belongs to $\vec{C}'$, except possibly for the first one and the last one. Thus, no edge of P' belongs to $\vec{C}'$, and by the assumption that the embedding of skel(μ) is planar and that $\mathcal{G}$ is the embedding resulting from the skeleton embedding choices, all edges of P' are embedded to the right of the directed cycle $\vec{C}'$ in skel(μ). In particular, the edge of skel(μ) containing x is to the right of $\vec{C}'$. Since the embedding of skel(μ) is assumed to be cycle-compatible with $\mathcal{H}$, x is to the right of $\vec{C}$ in $\mathcal{H}$.

This shows that $\mathcal{G}$ satisfies condition 2 of Lemma 2.1, as claimed. □

Definition 3.5. A planar embedding of the skeleton of a node μ of the SPQR-tree of G is *compatible with* $\mathcal{H}$ if it is both edge- and cycle-compatible with $\mathcal{H}$.

As a consequence of Lemmas 3.2 and 3.4, we obtain the following characterization of planar biconnected Pegs.

Theorem 3.6. *Let* $(G, H, \mathcal{H})$ *be a biconnected* Peg. *Then, G has an embedding that extends* $\mathcal{H}$ *if and only if the skeleton of each node of its SPQR-tree has an embedding compatible with* $\mathcal{H}$.

If G is biconnected, we can use Theorem 3.6 for devising a polynomial-time algorithm for Pep. Namely, we can test, for each node μ of the SPQR-tree $\mathcal{T}$ of G, whether skel(μ) has an embedding that is compatible with $\mathcal{H}$. For Q-, S-, and R-nodes, this test is easily done in polynomial time.

If μ is a P-node, the test is more complex. Let x and y be the two poles of skel(μ). We say that a virtual edge e of skel(μ) is *constrained* if the expansion graph of e contains at least one edge of H incident to x and at least one edge of H incident to y. To obtain an embedding of μ edge-compatible with $\mathcal{H}$, the constrained edges must be embedded in a cyclic order that is consistent with $\sigma_{\mathcal{H}}(x)$ and $\sigma_{\mathcal{H}}(y)$. Such a cyclic order, if it exists, is unique and can be determined in polynomial time. Note that if $\mathcal{H}$ has a facial cycle $\vec{C}$ that projects to a proper cycle $\vec{C}'$ in μ, then $\vec{C}'$ has exactly two edges and these two edges are both constrained. Thus, the embedding of any such cycle $\vec{C}'$ in μ is fixed as soon as we fix the cyclic order of the constrained edges. Once the cyclic order of the constrained edges of μ is determined, we process the remaining edges one by one and insert them among the edges that are already embedded in such a way that no edge- or cycle compatibility constraints are violated. It is not difficult to verify that this procedure constructs an embedding of μ compatible with $\mathcal{H}$, if such an embedding exists.

Thus, Pep can be solved in polynomial time for biconnected Pegs.

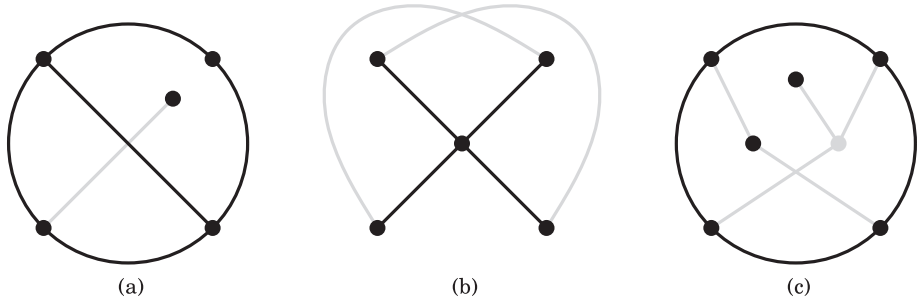

Fig. 7. Three examples of PEP instances $(G, H, \mathcal{H})$ that have no embedding extension, even though each block of G admits an embedding extending the corresponding sub-embedding of $\mathcal{H}$. The black edges and vertices represent $\mathcal{H}$, and the grey edges and vertices belong to G but not to H. Note that instance (a) fails to satisfy condition 3 of Lemma 3.8 (shown later), instance (b) fails to satisfy condition 2 of Lemma 3.8, and instance (c) has a nontrivial nonlocal H-bridge. The modification of instance (c) into an equivalent instance without nontrivial nonlocal H-bridges creates a block of G that does not have an embedding extension.

3.2. Planarity of Connected and Disconnected PEGs

A graph is planar if and only if each of its blocks is planar. Thus, planarity testing of general graphs can be reduced to planarity testing of biconnected graphs. For planarity testing of partially embedded graphs, the same simple reduction does not work (Figure 7). However, we will show that solving partially embedded planarity for a general instance $(G, H, \mathcal{H})$ can be reduced to solving the subinstances induced by the blocks of G and to checking additional conditions guaranteeing that the partial solutions can be combined into a full solution for $(G, H, \mathcal{H})$.

Let us consider a connected PEG $(G, H, \mathcal{H})$—that is, an instance of PEP in which G is connected. When dealing with such an instance, it is useful to assume that G has no nontrivial nonlocal H-bridge. We will now show that any instance of PEP can be transformed into an equivalent instance that satisfies this additional assumption.

Let K be a nontrivial nonlocal H-bridge of G. Since K is nonlocal, it must have at least two attachments that do not belong to any single block of H. Let f_K be the face of $\mathcal{H}$ whose boundary contains all attachments of the H-bridge K, if any such a face exists. Otherwise, let f_K be an arbitrary face of $\mathcal{H}$.

Let $\mathcal{K}$ be the set of nontrivial nonlocal H-bridges of G. It is clear that in any embedding of G extending $\mathcal{H}$, all vertices of $K - V(H)$ are embedded inside f_K for every $K \in \mathcal{K}$. This motivates the following definition.

Definition 3.7. Let H' be the graph whose edge set is equal to the edge set of H and whose vertex set is defined by $V(H') = V(H) \cup \bigcup_{K \in \mathcal{K}} V(K)$. Let $\mathcal{H}'$ be the embedding of H' that is obtained from $\mathcal{H}$ by inserting, for every H-bridge $K \in \mathcal{K}$, all vertices of $K - V(H)$ into the interior of face f_K.

Observe that the graph G has no nontrivial nonlocal H'-bridges. In addition, observe that any embedding of G that extends $\mathcal{H}$ also extends $\mathcal{H}'$, and vice versa. Thus, the instance $(G, H, \mathcal{H})$ of PEP is equivalent to the instance $(G, H', \mathcal{H}')$, which contains no nontrivial nonlocal bridges.

Before we state the next lemma, we need more terminology. Let $\mathcal{H}$ be an embedding of a graph H, and let H_1 and H_2 be edge-disjoint subgraphs of H. We say that H_1 and H_2 *alternate* around a vertex x of $\mathcal{H}$ if there are two pairs of edges $e, e' \in E(H_1)$ and $f, f' \in E(H_2)$ that are incident to x and that appear in the cyclic order (e, f, e', f') in the rotation of x restricted to these four edges. Let x and y be two vertices of H and let $\vec{C}$ be

a directed cycle in $\mathcal{H}$. We say that $\vec{C}$ *separates* x and y if $x \in V_{\mathcal{H}}^{\text{left}}(\vec{C})$ and $y \in V_{\mathcal{H}}^{\text{right}}(\vec{C})$, or vice versa.

LEMMA 3.8. *Let $(G, H, \mathcal{H})$ be an instance of* PEP *where G is connected and every nontrivial H-bridge of G is local. Let $G_1, \ldots, G_t$ be the blocks of G, let H_i be the subgraph of H induced by the vertices of G_i, and let $\mathcal{H}_i$ be $\mathcal{H}$ restricted to H_i. Then, G has an embedding extending $\mathcal{H}$ if and only if*

(1) *G_i has an embedding extending $\mathcal{H}_i$, for every $1 \leq i \leq t$,*
(2) *no two distinct graphs H_i and H_j alternate around any vertex of $\mathcal{H}$, and*
(3) *for every facial cycle $\vec{C}$ of $\mathcal{H}$ and for any two vertices x and y of $\mathcal{H}$ separated by $\vec{C}$, any path in G connecting x and y contains a vertex of $\vec{C}$.*

PROOF. Clearly, the three conditions of the lemma are necessary. To show that they are also sufficient, assume that the three conditions are satisfied and proceed by induction on the number t of blocks of G.

If $t = 1$, then G is biconnected and there is nothing to prove. Assume that $t \geq 2$. If there is at least one block G_i that does not contain any vertex of H, we consider the subgraph G' of G consisting of those blocks that contain at least one vertex of H. Since every nontrivial H-bridge of G is local, the graph G' is connected, and hence it satisfies the three conditions of the lemma. By induction, the embedding $\mathcal{H}$ can be extended into an embedding $\mathcal{G}'$ of G'. Since every block G_i of G is planar (by condition 1 of the lemma), it is easy to extend the embedding $\mathcal{G}'$ into an embedding $\mathcal{G}$ of G.

Assume now that every block of G contains at least one vertex of H. This implies that every cutvertex of G belongs to H, because otherwise the cutvertex would belong to a nontrivial nonlocal H-bridge, which is impossible by assumption. Let x be any cutvertex of G. Let $G'_1, G'_2, \ldots, G'_k$ be the connected components of $G - x$, where we select G'_1 by the following rules: if there is a component of $G - x$ that has no vertex connected to x by an edge of H, then let G'_1 be such a component; if each component of $G - x$ is connected to x by an edge of H, then choose G'_1 in such a way that the edges of H incident to x and belonging to G'_1 form an interval in $\sigma_{\mathcal{H}}(x)$. Such a choice of G'_1 is always possible, due to condition 2 of the lemma.

Let G' be the subgraph of G induced by $V(G'_1) \cup \{x\}$ and let G'' be the subgraph of G induced by $V(G'_2) \cup \cdots \cup V(G'_k) \cup \{x\}$. Let H' and H'' be the subgraphs of H induced by the vertices of G' and G'', respectively, and let $\mathcal{H}'$ and $\mathcal{H}''$ be $\mathcal{H}$ restricted to H' and H'', respectively. Both G' and G'' have fewer blocks than G. In addition, both instances, $(G', H', \mathcal{H}')$ and $(G'', H'', \mathcal{H}'')$, satisfy the conditions of the lemma. Thus, by induction, there is an embedding $\mathcal{G}'$ of G' that extends $\mathcal{H}'$ and an embedding $\mathcal{G}''$ of G'' that extends $\mathcal{H}''$. Our goal is to combine $\mathcal{G}'$ and $\mathcal{G}''$ into a single embedding of G that extends $\mathcal{H}$. To see that this is possible, we prove two auxiliary claims.

Claim 1. $\mathcal{H}'$ has a face f' whose boundary contains x and, for any facial cycle $\vec{C}$ of f', all vertices of H'' except for x are in $V_{\mathcal{H}}^{\text{left}}(\vec{C})$—that is, they are "inside" f'.

To see that the claim holds, assume first that H' has no edge incident to x (Figure 8(a)). Let f' be the unique face of $\mathcal{H}'$ incident to x. We show that all vertices of H'' are inside f' in $\mathcal{H}$. Let y be any vertex of H''. Since G'' is connected, there is a path P in G'' from y to x. Assume for contradiction that $\mathcal{H}'$ has a facial cycle $\vec{C}$ such that $\vec{C}$ separates y from x in $\mathcal{H}$. This cycle belongs to $H' - x$, and hence $\vec{C}$ and P are disjoint, contradicting condition 3 of the lemma.

Next, assume that H' has an edge incident to x (see Figure 8(b)). By the construction of G_1, each connected component of $G - x$ has at least one vertex connected to x by an edge of H. Moreover, the edges of $\mathcal{H}'$ incident to x form an interval in $\sigma_{\mathcal{H}}(x)$. This shows that $\mathcal{H}'$ has a face f' containing x on its boundary, such that every vertex of H''

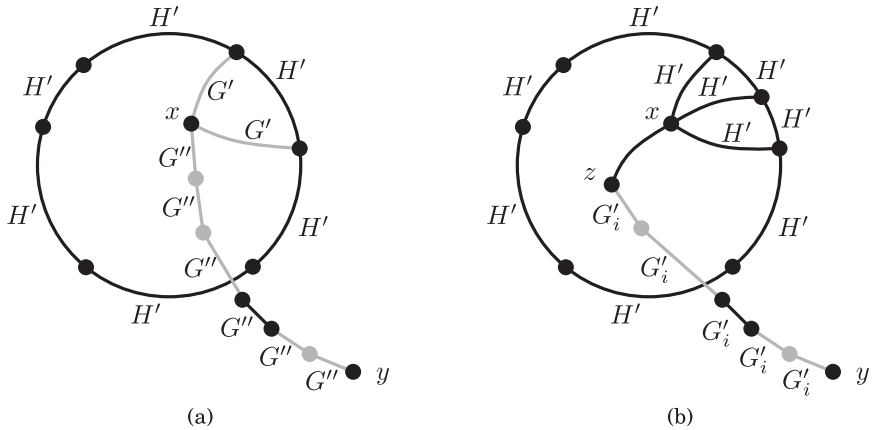

Fig. 8. Illustration for the proof of Lemma 3.8. (a) H' has no edge incident to x. (b) H' has an edge incident to x.

adjacent to x is inside f' in $\mathcal{H}$. We now show that all vertices of H'' except for x are inside f'. Let y be a vertex of H'' different from x. Let G'_i be the component of $G - x$ containing y. We know that G'_i has a vertex z adjacent to x by an edge of H and that z is inside f' in $\mathcal{H}$. Let P be a path in G'_i connecting y and z. If y is not inside f', then y is separated from z in $\mathcal{H}$ by a facial cycle of $\mathcal{H}'$, contradicting condition 3 of the lemma.

Claim 2. All vertices of H', except for x, appear in $\mathcal{H}$ inside the same face f'' of $\mathcal{H}''$; furthermore, x is on the boundary of f''.

To prove the claim, note that any two vertices from $H' - x$ are inside the same face f'' of $\mathcal{H}''$ in $\mathcal{H}$ by condition 3 of the lemma because they are connected by a path in G'_1. Vertex x is on the boundary of f'', as otherwise it would be separated in $\mathcal{H}$ from the remaining vertices of H' by a facial cycle of f'', again contradicting condition 3 of the lemma.

In view of the previous two claims, the embedding $\mathcal{G}'$ of G' and the embedding $\mathcal{G}''$ of G'' can be combined into a single embedding $\mathcal{G}$ of G that extends $\mathcal{H}$. To see this, note that when $\mathcal{H}'$ is extended into $\mathcal{G}'$, the face f' from Claim 1 can be subdivided into several faces of $\mathcal{G}'$, at least one of which, say g', contains x on its boundary. Analogously, the face f'' from Claim 2 can be subdivided into several faces of $\mathcal{G}''$, at least one of which, say g'', contains x on its boundary. We then obtain the embedding $\mathcal{G}$ by merging the faces g' and g'' into a single face. □

Observe that computing the nonlocal H-bridges $\mathcal{K}$ and, for each nonlocal H-bridge $K \in \mathcal{K}$, the face f_K can be done in polynomial time. Afterward, the second and third conditions of Lemma 3.8 can easily be checked in polynomial time.

Next, we focus on disconnected PEGs—that is, the instances $(G, H, \mathcal{H})$ of PEP in which G is not connected. The possibility of solving the subinstances of $(G, H, \mathcal{H})$ induced by the connected components of G does not guarantee that the instance $(G, H, \mathcal{H})$ of PEP has a solution. However, we show that solving PEP for an instance $(G, H, \mathcal{H})$ can be reduced to solving the subinstances induced by the connected components of G and to checking additional conditions that guarantee that the partial solutions can be combined into a full solution for $(G, H, \mathcal{H})$.

LEMMA 3.9. *Let $(G, H, \mathcal{H})$ be an instance of* PEP. *Let $G_1, \ldots, G_t$ be the connected components of G. Let H_i be the subgraph of H induced by the vertices of G_i, and let $\mathcal{H}_i$ be $\mathcal{H}$ restricted to H_i. Then G has an embedding extending $\mathcal{H}$ if and only if*

(1) *G_i has an embedding extending $\mathcal{H}_i$, for every $1 \leq i \leq t$, and*
(2) *for every $1 \leq i, j \leq t$ with $j \neq i$, and for each facial cycle $\vec{C}$ of H_i, no two vertices of H_j are separated by $\vec{C}$.*

PROOF. Clearly, the two conditions of the lemma are necessary. To show that they are also sufficient, assume that the two conditions are satisfied and proceed by induction on the number t of connected components of G.

If $t = 1$, then G is connected and there is nothing to prove. Assume now that G has $t \geq 2$ connected components $G_1, \ldots, G_t$. Let H_i and $\mathcal{H}_i$ be defined as in the statement of the lemma. Note that H_i may consist of several connected components. Let $\mathcal{CF}$ be the component-face tree of $\mathcal{H}$, rooted at a node that represents an arbitrary face of $\mathcal{H}$. We say that a face f_i of $\mathcal{H}$ is the *outer face* of $\mathcal{H}_i$ if at least one child of f_i in $\mathcal{CF}$ is a component of H_i but the parent of f_i is not a component of H_i. Observe that due to the second condition of the lemma, each $\mathcal{H}_i$ has exactly one outer face f_i. We thus have a sequence of (not necessarily distinct) outer faces $f_1, \ldots, f_t$ of $\mathcal{H}_1, \ldots, \mathcal{H}_t$.

Let us now assume, without loss of generality, that in the subtree of $\mathcal{CF}$ rooted at f_1, there is no outer face $f_i \neq f_1$. This implies that f_1 is the only face of $\mathcal{H}$ that is incident both to H_1 and to $H - H_1$. By induction, the embedding $\mathcal{H} - \mathcal{H}_1$ can be extended to an embedding $\mathcal{G}_{\geq 2}$ of the graph $G - G_1$. By the first condition of the lemma, $\mathcal{H}_1$ can be extended into an embedding $\mathcal{G}_1$ of G_1. The two embeddings $\mathcal{H} - \mathcal{H}_1$ and $\mathcal{H}_1$ share a single face f_1.

When extending the embedding $\mathcal{H}_1$ into $\mathcal{G}_1$, the face f_1 of $\mathcal{H}_1$ can be subdivided into several faces of $\mathcal{G}_1$. Let f' be any face of $\mathcal{G}_1$ obtained by subdividing f_1. Analogously, in the embedding $\mathcal{G}_{\geq 2}$ the face f_1 can be subdivided into several faces, among which we choose an arbitrary face f''.

We then glue the two embeddings $\mathcal{G}_1$ and $\mathcal{G}_{\geq 2}$ by identifying the face f' of $\mathcal{G}_1$ and the face f'' of $\mathcal{G}_{\geq 2}$ into a single face whose boundary is the union of the boundaries of f' and f''. This yields an embedding of G that extends $\mathcal{H}$. □

Note that the second condition of Lemma 3.9 can easily be tested in polynomial time. Thus, we can use the characterization to directly prove that PARTIALLY EMBEDDED PLANARITY is solvable in polynomial time. In the rest of the article, we describe a more sophisticated algorithm that solves PEP in linear time.

4. LINEAR-TIME ALGORITHM

In this section, we devise a linear-time algorithm for solving PEP. The algorithm basically follows the outline of the characterization. The first milestone is a linear-time algorithm for testing the planarity of biconnected PEGs. Afterward, we show that the additional conditions for the planarity of connected and disconnected PEGs can be checked in linear time.

Essentially, to solve the biconnected case, it is sufficient to give a linear-time implementation of the algorithm sketched at the end of Section 3.1. In fact, most of the steps sketched there are fairly easy to implement in linear time. The problem of finding a compatible embedding of a P-node, however, is tricky. Indeed, a P-node μ may contain a linear number of facial cycles of $\mathcal{H}$ that project to cycles in skel(μ). Further, a linear number of virtual edges of skel(μ) may have no H-edge adjacent to the poles of μ; hence, the positions at which they have to be inserted in the cyclic orderings around the poles of μ depend only on the cycle containment constraints. To process the skeleton of a P-node μ in time proportional to its size, we would need to find, for each virtual edge e of skel(μ), a position such that e is contained in all and only the cycles in which it needs to be contained.

Therefore, the main problem for reaching linear running time stems from the cycle compatibility constraints (condition 2 of Lemma 2.1). The constraints stemming from rotation system (condition 1 of Lemma 2.1) consist of orderings of subsets of the edges incident to each vertex. Thus, the total size of the latter constraints is linear, and additionally, the constraints are very "local." In light of these two properties, it is not surprising that the rotation system constraints are relatively simple to handle in total linear time. The same does not hold, however, for the cycle containment constraints. As specified in condition 2 of Lemma 2.1, these constraints determine, for each directed facial cycle $\vec{C}$ of H and each vertex v of $H - C$, whether v is to the left or to the right of $\vec{C}$. Note that the graph H may contain a linear number of facial cycles, and thus this amounts to quadratically many cycle–vertex constraints. Further, these constraints do not exhibit any locality on G. Evidently, a lot of the information encoded in the cycle containment constraints is redundant, as the set of cycles involved in these constraints is the set of facial cycles of a planar graph.

We use two different approaches to handle the cycle containment constraints in linear time. One is to ignore them; we prove that this yields a correct solution if H is connected, as in this case the cycle containment constraints are implied by the rotation system constraints. The second consists of considering restricted instances, where the constraints can easily be expressed in linear time and space. Suppose that we have a PEG $(G, H, \mathcal{H})$ with a face f of $\mathcal{H}$ such that all vertices of H are part of at least one facial cycle of f. This implies that each facial cycle of H has f on its left side and the right side does not contain any vertex of H. In this case, the cycle containment constraints of each facial cycle $\vec{C}$ of f can be expressed as $V_{\mathcal{H}}^{\text{right}}(\vec{C}) = \emptyset$, thus yielding a set of constraints whose size is linear. Moreover, in the SPQR-tree, it is sufficient to keep track of which virtual edges contain vertices of H and which do not. This information is much easier to aggregate than information about individual vertices and all of their cycle containment constraints.

First, we tackle the case in which G is biconnected. The algorithm solving this case, presented in Section 4.3, uses the algorithms presented in Sections 4.1 and 4.2 as subroutines to solve more restricted subcases. Then, we deal with the case in which G is simply connected and with the general case, where G may be disconnected, in Section 4.4. The algorithm we present exploits several auxiliary data structures, namely block-cutvertex trees, SPQR-trees, enriched block-cutvertex trees, block-face trees, component-face trees, and vertex-face incidence graphs. Note that all of these data structures can easily be computed in linear time (see Section 2.3).

4.1. *G* Biconnected, *H* Connected

In this section, we show how to solve PEP in linear time for biconnected PEGS $(G, H, \mathcal{H})$ with H connected. We first show that in this case the rotation system alone is sufficient for finding an embedding extension.

LEMMA 4.1. *Let $(G, H, \mathcal{H})$ be a* PEG *such that H is connected. Let $\mathcal{G}$ be any planar embedding of G satisfying condition 1 of Lemma 2.1. Then, $\mathcal{G}$ satisfies condition 2 of Lemma 2.1.*

PROOF. Suppose, for a contradiction, that a planar embedding $\mathcal{G}$ of G exists such that $\mathcal{G}$ satisfies condition 1 and does not satisfy condition 2 of Lemma 2.1. Then, there exists a facial cycle $\vec{C}$ of $\mathcal{H}$ such that either there exists a vertex $x \in V_{\mathcal{H}}^{\text{left}}(\vec{C})$ with $x \in V_{\mathcal{G}}^{\text{right}}(\vec{C})$ or there exists a vertex $x \in V_{\mathcal{H}}^{\text{right}}(\vec{C})$ with $x \in V_{\mathcal{G}}^{\text{left}}(\vec{C})$. Suppose that we are in the former case, as the latter case can be treated analogously. Since $\mathcal{H}$ is a planar embedding and H is connected, there exists a path $P = (x_1, x_2, \ldots, x_k) \in H$ such that x_1

is a vertex of $\vec{C}$, $x_i \in V_{\mathcal{H}}^{\text{left}}(\vec{C})$, for each $i = 2, \ldots, k$, and $x_k = x$. Denote by x_1^- and by x_1^+ the vertex preceding and following x_1 in the oriented cycle $\vec{C}$, respectively. Consider the placement of x_2 with respect to $\vec{C}$ in $\mathcal{G}$. As $x_2 \notin \vec{C}$, either $x_2 \in V_{\mathcal{G}}^{\text{left}}(\vec{C})$ or $x_2 \in V_{\mathcal{G}}^{\text{right}}(\vec{C})$. In the first case, the path $(x_2, \ldots, x_k)$ crosses $\vec{C}$, since $x_2 \in V_{\mathcal{G}}^{\text{left}}(\vec{C})$, $x_k \in V_{\mathcal{G}}^{\text{right}}(\vec{C})$, and no vertex v_i belongs to $\vec{C}$, for $i = 2, \ldots, k$, thus contradicting the planarity of the embedding $\mathcal{G}$. In the second case, the clockwise order of the edges incident to x_1 in $\mathcal{H}$ is (x_1, x_1^-), (x_1, x_2), and (x_1, x_1^+), whereas the clockwise order of the edges incident to x_1 in $\mathcal{G}$ is (x_1, x_1^-), (x_1, x_1^+), and (x_1, x_2), thus contradicting the assumption that $\mathcal{G}$ satisfies condition 1 of Lemma 2.1. □

By Lemma 4.1, testing whether a planar embedding $\mathcal{G}$ exists satisfying conditions 1 and 2 of Lemma 2.1 is equivalent to testing whether a planar embedding $\mathcal{G}$ exists satisfying condition 1 of Lemma 2.1. Due to Lemma 3.2, testing whether a planar embedding $\mathcal{G}$ exists satisfying condition 1 is equivalent to testing whether the skeleton of each node of the SPQR-tree of G has a planar embedding that is edge-compatible with $\mathcal{H}$. We now describe an algorithm—Algorithm BC (for G Biconnected and H Connected)—that achieves this in linear time.

Algorithm BC. Construct the SPQR-tree $\mathcal{T}$ of G and root it at an arbitrary Q-node. This choice determines the pertinent graph of each node μ of $\mathcal{T}$ and also determines the parent virtual edge in the skeleton of each nonroot node μ of $\mathcal{T}$. A bottom-up visit of $\mathcal{T}$ is performed, such that after a node μ of $\mathcal{T}$ has been visited, an embedding of skel(μ) that is edge-compatible with $\mathcal{H}$ is selected, if it exists. To find an edge-compatible embedding of the skeleton skel(μ) of a node μ of $\mathcal{T}$, we need to know whether the expansion graph of each virtual edge uv of skel(μ) contains H-edges incident to u and v. Due to the bottom-up traversal, the pertinent graphs of all children have already been processed by the algorithm, and we can thus aggregate this information for all virtual edges, except for the parent edge.

To keep track of the edges of H that belong to pert(μ) and that are incident to the pole $u(\mu)$, define the *first edge* $f_{u(\mu)}$ and the *last edge* $l_{u(\mu)}$ as the edges of H incident to $u(\mu)$ in pert(μ) such that all other edges of H incident to $u(\mu)$ in pert(μ) appear between $f_{u(\mu)}$ and $l_{u(\mu)}$ in the counterclockwise order of the edges incident to $u(\mu)$ in $\mathcal{H}$. The first and last edge of $v(\mu)$ are defined analogously. Note that if the edges of H that belong to pert(μ) and are incident to $u(\mu)$ do not form an interval in the rotation of u in $\mathcal{H}$, then the skeleton of μ has no edge-compatible embedding and the PEG (G, H, $\mathcal{H}$) is nonplanar.

After a node μ of $\mathcal{T}$ has been visited by the algorithm, edges $f_{u(\mu)}$, $l_{u(\mu)}$, $f_{v(\mu)}$, and $l_{v(\mu)}$ are associated with μ. We can then also refer to them as $f_{u(e)}$, $l_{u(e)}$, $f_{v(e)}$, and $l_{v(e)}$, where e is the virtual edge corresponding to μ in the skeleton of the parent of μ.

If μ is a Q- or an S-node, no check is needed. As skel(μ) is a cycle (possibly of length 2 in case of a Q-node), the only planar embedding of skel(μ) is edge-compatible with $\mathcal{H}$. The edges $f_{u(\mu)}$, $l_{u(\mu)}$, $f_{v(\mu)}$, and $l_{v(\mu)}$ are easily computed.

If μ is an R-node, then skel(μ) has only two planar embeddings. For each of them, verify if it is edge-compatible with $\mathcal{H}$ by performing the following check. For each vertex x of skel(μ), restrict the circular list of its incident virtual edges to the virtual edges $e_1, \ldots, e_h$ that contain an edge of H incident to x. Check if $l_{x(e_i)}$ precedes $f_{x(e_{i+1})}$ (for $i = 1, \ldots, h$, where $e_{h+1} = e_1$) in the list $E_H(x)$ of edges incident to x in $\mathcal{H}$. If x is a pole, do an analogous check on the linear list of its incident virtual edges obtained by removing the parent edge from the circular list. If one of the tests succeeds, then select the corresponding embedding for skel(μ). Set $f_{u(\mu)} = f_{u(f_1)}$, $l_{u(\mu)} = l_{u(f_p)}$, $f_{v(\mu)} = f_{v(g_1)}$, and $l_{v(\mu)} = l_{v(g_q)}$, where f_1 and f_p (g_1 and g_q) are the first and the last virtual edge in the

linear list of the virtual edges containing an edge of H and incident to $u(\mu)$ (respectively to $v(\mu)$).

If μ is a P-node, an embedding of skel(μ) is a counterclockwise order of its virtual edges around $u(\mu)$. We describe how to verify whether an embedding of skel(μ) exists that is edge-compatible with $\mathcal{H}$.

Consider the virtual edges containing edges of H incident to $u(\mu)$. We show how to construct a list L_u of such edges corresponding to the ordering they have in any embedding of skel(μ) that is edge-compatible with $\mathcal{H}$. Insert one such edge, say e_i, into L_u. Repeatedly consider the last element e_j of L_u, and insert as the new last element of L_u the edge e_{j+1} such that $l_{u(e_j)}$ immediately precedes $f_{u(e_{j+1})}$ in the counterclockwise order of the edges incident to $u(\mu)$ in $\mathcal{H}$. If $e_{j+1} = e_i$, then L_u is the desired circular list. If e_{j+1} does not exist, then the edge following $l_{u(e_j)}$ belongs to the parent edge of μ. Then, consider the first edge e_i. Repeatedly consider the first element e_j of L_u, and insert as the new first element of L_u the edge e_{j-1} such that $f_{u(e_j)}$ immediately follows $l_{u(e_{j-1})}$ in the counterclockwise order of the edges incident to $u(\mu)$ in $\mathcal{H}$. If e_{j-1} does not exist, then check whether all virtual edges containing edges of H incident to $u(\mu)$ have been processed, and in this case insert the the parent edge of μ as the first element of L_u. Analogously, construct a list L_v.

Let L_{uv} be the sublist obtained by restricting L_u to those edges that appear in L_v. Let L_{vu} be the corresponding sublist of L_v. Check whether L_{uv} and L_{vu} are the reverse of each other. If this is the case, a list L of the virtual edges of skel(μ) containing edges of H incident to $u(\mu)$ or to $v(\mu)$ can easily be constructed compatible with both L_u and L_v.

Finally, arbitrarily insert into L the virtual edges of skel(μ) not in L_u and not in L_v, thus obtaining an embedding of skel(μ) edge-compatible with $\mathcal{H}$.

Denote by f_1 and f_p (by g_1 and g_q) the virtual edges containing edges of H incident to $u(\mu)$ (respectively to $v(\mu)$) following and preceding the parent edge of μ in L. Set $f_{u(\mu)} = f_{u(f_1)}$, $l_{u(\mu)} = l_{u(f_p)}$, $f_{v(\mu)} = f_{v(g_1)}$, and $l_{v(\mu)} = l_{v(g_q)}$.

THEOREM 4.2. *Let $(G, H, \mathcal{H})$ be an n-vertex instance of* PEP *such that G is biconnected and H is connected. Algorithm BC solves* PEP *for $(G, H, \mathcal{H})$ in $O(n)$ time.*

PROOF. We show that Algorithm BC processes each node μ of $\mathcal{T}$ in $O(k_\mu)$ time, where k_μ is the number of children of μ in $\mathcal{T}$.

First, observe that the computation of $f_{u(\mu)}$, $l_{u(\mu)}$, $f_{v(\mu)}$, and $l_{v(\mu)}$ is trivially done in $O(1)$ time once the embedding of skel(μ) has been decided.

If μ is a Q-node or an S-node, Algorithm BC does not perform any check or embedding choice.

If μ is an R-node, Algorithm BC computes the two planar embeddings of skel(μ) in $O(k_\mu)$ time. For each of these embeddings, Algorithm BC processes each vertex x of skel(μ) separately, considering the list of the virtual edges incident to x (which is trivially constructed in $O(t)$ time, where t is the number of such edges), and restricting the list to those virtual edges containing an edge of H incident to x (for each virtual edge, it suffices to check whether the first edge incident to x is associated with an edge of H, which is done in $O(1)$ time). Checking whether $l_{x(e_i)}$ precedes $f_{x(e_{i+1})}$ in the list of the edges incident to x in $\mathcal{H}$ is done in $O(1)$ time. Hence, the total time spent for each node x is $O(t)$. Summing up over all nodes of skel(μ) results in a total $O(k_\mu)$ time, as every edge is incident to two nodes and the total number of edges in skel(μ) is $O(k_\mu)$.

If μ is a P-node, extracting the virtual edges of skel(μ) containing edges of H incident to $u(\mu)$ or to $v(\mu)$ can be done in $O(k_\mu)$ time, as in the R-node case. For each of such edges, equipping $f_{u(e)}$, $l_{u(e)}$, $f_{v(e)}$, and $l_{v(e)}$ with a link to e is done in constant time. Determining an ordering of the virtual edges containing edges of H incident to $u(\mu)$ can be done in $O(k_\mu)$ time, as the operations performed for each virtual edge e_i are accessing the first

and the last edge of e_i, accessing the edge following the last edge of e_i (preceding the first edge of e_i) in the counterclockwise order of the edges incident to $u(\mu)$ in $\mathcal{H}$, and accessing a virtual edge linked from the first or last edge; each of these operations is trivially done in $O(1)$ time. Marking the virtual edges in L_u and in L_v is done in $O(k_\mu)$ time, as L_u and L_v have $O(k_\mu)$ elements. Then, obtaining L_{uv} and L_{vu}, and checking whether they are the reverse of each other, is done in $O(k_\mu)$ time. Finally, extending L_{uv} to L is also easily done in $O(k_\mu)$ time; namely, if L_{uv} is empty, then let L be the concatenation of L_u and L_v (where such lists are made linear by cutting them at any point). Otherwise, start from an edge e_i of L_{uv}; e_i is also in L_u and in L_v; insert e_i into L; insert into L all the edges of L_u following e_i until the next edge e_{i+1} of L_{uv} has been found; insert into L all edges of L_v preceding e_i until the next edge e_{i+1} of L_{uv} has been found; insert e_{i+1} into L, and repeat the procedure. Each element of L_{uv}, L_u, and L_v is visited once, and hence such a step is performed in $O(k_\mu)$ time.

As $\sum_{\mu \in \mathcal{T}} k_\mu = O(n)$, the total running time of the algorithm is $O(n)$. □

Note that although Algorithm BC relies only on the assumptions that G is biconnected and H is connected, we will only use it in the more special case where H is also biconnected.

4.2. G Biconnected, All Vertices and Edges of G Lie in the Same Face of $\mathcal{H}$

The PEGs considered in this section are denoted by $(G(f), H(f), \mathcal{H}(f))$. Such instances are assumed to satisfy the following properties: (i) $G(f)$ is biconnected, (ii) $G(f)$ and $H(f)$ have the same vertex set, (iii) all vertices and edges of $H(f)$ are incident to the same face f of $\mathcal{H}(f)$, and (iv) no edge of $G(f) \backslash H(f)$ connects two vertices of the same block of $H(f)$. Algorithm BF, which deals with such a setting, is used as a subroutine by Algorithm BA, to be shown later, dealing with the instances of PEP in which G is biconnected and H is arbitrary.

First, we show that the structure of the cycles in $H(f)$ is very special.

PROPERTY 3. *Every simple path with at least two vertices of $H(f)$ is contained in at most one simple cycle of $H(f)$.*

PROOF. Suppose that there exists a path (that can possibly be a single edge) of $H(f)$ belonging to at least two simple cycles of $H(f)$. Then, such cycles share edges and define at least three regions of the plane. Not all edges of the two cycles can be incident to the same region, contradicting the fact that all edges of $H(f)$ are incident to the same region of the plane in $\mathcal{H}(f)$. □

Since all vertices and edges are incident to f, the only relevant cycles for which cycle–vertex constraints have to be checked are the facial cycles of f. We exploit this particular structure of the input to simplify the test of cycle compatibility with $\mathcal{H}(f)$ for the skeleton of a node μ of $\mathcal{T}(f)$, where $\mathcal{T}(f)$ is the SPQR-tree of $G(f)$.

LEMMA 4.3. *Consider any node μ of $\mathcal{T}(f)$. Then, an embedding of* skel(μ) *is cycle-compatible with $\mathcal{H}(f)$ if and only if for every facial cycle $\vec{C}$ of $\mathcal{H}(f)$ whose edges project to a cycle $\vec{C}'$ of* skel(μ)*, no vertex and no edge of* skel(μ) *is to the right of $\vec{C}'$, where $\vec{C}'$ is oriented according to the orientation of $\vec{C}$.*

PROOF. By assumption (iii) on the input, all vertices and edges of $H(f)$ are incident to the same face f of $\mathcal{H}(f)$. By construction, every facial cycle $\vec{C}$ of $H(f)$ is oriented in such a way that f and hence all vertices of $H(f)$ are to the left of $\vec{C}$. Then, by Lemma 3.4, if the edges of $\vec{C}$ determine a cycle $\vec{C}'$ of virtual edges of skel(μ), all vertices of skel(μ) that are not in $\vec{C}$ and all virtual edges of skel(μ) that are not in $\vec{C}'$ and that

contain vertices of $G(f)$ have to be to the left of $\vec{C}'$. Finally, all virtual edges that are not in $\vec{C}'$ and that do not contain any vertex of $G(f)$ (i.e., virtual edges corresponding to Q-nodes) have one end-vertex that is not in $\vec{C}$, by assumption (iv) on the input. Such an end-vertex forces the edge to be to the left of $\vec{C}'$. □

To find compatible embeddings for the skeletons of the nodes of $\mathcal{T}(f)$, we again need to find edge-compatible embeddings, which can be done as in Algorithm BC. However, unlike Algorithm BC, Algorithm BF has also to make embedding choices to satisfy cycle compatibility constraints. Such constraints are the ones expressed by Lemma 4.3: for any node μ of $\mathcal{T}(f)$, the sought-after embedding of skel(μ) is such that for every facial cycle $\vec{C}$ of $\mathcal{H}(f)$ whose edges project to a cycle $\vec{C}'$ of skel(μ), no vertex and no edge of skel(μ) is to the right of $\vec{C}'$, where $\vec{C}'$ is oriented according to the orientation of $\vec{C}$.

The choice of an embedding for skel(μ) does not affect whether the cycle compatibility constraints are satisfied for the facial cycles $\vec{C}$ of $\mathcal{H}(f)$ whose edges project to a single virtual edge of skel(μ). On the other hand, the choice of an embedding for skel(μ) does affect whether the cycle compatibility constraints are satisfied for the facial cycles $\vec{C}$ of $\mathcal{H}(f)$ whose edges project to a cycle $\vec{C}'$ of skel(μ). Hence, to ensure cycle compatibility, we need to (quickly) find the projections of the facial cycles of $\mathcal{H}(f)$ in the skeletons of the nodes of $\mathcal{T}(f)$ and need to perform embedding choices for such skeletons that satisfy the constraints of Lemma 4.3. These tasks are simplified by the following two observations.

First, the facial cycles $\vec{C}$ of $\mathcal{H}(f)$ whose edges project to a cycle $\vec{C}'$ of skel(μ) are composed of a sequence of *traversing paths* for the neighbors of μ in $\mathcal{T}(f)$: for any neighbor ν of μ in $\mathcal{T}(f)$, a traversing path is a path between $u(\nu)$ and $v(\nu)$ that is composed of edges of $H(f)$, that belongs to pert(ν), and that is part of a facial cycle $\vec{C}$ of $\mathcal{H}(f)$ not entirely contained in pert(ν).

Second, by Property 3, every edge of $H(f)$ (and hence every path of $H(f)$) can be contained in at most one facial cycle of $\mathcal{H}(f)$. Therefore, a single flag suffices to encode the existence of a traversing path for a node of $\mathcal{T}(f)$.

We now give a high-level description of Algorithm BF.

Like Algorithm BC, Algorithm BF starts with the construction of the SPQR-tree $\mathcal{T}(f)$ of $G(f)$, roots it at an arbitrary Q-node, and visits $\mathcal{T}(f)$ in bottom-up order in such a way that after a node μ of $\mathcal{T}(f)$ has been visited, an embedding of skel(μ) that is compatible with $\mathcal{H}(f)$ is selected, if it exists.

To deal with edge compatibility constraints, Algorithm BF maintains edges $f_{u(\mu)}$, $l_{u(\mu)}$, $f_{v(\mu)}$, and $l_{v(\mu)}$ for each node μ of $\mathcal{T}(f)$ (and for each virtual edge in the skeleton of each node of $\mathcal{T}(f)$) as in Algorithm BC. Additionally, to deal with cycle compatibility constraints, the algorithm maintains a flag $p(\mu)$ for each node μ of $\mathcal{T}(f)$ such that $p(\mu)$ is set to TRUE if there exists a traversing path P for μ; flag $p(\mu)$ is set to FALSE otherwise. Furthermore, to encode the direction of P the algorithm maintains a flag $uv(\mu)$. If $p(\mu) =$ TRUE, the flag $uv(\mu)$ is set to TRUE if P is oriented from $u(\mu)$ to $v(\mu)$ according to the orientation of $\vec{C}$, and it is set equal to FALSE otherwise. We also refer to these flags as $p(e)$ and $uv(e)$, where e is the virtual edge corresponding to μ in the skeleton of the parent of μ.

Now, we state lemmas specifically dealing with S-, R-, and P-nodes of $\mathcal{T}(f)$. These lemmas will be used in the description of Algorithm BF.

LEMMA 4.4. *Let μ be an S-node of $\mathcal{T}(f)$ with children $\mu_1, \mu_2, \ldots, \mu_k$. Then, $p(\mu_i) =$ TRUE for some $1 \leq i \leq k$ if and only if $p(\mu_j) =$ TRUE for all $1 \leq j \leq k$.*

PROOF. If $p(\mu_j) = \text{TRUE}$ for all $1 \leq j \leq k$, then trivially $p(\mu_i) = \text{TRUE}$. If $p(\mu_i) = \text{TRUE}$ for some $1 \leq i \leq k$, there exists a traversing path of μ_i that is part of a simple cycle $\vec{C}$ of $H(f)$ not entirely contained in $\text{pert}(\mu_i)$; however, as μ is an S-node, $\vec{C}$ does not entirely lie inside $\text{pert}(\mu)$, as otherwise it would entirely lie inside $\text{pert}(\mu_i)$. Then, $\vec{C}$ consists of a traversing path of $\text{pert}(\mu_j)$, for all $1 \leq j \leq k$, and of a traversing path of the parent edge of $\text{skel}(\mu)$, thus proving the lemma. □

Next, we derive a simple criterion for an embedding of an R-node to be cycle-compatible. By Lemma 4.3, an embedding of a skeleton $\text{skel}(\mu)$ is cycle-compatible if for each facial cycle $\vec{C}$ of f that projects to a cycle $\vec{C}'$ in $\text{skel}(\mu)$, the right side of $\vec{C}'$ is empty. For an R-node μ this condition can be reformulated: each such cycle $\vec{C}'$ must have a face on its right side.

LEMMA 4.5. *Let μ be an R-node of $\mathcal{T}(f)$. If an edge e of* $\text{skel}(\mu)$ *has a traversing path belonging to a facial cycle $\vec{C}$, let us orient e in the direction determined by the projection of $\vec{C}$ in* $\text{skel}(\mu)$. *An embedding of* $\text{skel}(\mu)$ *is cycle-compatible with $\mathcal{H}(f)$ if and only if for each face g of the embedding of* $\text{skel}(\mu)$, *either (i) every virtual edge e on the boundary of g is oriented so that g is to the right of e or (ii) none of the virtual edges on the boundary of g is oriented in a way that g is to the right of it.*

PROOF. Suppose that an embedding of $\text{skel}(\mu)$ is cycle-compatible with $\mathcal{H}(f)$. Let g be a face of the embedding. If no edge e on the boundary of g contains a traversing path, then g satisfies condition (ii). Otherwise, assume that on the boundary of g there is an edge e containing a traversing path P such that g is to the right of e. Let $\vec{C}$ be the facial cycle of $\mathcal{H}(f)$ that contains P. By Lemma 4.3, $\vec{C}$ projects to a directed cycle $\vec{C}'$ in $\text{skel}(\mu)$, and no vertex or edge of $\text{skel}(\mu)$ is embedded to the right of $\vec{C}'$. Thus, $\vec{C}'$ corresponds to the boundary of the face g, and hence g satisfies condition (i).

Suppose now that in an embedding of $\text{skel}(\mu)$, every face satisfies condition (i) or condition (ii). We claim that the embedding of $\text{skel}(\mu)$ is cycle-compatible with $\mathcal{H}(f)$. To prove it, we use Lemma 4.3. Let $\vec{C}$ be a facial cycle of $\mathcal{H}(f)$ that projects to a simple cycle $\vec{C}'$ in $\text{skel}(\mu)$. Let e be any edge of $\vec{C}'$ and let g be the face to the right of e in the embedding of $\text{skel}(\mu)$. Necessarily, g satisfies condition (i). Hence, each edge on the boundary of g has a traversing path. The union of these paths forms a cycle in $\mathcal{H}(f)$, and by Property 3, this cycle is equal to $\vec{C}$. Thus, the boundary of g coincides with the cycle $\vec{C}'$. In particular, no vertex and no edge of $\text{skel}(\mu)$ is embedded to the right of $\vec{C}'$. By Lemma 4.3, this means that the embedding of $\text{skel}(\mu)$ is cycle-compatible with $\mathcal{H}(f)$. □

We next deal with P-nodes. The special structure of the PEGs $(G(f), H(f), \mathcal{H}(f))$ considered in this section implies that for each P-node μ, there exists at most one facial cycle $\vec{C}$ of f that projects to a cycle in $\text{skel}(\mu)$.

LEMMA 4.6. *Let μ be a P-node of $\mathcal{T}(f)$. There exist either zero or two virtual edges of* $\text{skel}(\mu)$ *containing a traversing path.*

PROOF. If there exists one virtual edge e_i of $\text{skel}(\mu)$ containing a traversing path that is part of a simple cycle $\vec{C}$ of $H(f)$ not entirely contained in $\text{pert}(e_i)$, another virtual edge of $\text{skel}(\mu)$ containing a traversing path that is part of $\vec{C}$ exists, as otherwise $\vec{C}$ would not be a cycle. Further, if there exist at least three virtual edges of $\text{skel}(\mu)$ containing traversing paths, then each such path belongs to two simple cycles, thus contradicting Property 3. □

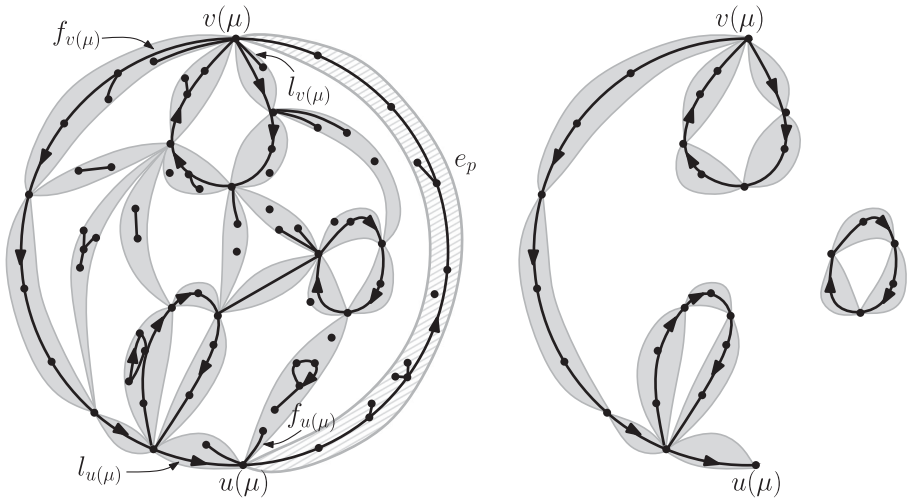

Fig. 9. Illustration for the case in which μ is an R-node. (a) One of the two embeddings of skel(μ). Grey regions represent expansion graphs of the virtual edges of the skeleton of μ. The expansion graph of the parent edge e_p is shaded. Edges in $G(f)$ not in $H(f)$ are not shown. (b) Graph skel$'(\mu)$ obtained by restricting skel(μ) to those edges $e_i \neq e_p$ with $p(e_i) =$ TRUE. The only edges of $H(f)$ shown are those belonging to traversing paths for children of μ in $\mathcal{T}(f)$.

Hence, the skeleton of every P-node contains at most one such cycle, whose right side must be empty by Lemma 4.3. This considerably simplifies the problem of finding a cycle-compatible embedding of a P-node. We are now ready to exhibit the main steps of Algorithm BF.

Algorithm BF. As stated earlier, Algorithm BF performs a bottom-up traversal of the rooted SPQR-tree $\mathcal{T}(f)$ of $G(f)$ such that for each processed node μ, a compatible embedding of skel(μ) is computed, if it exists. The algorithm computes the edges $f_{u(\mu)}$, $l_{u(\mu)}$, $f_{v(\mu)}$, and $l_{v(\mu)}$ for each node μ of $\mathcal{T}(f)$ (and for each virtual edge in the skeleton of each node of $\mathcal{T}(f)$) as in Algorithm BC to find edge-compatible embeddings. Further, it computes the flags $p(\mu)$ and $uv(\mu)$ for each processed node to identify facial cycles of f that project to cycles in skel(μ). We now give a detailed description of how Algorithm BF processes a node μ, assuming that all flags and edges for all children of μ have already been computed.

If μ is a Q- or an S-node, no check is needed. As skel(μ) is a cycle, the only planar embedding of skel(μ) is compatible with $\mathcal{H}(f)$. Edges $f_{u(\mu)}$, $l_{u(\mu)}$, $f_{v(\mu)}$, and $l_{v(\mu)}$, as well as flags $p(\mu)$ and $uv(\mu)$, can easily be computed. In particular, by Lemma 4.4, if μ is an S-node, then $p(\mu) = p(\mu_i)$ for any child μ_i of μ.

If μ is an R-node, as in Figure 9(a), then for each of the two planar embeddings of skel(μ), check if it is edge-compatible with $\mathcal{H}(f)$ and set values for $f_{u(\mu)}$, $l_{u(\mu)}$, $f_{v(\mu)}$, and $l_{v(\mu)}$ as in Algorithm BC. To check if any of the two embeddings is cycle-compatible with $\mathcal{H}(f)$, we check if the conditions of Lemma 4.5 are satisfied. To perform this test, we need, for each virtual edge $e = uv$ of skel(μ), the corresponding flags $p(e)$ and $uv(e)$. This information is already known for all virtual edges, except the parent edge e_p of skel(μ). To compute the flags for e_p, we need to determine whether the virtual edge e_p contains a traversing path P_p and, if it does, determine its orientation. By definition of traversing path, P_p exists if and only if there exists a traversing path in pert(μ). Restrict skel(μ) to those edges $e_i \neq e_p$ with $p(e_i) =$ TRUE, and denote by skel$'(\mu)$ the

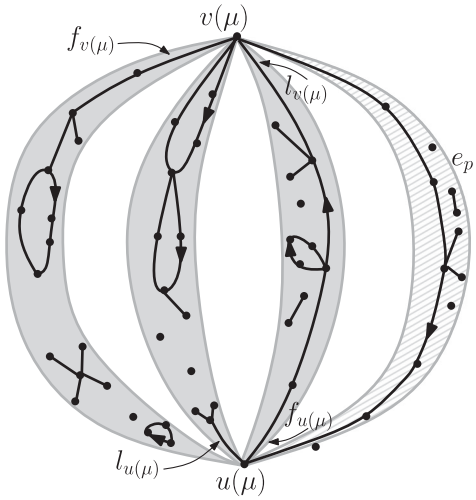

Fig. 10. Illustration for the case in which μ is a P-node. An embedding of skel(μ) is shown. Grey regions represent expansion graphs of the virtual edges of the skeleton of μ. The expansion graph of the parent edge e_p is shaded. Edges in $G(f)$ not in $H(f)$ are not shown.

resulting graph. Refer to Figure 9(b). Note that by Property 3, for each virtual edge $e_i \in \text{skel}'(\mu)$, there exists exactly one traversing path in pert(e_i), and this traversing path is contained in exactly one simple cycle of $H(f)$. In addition, for each simple cycle $\vec{C}$ of $H(f)$ passing through a vertex of skel$'(\mu)$, there exist exactly two virtual edges of skel$'(\mu)$ incident to this vertex that contain a traversing path that is part of $\vec{C}$. These two observations imply that a traversing path P_p exists if and only if both the degree of $u(\mu)$ and the degree of $v(\mu)$ in skel$'(\mu)$ are odd. Hence, if both the degree of $u(\mu)$ and the degree of $v(\mu)$ are odd, then set $p(\mu) = \textsc{True}$ and $p(e_p) = \textsc{True}$; otherwise, set $p(\mu) = \textsc{False}$ and $p(e_p) = \textsc{False}$. In the former case, the orientation of P_p is the only one that makes the number of edges e_i incident to $u(\mu)$ with $uv(e_i) = \textsc{True}$ equal to the number of edges e_i incident to $u(\mu)$ with $uv(e_i) = \textsc{False}$; this determines $uv(\mu)$ and $uv(e_p)$.

Now, $p(e_i)$ and $uv(e_i)$ are defined for every virtual edge e_i of skel(μ). Consider every face g of skel(μ) and denote by $e_j = (u_j, v_j)$ any edge incident to g. Suppose, without loss of generality, that g is to the right of e_j when traversing it from u_j to v_j. Then, check if $p(e_j) = \textsc{False}$, or $p(e_j) = \textsc{True}$ and $uv(e_j) = \textsc{False}$ for all edges e_j incident to g, and check whether $p(e_j) = \textsc{True}$ and $uv(e_j) = \textsc{True}$ for all edges e_j incident to g. If one of the two checks succeeds, the face does not violate Lemma 4.5, but otherwise it does.

If μ is a P-node, as in Figure 10, check if an embedding of skel(μ) exists that is compatible with $\mathcal{H}(f)$ as follows. By Lemma 4.6, there exist either zero or two virtual edges of skel(μ) containing a traversing path. Then, consider the children μ_i of μ such that $p(\mu_i) = \textsc{True}$. If zero or two such children exist, then the parent edge of skel(μ) has no traversing path; if one such a child exists, then the parent edge of skel(μ) has a traversing path. Denote by e_i and e_j the edges of skel(μ) containing a traversing path, if such edges exist, where possibly e_j is the parent edge (in this case, set $p(e_j) = \textsc{True}$, and set $uv(e_j) = \textsc{True}$ if $uv(e_i) = \textsc{False}$ and $uv(e_j) = \textsc{False}$ otherwise). If there exists no edge e_i of skel(μ) such that $p(e_i) = \textsc{true}$, then construct an embedding of skel(μ) that is edge-compatible with $\mathcal{H}(f)$, if possible, as in Algorithm BC; as there exists no facial cycle of $\mathcal{H}(f)$ whose edges belong to distinct virtual edges of skel(μ), then an edge-compatible embedding is also cycle-compatible with $\mathcal{H}(f)$. Edges $f_{u(\mu)}$, $l_{u(\mu)}$, $f_{v(\mu)}$,

and $l_{v(\mu)}$ are computed as in Algorithm BC. Set flag $p(\mu) = \text{FALSE}$. If there exist two edges e_i and e_j such that $p(e_i) = \text{TRUE}$, $p(e_j) = \text{TRUE}$, and $p(e_l) = \text{FALSE}$ for every edge $e_l \neq e_i, e_j$, suppose that $uv(e_i) = \text{TRUE}$ and $uv(e_j) = \text{FALSE}$, the case in which $uv(e_i) = \text{FALSE}$ and $uv(e_j) = \text{TRUE}$ being analogous. Note that the expansion graphs of e_i and e_j must contain at least one H-edge incident to $u(\mu)$ as well as at least one H-edge incident to $v(\mu)$. By Lemma 4.3, e_j has to immediately precede e_i in the counterclockwise order of the edges incident to $u(\mu)$. Then, construct L_u and L_v as in Algorithm BC; check whether L_u and L_v, restricted to the edges that appear in both lists, are the reverse of each other; and further, check whether e_j precedes e_i in L_u and whether e_i precedes e_j in L_v. If the checks are positive, construct the list L of all edges of skel(μ) as in Algorithm BC, except for the fact that the edges of skel(μ) not in L_u and not in L_v are not inserted between e_j and e_i. Edges $f_{u(\mu)}$, $l_{u(\mu)}$, $f_{v(\mu)}$, and $l_{v(\mu)}$ are computed as in Algorithm BC. Set $p(\mu) = \text{FALSE}$ if e_j corresponds to a child μ_j of μ and $p(\mu) = \text{TRUE}$ if e_j is the parent edge of μ; in the latter case, $uv(\mu) = \text{TRUE}$ if $uv(\mu_i) = \text{TRUE}$ and $uv(\mu) = \text{FALSE}$ otherwise.

We get the following theorem.

THEOREM 4.7. *Let $(G(f), H(f), \mathcal{H}(f))$ be a biconnected* PEG *with n vertices such that $G(f)$ and $H(f)$ have the same vertex set, all vertices and edges of $H(f)$ are incident to the same face f of $\mathcal{H}(f)$, and no edge of $G(f) \setminus H(f)$ connects two vertices belonging to the same block of $H(f)$. Algorithm BF solves* PEP *for $(G(f), H(f), \mathcal{H}(f))$ in $O(n)$ time.*

PROOF. We show that Algorithm BF processes each node μ of $\mathcal{T}(f)$ in $O(k_\mu)$ time, where $\mu_1, \ldots, \mu_{k_\mu}$ are the children of μ in $\mathcal{T}(f)$.

Observe that the computation of $f_{u(\mu)}$, $l_{u(\mu)}$, $f_{v(\mu)}$, and $l_{v(\mu)}$ and the check of edge-compatibility are done as in Algorithm BC; hence, they take $O(k_\mu)$ time. We describe how to check the cycle compatibility of an embedding of skel(μ) in $O(k_\mu)$ time.

If μ is a Q-node or an S-node, Algorithm BF neither performs any checks nor does it make any embedding choices.

If μ is a P-node, then Algorithm BF performs the same checks and embedding choices as Algorithm BC, plus the check that the two edges e_i and e_j with $p(e_i) = \text{TRUE}$ and $p(e_j) = \text{FALSE}$ (notice that one of these edges could be the the parent edge of μ) are consecutive (with the right order) in L_u and L_v. This is done in constant time. Flags $p(\mu)$ and $uv(\mu)$ are computed in $O(k_\mu)$ time, by simply checking the flags $p(\mu_i)$ and $uv(\mu_i)$, for $i = 1, \ldots, k$.

Suppose that μ is an R-node. The construction of skel$'(\mu)$ can easily be done in $O(k_\mu)$ time, as such a graph can be obtained from skel(μ) by simply checking flag $p(e_i)$, for each edge e_i in skel(μ). Then, the degree of $u(\mu)$ and $v(\mu)$ in skel$'(\mu)$, as well as the flags $p(\mu)$, $uv(\mu)$, $p(e_p)$ and $uv(e_p)$, can be computed in total $O(k_\mu)$ time. The test on each face takes time linear in the number of edges incident to the face. Namely, such a test consists of two checks, each of which requires considering a constant number of flags associated with each edge of the face. As every edge is incident to two faces of skel(μ) and the number of edges in skel(μ) is $O(k_\mu)$, the total time spent for the test on the faces of skel(μ) is $O(k_\mu)$.

As $\sum_{\mu \in \mathcal{T}} k_\mu = O(n)$, the total running time of the algorithm is $O(n)$. □

4.3. *G* Biconnected

In this section, we show how to solve PEP for general biconnected PEGs—that is PEGs $(G, H, \mathcal{H})$ where G is biconnected and H is arbitrary. The algorithm employs the algorithms from the previous two sections as subroutines. The general outline is as follows. First, compute a subgraph H^+ of G with the following properties: (i) H^+ is biconnected; (ii) H is a subgraph of H^+; and (iii) H^+ contains every nonlocal H-bridge of G. Second, solve instance $(H^+, H, \mathcal{H})$ obtaining an embedding $\mathcal{H}^+$ of H^+ extending $\mathcal{H}$, if H^+ admits

one. We will show that this step can be reduced to several applications of Algorithm BF. Finally, solve instance $(G, H^+, \mathcal{H}^+)$ with Algorithm BC—we will see that H^+ is connected (even biconnected), and hence the algorithm can be applied.

In a first step, we ensure that all nonlocal H-bridges of G are trivial. Recall that every nontrivial nonlocal H-bridge K has at most one candidate face f_K of $\mathcal{H}$ where it can be embedded. We obtain the graph H' with embedding $\mathcal{H}'$ as in Definition 3.7 by adding the vertices of $K - V(H)$ to H and embedding them into the face f_K for each nontrivial nonlocal H-bridge K of G.

Let H^+ be the graph obtained from G by removing the vertices and edges (but not the attachments) of all local H-bridges of G. Note that $H' \subseteq H^+$, that H^+ and H' have the same vertex set, and that any embedding of H^+ that extends $\mathcal{H}$ also extends $\mathcal{H}'$ and vice versa.

Each H'-bridge K of H^+ is nonlocal, and therefore there exists a unique face f_K where it needs to be embedded. Since H-bridges that are embedded in distinct faces of $\mathcal{H}$ do not interact, we can solve the instances stemming from the faces of $\mathcal{H}$ independently, which enables us to use Algorithm BF to find an embedding extension of H^+. This motivates the following definitions, which take a more local view at the PEG $(H^+, H', \mathcal{H}')$. Let f be a face of $\mathcal{H}'$ and let $V(f)$ be the set of vertices of H' that are incident to f. Let $H(f)$ be the subgraph of H' induced by $V(f)$, let $\mathcal{H}(f)$ be $\mathcal{H}'$ restricted to $H(f)$, and let $G(f)$ be the subgraph of H^+ induced by $V(f)$. By construction, in any embedding of H^+ that extends $\mathcal{H}$, the edges of $G(f)$ not belonging to $H(f)$ are embedded inside f.

Our approach is to first find an embedding $\mathcal{H}^+$ of H^+ that extends $\mathcal{H}'$ (i.e., solve PEP for $(H^+, H', \mathcal{H}')$) and then find an embedding $\mathcal{G}$ for $(G, H^+, \mathcal{H}^+)$ (i.e., solve PEP for $(G, H^+, \mathcal{H}^+)$). The latter step is actually simple, as H^+ is biconnected and thus connected. Therefore, Algorithm BC can be used to solve this subproblem.

LEMMA 4.8. *H^+ is biconnected.*

PROOF. By construction of H^+, each H^+-bridge of G has all of its attachment vertices in the same block of H, and hence in the same block of H^+, as H is a subgraph of H^+. Therefore, the number of blocks of H^+ is not modified by the addition of the H^+-bridges of G. Since such an addition produces G, which is biconnected, it follows that H^+ is biconnected. □

Clearly, if $(G, H, \mathcal{H})$ is planar, then an embedding $\mathcal{G}$ of G extending $\mathcal{H}$ exists, and the restriction of $\mathcal{G}$ to H^+ yields an intermediate embedding $\mathcal{H}^+$ of H^+ extending $\mathcal{H}'$, which can then be extended to an embedding of G extending $\mathcal{H}$. We show that the choice of $\mathcal{H}^+$ does not change the possibility of finding such an embedding extension. In particular, if $\mathcal{H}_1^+$ and $\mathcal{H}_2^+$ are two embeddings of H^+ extending $\mathcal{H}$, then the PEG $(G, H^+, \mathcal{H}_1^+)$ is planar if and only if $(G, H^+, \mathcal{H}_2^+)$ is planar.

LEMMA 4.9. *A biconnected* PEG $(G, H, \mathcal{H})$ *is planar if and only if (a) H^+ admits a planar embedding extending $\mathcal{H}$ and (b) for every planar embedding $\mathcal{H}^+$ of H^+, $(G, H^+, \mathcal{H}^+)$ is planar.*

PROOF. Clearly, if conditions (a) and (b) hold, then G has an embedding extending $\mathcal{H}$.

To prove the converse, assume that G has an embedding $\mathcal{G}$ extending $\mathcal{H}$. Clearly, $\mathcal{G}$ contains a subembedding $\mathcal{H}^+$ of H^+ that extends $\mathcal{H}$, so condition (a) holds. It remains to prove that condition (b) holds as well.

First, we introduce some terminology. Let f be any face of $\mathcal{H}$ and let $\mathcal{H}^+$ be any embedding of H^+ that extends $\mathcal{H}$. In $\mathcal{H}^+$, the face f can be partitioned (by the edges of H^+ not in H) into several faces, which we will call the *subfaces of f*. A set of vertices $S \subseteq V(H)$ is said to be *mutually visible in f with respect to $\mathcal{H}^+$* if $\mathcal{H}^+$ has a subface of f that contains all vertices of S on its boundary.

The proof that condition (b) holds is based on two claims. The first one shows that for the vertices that belong to the same block of H, mutual visibility is independent of the choice of $\mathcal{H}^+$.

Claim 3. Let $\vec{C}$ be a facial cycle of f and let $S \subseteq V(\vec{C})$ be a set of vertices of $\vec{C}$. If the vertices in S are mutually visible in f with respect to at least one embedding of H^+ that extends $\mathcal{H}$, then they are mutually visible in f with respect to every embedding of H^+ that extends $\mathcal{H}$.

Note that the mutual visibility of S in f only depends on the embedding $\mathcal{H}^+$ restricted to $G(f)$. Let $\mathcal{T}$ be the SPQR-tree of $G(f)$. By Theorem 3.6, the embeddings of $G(f)$ that extend $\mathcal{H}(f)$ are exactly obtained by specifying a compatible embedding for the skeleton of each node of $\mathcal{T}$. Assume that $\mathcal{G}_1$ and $\mathcal{G}_2$ are two embeddings of $G(f)$ that extend $\mathcal{H}$. Assume that the vertices of S are mutually visible in f with respect to $\mathcal{G}_1$. We will show that they are also mutually visible with respect to $\mathcal{G}_2$. In view of Theorem 3.6, we may assume that $\mathcal{G}_2$ was obtained from $\mathcal{G}_1$ by changing the embedding of the skeleton of a single node $\mu \in \mathcal{T}$.

Let us distinguish two cases, depending on whether $\vec{C}$ is contained in the expansion graph of a single virtual edge of skel(μ) or whether it projects to a cycle in skel(μ).

If $\vec{C}$ is part of the expansion graph of a single virtual edge $e = \{x, y\} \in \text{skel}(\mu)$, then let $\mathcal{G}_e$ be the embedded graph obtained as the union of the expansion graph of e and a single edge connecting x and y, embedded in the outer face of the expansion graph. We easily see that the vertices in S are mutually visible in f if and only if they share the same face of $\mathcal{G}_e$, other than the face that is to the right of $\vec{C}$. Since $\mathcal{G}_e$ does not depend on the embedding of skel(μ), the vertices in S are mutually visible in $\mathcal{G}_2$.

Assume now that the cycle $\vec{C}$ projects to a cycle $\vec{C}'$ in skel(μ). By Lemma 4.3, in any compatible embedding of skel(μ), all vertices and edges of skel(μ) that do not belong to $\vec{C}'$ are embedded to the left of $\vec{C}'$. In particular, if μ is an R-node, then skel(μ) only has a single compatible embedding. Thus, μ must be a P-node. Let e and e' be the two virtual edges of skel(μ) that form $\vec{C}'$. In each compatible embedding of skel(μ), these two edges must be embedded next to each other and in the same order. It follows easily that any two compatible embeddings of skel(μ) yield embeddings of $G(f)$ in which the vertices from S have the same mutual visibility. This completes the proof of the claim.

Let us proceed with the proof that condition (b) holds. We need more terminology. Let K and K' be a pair of local H-bridges of G whose attachments all appear on a facial cycle $\vec{C}$ of a face f in $\mathcal{H}$. We say that K and K' have a *three-vertex conflict on* $\vec{C}$ if they share at least three attachments, and that they have a *four-vertex conflict on* $\vec{C}$ if there are four vertices x, x', y, y' that appear on $\vec{C}$ in this cyclic order, and x, y are attachments of K, whereas x', y' are attachments of K'.

Claim 4. Assume that a face f_K of $\mathcal{H}$ has been assigned to every local H-bridge K of G so that all attachments of K are on the boundary of f_K. Let $\mathcal{H}^+$ be an embedding of H^+ extending $\mathcal{H}$. There is an embedding $\mathcal{G}$ of G extending $\mathcal{H}^+$, with the additional property that each local H-bridge K is embedded inside a subface of f_K if and only if:

(1) For any local H-bridge K, all attachments of K are mutually visible in f_K with respect to $\mathcal{H}^+$.
(2) If K and L are distinct local H-bridges assigned to the same face $f_K = f_L$ such that the attachments of K and L appear on a common facial cycle $\vec{C}$ of $\mathcal{H}^+$, then K and L have no conflict on $\vec{C}$.

Clearly, the two conditions are necessary. To prove that they are also sufficient, assume that both the conditions hold. Construct an embedding of $\mathcal{G}$ with the desired properties as follows. Let f be any face of $\mathcal{H}$. Observe that the first condition of the claim guarantees that for every H-bridge K assigned to f, there is a face f' of $\mathcal{H}^+$ that is a subface of f such that all attachments of K appear on the boundary of f'. Let f' be a face of $\mathcal{H}^+$ that is a subface of f, and let $K_1, \dots K_s$ be all local H-bridges that were assigned to f and whose attachments all appear on the boundary of f'. We show that all bridges $K_1, \dots, K_s$ can be embedded inside f'.

First, observe that the boundary of f' is a simple cycle C', because H^+ is biconnected. In addition, observe that no two bridges K_i and K_j have a conflict on C', by the second condition of the claim. To show that all bridges $K_1, \dots, K_s$ can be embedded inside C', proceed by induction on s. If $s = 1$, the statement is clear. Assume that $s \geq 2$ and that the bridge K_1 has been successfully embedded into f'. The embedding of K_1 partitions f' into several subfaces $f'_1, \dots, f'_t$. Such subfaces are again bounded by simple cycles; otherwise, G would not be biconnected. We claim that for every bridge K_i, with $i \geq 2$, there is a subface f'_j containing all attachments of K_i. Consider any bridge K_i. Assume first that K_i has an attachment x that is not an attachment of K_1. Then, x belongs to a unique subface f'_j. Hence, if K_i has another attachment not belonging to f'_j, there is a four-vertex conflict of K_1 and K_i on $\vec{C}'$, contradicting the second condition of the claim. Assume next that each attachment of K_i is also an attachment of K_1. Then, K_i has exactly two attachments, and if such attachments do not share a face f'_j, a four-vertex conflict of K_1 and K_i on $\vec{C}'$ is created, again contradicting the second condition of the claim.

We can thus assign to each K_i a subface f'_j that contains all of its attachments. By induction, all K_i's can be embedded into their assigned faces, thus proving the second claim.

The proof that condition (b) holds follows easily from the two claims. Namely, assume that G has an embedding $\mathcal{G}$ extending $\mathcal{H}$. Let $\mathcal{H}^+$ be $\mathcal{G}$ restricted to H^+. For every local H-bridge K of G, let f_K be the face of $\mathcal{H}$ inside which K is embedded in $\mathcal{G}$. Clearly, $\mathcal{H}^+$ satisfies the two conditions of the second claim, as it can be extended into $\mathcal{G}$. Then, every embedding of H^+ that extends $\mathcal{H}$ satisfies the two conditions of the second claim: for the first condition, this is a consequence of the first claim, and for the second condition, this is obvious. We conclude that every embedding of H^+ that extends $\mathcal{H}$ can be extended into an embedding of G, thus proving condition (b) and hence the lemma. □

As stated earlier, each H'-bridge K of $(H^+, H', \mathcal{H}')$ is nonlocal, and we therefore know into which face f_K it needs to be embedded. Since H'-bridges that are embedded in different faces do not interact, we can solve the subinstance $(G(f), H(f), \mathcal{H}(f))$ arising from each face separately. Clearly, if one of the instances fails, then G does not have an embedding extension. If all instances admit embedding extensions, gluing them together yields an embedding $\mathcal{H}^+$ of H^+ extending $\mathcal{H}'$. The previous lemma then implies that $(G, H, \mathcal{H})$ is planar if and only if $(G, H^+, \mathcal{H}^+)$ is planar. We are now ready to describe Algorithm BA (for G Biconnected and H Arbitrary).

Algorithm BA. Starting from an instance $(G, H, \mathcal{H})$ of PEP, graphs $G(f)$ and $H(f)$, and embedding $\mathcal{H}(f)$, for every face f of $\mathcal{H}$, are computed as follows. For each H-bridge K of G, determine whether it is local to a block of H or not. In the former case, K is not associated to any face f of $\mathcal{H}$. In the latter case, we compute the unique face f of $\mathcal{H}$ in which K has to be embedded in any solution of instance $(G, H, \mathcal{H})$ of PEP, and we associate K with f.

These computations can be performed in linear time by applying Lemma 2.2. To do so, we have to construct the $\mathcal{CF}$-tree of $\mathcal{H}$, the $\mathcal{BF}$-tree of $\mathcal{H}$, the $\mathcal{VF}$-graph of $\mathcal{H}$, and the enriched block-cutvertex tree of each connected component of H. As shown in Section 2.3, this can be done in linear time.

Then, for each face f of $\mathcal{H}$, consider every H-bridge K associated with f. Add the vertices and the edges of K to $G(f)$, and add the vertices of K to $\mathcal{H}(f)$ inside f. Let $H^+ = \bigcup_{f \in H} G(f)$. For each face f of $\mathcal{H}$, call Algorithm BF with input $(G(f), H(f), \mathcal{H}(f))$. If Algorithm BF succeeds for every instance $(G(f), H(f), \mathcal{H}(f))$ (thus providing an embedding $\mathcal{H}^+(f)$ of $G(f)$ whose restriction to $H(f)$ is $\mathcal{H}(f)$), merge the embeddings $\mathcal{H}^+(f)$ of $G(f)$ into a planar embedding $\mathcal{H}^+$ of H^+. Finally, call Algorithm BC with $(G, H^+, \mathcal{H}^+)$.

THEOREM 4.10. *Let $(G, H, \mathcal{H})$ be an n-vertex instance of* PEP *such that G is biconnected. Algorithm BA solves* PEP *for $(G, H, \mathcal{H})$ in $O(n)$ time.*

PROOF. The correctness of the algorithm follows from Lemma 4.9.

By Lemma 2.2, determining whether an H-bridge K is local or not can be done in time linear in the size of K. Further, if K is nonlocal, the only face of $\mathcal{H}$ incident to all attachment vertices of K can be computed, if it exists, in time linear in the size of K. Then, the construction of graphs $G(f)$, $H(f)$, H^+ and of embeddings $\mathcal{H}(f)$ takes $O(n)$ time, as it only requires to perform the union of graphs that have total $O(n)$ edges.

By Theorem 4.7, Algorithm BF runs in time linear in the number of edges of $G(f)$, and hence all executions of Algorithm BF take a total $O(n)$ time. By Theorem 4.2, Algorithm BC runs in $O(n)$ time, and hence the total running time of Algorithm BA is $O(n)$. □

This concludes the case of biconnected PEGs.

4.4. *G* Connected or Disconnected

In this section, we give an algorithm that decides the planarity of general PEGs. First, we deal with instances $(G, H, \mathcal{H})$ of PEP in which G is connected, every nontrivial H-bridge of G is local, and H is arbitrary. We show that the three conditions of Lemma 3.8 can be checked in linear time. The first condition can be checked in linear time by Theorem 4.10. The second and the third conditions can be checked in linear time by the following two lemmas.

LEMMA 4.11. *Let $(G, H, \mathcal{H})$ be a connected* PEG. *Let $G_1, \ldots, G_t$ be the blocks of G, and let H_i be the subgraph of H induced by the vertices of G_i. There is a linear-time algorithm that checks whether any two distinct graphs among $H_1, \ldots, H_t$ alternate around any vertex of $\mathcal{H}$.*

PROOF. Let us describe the algorithm that performs the required checks. We assume that every edge e of H has an associated label indicating the block of G that contains e. We also associate to each block two integer counters that will be used in the algorithm.

We now describe a procedure TEST(x), which, for a given vertex $x \in V(H)$, checks whether any two graphs H_i, H_j alternate around x. Let us use the term *x-edge* to refer to any edge of H incident to x, and let *x-block* refer to any block of G that contains at least one x-edge.

The procedure TEST(x) proceeds as follows. First, for every x-block G_i, it determines the number of x-edges in G_i and stores this in a counter associated with G_i. This is done by simply looking at every edge incident to x and incrementing the counter of the corresponding block. Next, TEST(x) visits all x-edges in the order determined by the rotation $\sigma_{\mathcal{H}}(x)$, starting at an arbitrary x-edge. For each x-block, it maintains in a counter the number of its x-edges that have been visited so far. An x-block is *active* if some but not all of its x-edges have already been visited.

The procedure TEST(x) also maintains a stack containing the active x-blocks. At the beginning of the procedure, the counters of visited edges of each x-block are set to zero and the stack is empty.

For every edge e that TEST(x) visits, it performs the following steps:

(1) Let G_i denote the block containing e. Increment the counter of visited x-edges of G_i.
(2) If no other edge of G_i has been visited so far, push G_i on the stack.
(3) If some x-edge of G_i has been visited before e, we know that G_i is currently somewhere on the stack. Check whether G_i is on the top of the stack. If the top of the stack contains an x-block G_j different from G_i, output that H_i and H_j alternate around x and stop.
(4) Check whether e is the last x-edge of G_i to be visited (comparing its counter of visited x-edges to the counter of total x-edges), and if it is, pop G_i from the stack. (Note that if G_i has only one x-edge, it is pushed and popped during the visit of this edge.)

If TEST(x) visits all x-edges without rejecting, it outputs that there is no alternation around x.

The procedure TEST(x) takes time proportional to the number of x-edges. Thus, we can call TEST(x) for all vertices $x \in V(H)$ in linear time to test whether there is any alternation in $\mathcal{H}$.

Let us now argue that the procedure TEST(x) is correct. Assume that TEST(x) outputs an alternation of H_i and H_j. This can only happen when G_j is on the top of the stack while an x-edge $e \in G_i$ is visited, and furthermore, e is not the first edge of G_i to be visited. It follows that the first edge of G_i was visited before the first edge of G_j, and G_j is still active when e is visited. This shows that H_i and H_j indeed alternate around x.

Conversely, assume that there is a pair of graphs H_i and H_j that alternate around x, and the alternation is witnessed by two pairs of x-edges $e, e' \in H_i$ and $f, f' \in H_j$. For contradiction, assume that TEST(x) outputs that there is no alternation. Without loss of generality, assume that at least one x-edge of H_i is visited before any x-edge of H_j, that e is visited before e', and that f is visited before f'. Thus, the four x-edges are visited in the order e, f, e', f'. When the procedure visits e', both G_i and G_j are active, and G_j is on the stack above G_i since we assumed that the first x-edge of G_i is visited before the first x-edge of G_j. This means that when TEST(x) visited e', G_i was not on the top of the stack and an alternation should have been reported.

This contradiction completes the proof of the lemma. □

The next lemma shows that the third condition of Lemma 3.8 can also be tested in linear time, assuming that the first and second conditions of the lemma hold.

LEMMA 4.12. *Let $(G, H, \mathcal{H})$ be a connected* PEG. *Let $G_1, \ldots, G_t$ be the blocks of G, and let H_i be the subgraph of H induced by the vertices of G_i. Let $\mathcal{H}_i$ be $\mathcal{H}$ restricted to H_i. Assume that the following conditions hold:*

(1) each nontrivial H-bridge of G is local,
(2) each G_i has an embedding that extends $\mathcal{H}_i$, and
(3) no two of the graphs $H_1, \ldots, H_t$ alternate around any vertex of H.

There is a linear-time algorithm that decides whether there exists a facial cycle $\vec{C}$ of $\mathcal{H}$ that separates a pair of vertices x and y of H such that x and y are connected by a path of G that has no vertex in common with $\vec{C}$.

PROOF. Let P be a path in G with end-vertices in H, and let $\vec{C}$ be a facial cycle of $\mathcal{H}$. If P and $\vec{C}$ are vertex-disjoint and the end-vertices of P are separated by $\vec{C}$, we say that P and $\vec{C}$ form a *PC-obstruction*. A PC-obstruction $(P, \vec{C})$ is called *minimal* if

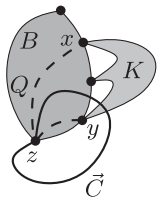

Fig. 11. Illustration for the case in which K is local to a block B of H. Two edges of the path $Q \subseteq B \subseteq G_i$ connecting x and y in H alternate with two edges of cycle $\vec{C} \in G_j$ around vertex z, which is a contradiction to condition 3 of the lemma.

no proper subpath $P' \subset P$ forms a PC-obstruction with $\vec{C}$. Observe that in a minimal PC-obstruction, all internal vertices of P belong to $V(G) \backslash V(H)$.

We want to show that the existence of a PC-obstruction can be tested in linear time. Of course, it is sufficient to test the existence of a minimal PC-obstruction. Before we explain how this test is done, we make some more observations concerning the structure of minimal PC-obstructions.

Let $(P, \vec{C})$ be a minimal PC-obstruction, and let x and y be the end-vertices of P. As the internal vertices of P belong to $V(G) \backslash V(H)$, P is a subgraph of an H-bridge K, and x and y are among the attachments of K. Let us now distinguish two cases, depending on whether K is local to some block or not.

First, assume that K is local to a block B of H. Refer to Figure 11. Then, both B and P are part of the same block G_i of G. Hence, $\vec{C}$ belongs to a different block of G, because if it belonged to G_i, then G_i would contain the whole PC-obstruction $(P, \vec{C})$ and it would be impossible to extend the embedding $\mathcal{H}_i$ to G_i, thus contradicting condition 2 of the lemma. Then, let G_j be the block of G that contains $\vec{C}$. Since x and y belong to a common block B of H, they are connected by a path $Q \subseteq B$. Since x and y are separated by $\vec{C}$, Q shares a vertex z with $\vec{C}$ (otherwise, the embedding $\mathcal{H}$ would not be planar). Since Q and $\vec{C}$ belong to distinct blocks, z is their unique common vertex. This, together with the fact that $\vec{C}$ separates x and y, implies that the two edges that belong to Q alternate with the two edges that belong to $\vec{C}$ in the rotation of z. Thus, G_i alternates with G_j around z, contradicting condition 3 of the lemma. Then, K cannot be a local bridge.

Suppose now that K is nonlocal. By condition 1 of the lemma, K consists of a single edge of $E(G) \backslash E(H)$. We conclude that any minimal PC-obstruction $(P, \vec{C})$ has the property that P is a single edge that forms a nonlocal H-bridge of G.

Observe that two vertices x and y belonging to distinct blocks of H are separated by a facial cycle of $\mathcal{H}$ if and only if there is no face of $\mathcal{H}$ to which both x and y are incident.

We are now ready to describe the algorithm that determines the existence of a minimal PC-obstruction. The algorithm tests all edges of $E(G) \backslash E(H)$ one by one. For any such edge e, it determines in constant time whether it is an H-bridge—that is, whether its endpoints x and y belong to H. If it is an H-bridge, it checks whether it is nonlocal in constant time by using Lemma 2.2. For a nonlocal bridge, the algorithm then checks in constant time whether there is a face f of H into which this bridge can be embedded, again using Lemma 2.2. Such a face f, if it exists, is uniquely determined, and its boundary contains x and y.

Overall, for any edge e, the algorithm determines in constant time whether this edge is a nonlocal bridge that is part of a minimal PC-obstruction. Hence, in linear time, we determine whether G has any PC-obstruction, thus concluding the proof of the lemma. □

Combining Lemmas 2.2, 3.8, 4.11, and 4.12 with Theorem 4.10, we obtain the following result.

THEOREM 4.13. PEP *can be solved in linear time when restricted to instances* $(G, H, \mathcal{H})$ *where* G *is connected.*

PROOF. By Lemma 2.2, an instance of PEP where G is connected can be reduced in linear time to an equivalent instance that has the additional property that all nontrivial H-bridges are local. Namely, by Lemma 2.2, we may compute whether an H-bridge K is nonlocal and, in such case, which is the face of $\mathcal{H}$ in which K has to be embedded, in time linear in the size of K. We may thus assume that $(G, H, \mathcal{H})$ is an instance of PEP where G is simply connected and all nontrivial H-bridges in G are local to some block.

To solve PEP for $(G, H, \mathcal{H})$, we present an algorithm based on the characterization of Lemma 3.8. First, we generate all subinstances $(G_i, H_i, \mathcal{H}_i)$ for $i = 1, \ldots, t$, induced by the blocks of G. It is not difficult to see that the subinstances can be generated in linear time. We then solve these subinstances using Algorithm BA, which takes linear time, by Theorem 4.10, since the total size of the subinstances is linear. If any of the subinstances does not have an embedding extension, we reject $(G, H, \mathcal{H})$, and otherwise we continue.

In the next step, we check whether there is a pair of graphs H_i, H_j that have an alternation around a vertex of $\mathcal{H}$. If there is an alternation, we reject the instance, and otherwise we continue. This step can be implemented in linear time, due to Lemma 4.11.

Finally, we check the existence of PC-obstructions, which by Lemma 4.12 can be done in linear time. We accept the instance if and only if we find no PC-obstruction. The correctness of this algorithm follows from Lemma 3.8. □

Next, we deal with the instances $(G, H, \mathcal{H})$ of PEP in which G is disconnected and H is arbitrary. We use Lemma 3.9 directly and show that the two conditions of the lemma can be checked in linear time. The first condition of Lemma 3.9 can be checked in linear time by Theorem 4.13. As the proof of the next theorem shows, the second condition can be tested efficiently as well.

THEOREM 4.14. PEP *can be solved in linear time.*

PROOF. Let $(G, H, \mathcal{H})$ be an instance of PEP. Let $G_1, \ldots, G_t$ be the connected components of G, let H_i be the subgraph of H induced by the vertices of G_i, and let $\mathcal{H}_i$ be $\mathcal{H}$ restricted to H_i.

By Lemma 3.9, $(G, H, \mathcal{H})$ has an embedding extension if and only if each instance $(G_i, H_i, \mathcal{H}_i)$ has an embedding extension and, for $i \neq j$, no facial cycle of $\mathcal{H}_i$ separates a pair of vertices of H_j. By Theorem 4.13, we can test in linear time whether all instances $(G_i, H_i, \mathcal{H}_i)$ have an embedding extension.

It remains to test the existence of a facial cycle of $\mathcal{H}_i$ that separates vertices of H_j. For this test, we use the component-face tree $\mathcal{CF}$ of $\mathcal{H}$. Assume that $\mathcal{CF}$ is rooted at any node representing a face of $\mathcal{H}$; call this face the *root face of* $\mathcal{H}$. A face f is an *outer face of* $\mathcal{H}_j$ if at least one child of f in $\mathcal{CF}$ is a component of H_j but the parent of f does not belong to H_j (which includes the possibility that f is the root face).

We claim that a pair of vertices of H_j is separated by a facial cycle belonging to another component of H if and only if there are at least two distinct outer faces of $\mathcal{H}_j$ in $\mathcal{CF}$. To see this, assume first that $\mathcal{H}_j$ has two distinct outer faces f_1 and f_2, and let C_1 (or C_2) be a component of H_j which is a child of f_1 (or f_2, respectively). Any path from C_1 to C_2 in $\mathcal{CF}$ visits the parent of f_1 or the parent of f_2. These parents correspond to components of H not belonging to H_j, and at least one facial cycle determined by these components separates C_1 from C_2.

Conversely, if C_1 and C_2 are components of H_j separated by a facial cycle belonging to a component C_3 of H_i ($i \neq j$), then the path in $\mathcal{CF}$ that connects C_1 to C_2 visits C_3, and in such a case it is easy to see that $\mathcal{H}_j$ has at least two outer faces.

We now describe the algorithm that tests the second condition of Lemma 3.9. We assume that each connected component of H has associated its corresponding subgraph H_i in $\mathcal{CF}$. We then process the components of H one by one, and for each component C, we check whether its parent node is an outer face of the embedding $\mathcal{H}_i$ of the subgraph H_i containing C. We accept $(G, H, \mathcal{H})$ if and only if each $\mathcal{H}_i$ has one outer face. This algorithm clearly runs in linear time. □

The algorithms for PEP we presented in this section, handling simply connected and disconnected PEGs, are nonconstructive. For simplicity, we preferred to present shorter and nonconstructive versions of these algorithms. We now briefly sketch how they can be extended to constructive linear-time algorithms.

Sketch of constructive algorithms. For the reduction from disconnected to connected PEGs, designing a constructive algorithm is rather simple. Let $(G, H, \mathcal{H})$ be a PEG and let $G_1, \ldots, G_t$ be the connected components of G. Assume that we already have an embedding $\mathcal{G}_i$ for each instance $(G_i, H_i, \mathcal{H}_i)$, where H_i is the subgraph of H contained in G_i and $\mathcal{H}_i$ is the restriction of $\mathcal{H}$ to H_i. Note that each H_i may consist of several connected components. To merge the embeddings $\mathcal{G}_i$ into a single embedding $\mathcal{G}$ that extends $\mathcal{H}$, we make use of the following auxiliary graph. Create a node for each face of $\mathcal{H}$ and a node for each H_i, $i = 1, \ldots, t$. Connect a face f and a component H_i if f is incident to an edge of H_i. For each edge (f, H_i) of this auxiliary graph, additionally store a pointer to some edge $e_{f,H_i} \in H_i$ such that f is to the right of e_{f,H_i}. By Lemma 3.9, this auxiliary graph is a tree T if the instance admits a solution. It can be computed from the component face tree of $\mathcal{H}$ in $O(n)$ time. We use T to guide the merging of the embeddings $\mathcal{G}_i$. Namely, for each face node f, let I_f be the set of indices i so that f is adjacent to H_i. To merge the embeddings $\mathcal{G}_i, i \in I_f$, we find in each of the $\mathcal{G}_i$ a subface of f and merge their face boundaries. To find a subface of f in G_i, we use the additional information stored in T. Namely, T contains the edge (f, H_i), and using the pointer stored there, we find the edge e_{f,H_i} of H_i. Recall that f lies to the right of e_{f,H_i} in $\mathcal{H}$. Let f' be the face that lies to the right of e_{f,H_i} in $\mathcal{G}_i$. Clearly, f' is a subface of f, and it can be found in $O(1)$ time. Thus, the merge step for face f can be implemented to run in $O(|I_f|)$ time, and the tree structure of T implies that the total time for merging all faces is $O(n)$.

Let us now consider the reduction from connected to biconnected PEGs. Recall that for a cutvertex x of G, an x-edge is an edge of H incident to x, and an x-block is a block of G containing at least one x-edge. Observe that the procedure TEST(x) described in the proof of Lemma 4.11 does not just check whether, around each cutvertex x, the x-blocks $B_1, \ldots, B_t$ of G have a parenthetical structure, but it can actually be employed to find an ordering of $B_1, \ldots, B_t$ such that the blocks can be removed one by one in such a way that the x-edges of B_i form an interval when B_i is removed. Further, the check whether a trivial H-bridge is part of a PC-obstruction actually reveals a unique face of $\mathcal{H}$ into which the block containing the H-bridge has to be embedded. We use an arbitrary such H-bridge to determine the correct face into which any block that does not contain any x-edge has to be embedded. This either gives a correct embedding or one of the conditions of Lemma 4.11 or Lemma 4.12 is violated, in which case an embedding does not exist. In the following, we assume that we have a feasible instance.

Our procedure for handling simply connected graphs is as follows. We first compute the block-cutvertex tree $\mathcal{BC}$ of G and use Lemma 2.2 to ensure that all nonlocal H-bridges are trivial. Let $G_1, \ldots, G_t$ be the blocks of G and let $H_1, \ldots, H_t$ be the subgraphs of H induced by $G_1, \ldots, G_t$, together with the embeddings $\mathcal{H}_1, \ldots, \mathcal{H}_t$ induced

by $\mathcal{H}$. First, we compute in linear time an embedding extension $\mathcal{G}_i$ for each instance $(G_i, H_i, \mathcal{H}_i)$. Next, we merge embeddings $\mathcal{G}_i$ into a single embedding $\mathcal{G}$ extending $\mathcal{H}$, if possible. To this end, we need to merge the rotation systems of the embeddings $\mathcal{G}_i$ at the cutvertices of G. First, we iteratively remove each leaf block of $\mathcal{BC}$ that does not contain any vertex of H, except, possibly, for the unique cutvertex of G that it contains. Clearly, the removed blocks can easily be embedded later, as they are not subject to any constraints. The remaining instance (let us denote it again by $(G, H, \mathcal{H})$) is still connected. Moreover, by the assumption that all nonlocal H-bridges are trivial, all cutvertices of G belong to H. Therefore, every block contains at least two vertices of H (namely, every block of G that is not a leaf in $\mathcal{BC}$ contains at least two cutvertices, and every block of G that is a leaf in $\mathcal{BC}$ contains at least two vertices of H, as otherwise it would have been removed).

Let x be a cutvertex of G and let $B_1, \ldots, B_t$ be the blocks incident to x. Denote by $\mathcal{B}_i$ the embedding of B_i that has been already computed and that extends the restriction of embedding $\mathcal{H}$ to the vertices of H in B_i. Assume that the ordering of the blocks incident to x is such that $B_1, \ldots, B_k$ contain an x-edge, whereas $B_{k+1}, \ldots, B_t$ do not. We embed the blocks $B_1, \ldots, B_k$ by using a modified version of the procedure TEST(x), described in the proof of Lemma 4.11. For each i in $k+1, \ldots, t$, let $e = xy$ be any edge incident to x in B_i. Since B_i contains an H-vertex distinct from x, vertex y belongs to H as well; otherwise, e would be part of a nontrivial nonlocal H-bridge. Since edge e does not belong to H, we have that x and y belong to distinct connected components of H. We use the component face tree $\mathcal{CF}$ to find in $O(1)$ time the unique face f_i of $\mathcal{H}$ that is shared by x and y. We associate B_i with f_i. Although the choice of e is arbitrary, either all edges of B_i incident to x yield the same face or at least one of them is part of a PC-obstruction, which we can rule out by first running the checking algorithm.

We now construct the cyclic ordering of all edges of G incident to x by a single traversal of $\sigma_{\mathcal{H}}(x)$, similarly to procedure TEST(x), except we alternately visit edges and faces as they occur in counterclockwise order around x in $\mathcal{H}$.

When the procedure visits a face f, it appends the edges of all blocks associated with this face in the order as they occur in the embedding of the block. More precisely, let B_i be any block associated with f, let e be any edge of B_i incident to x, and let e' be the predecessor of e in the counterclockwise ordering of x in $\mathcal{B}_i$. Then, the counterclockwise order of the edges incident to x in $\mathcal{B}_i$ forms a sequence $e, \ldots, e'$, which we append to our global ordering of the edges of G incident to x. We do this for all blocks associated with f in an arbitrary order. When the procedure encounters the first x-edge e_i of a block B_i (recall that such an edge belongs to H and is incident to x), it appends e_i to the global ordering of x and stores the last encountered x-edge of B_i as e_i. Whenever it encounters another x-edge e' of B_i, it appends all edges between e_i and e', excluding e_i and including e', to the global ordering of the edges incident to x, then updates the last encountered x-edge of B_i to e'. When the procedure encounters the last x-edge of a block, it also inserts all remaining edges of $\mathcal{B}_i$ between the last encountered x-edge and the first x-edge of B_i. As the x-edges occur in the embedding $\mathcal{B}_i$ of each block B_i in the same order as in $\sigma_{\mathcal{H}}(x)$, each edge is inserted exactly once into the cyclic ordering. Considering the output sequence as a cyclic sequence, we have found a cyclic ordering of all edges incident to x in G. Clearly, the running time of the procedure is proportional to the number of edges incident to x in G. In addition, the ordering is such that its restriction to H yields $\sigma_{\mathcal{H}}(x)$, no two blocks alternate, and the ordering of the edges of each incident block B_i are compatible with $\mathcal{B}_i$. Finally, also the blocks that do not have an x-edge are embedded into the correct face since this face exists and thus is uniquely determined. The previously removed blocks containing at most one vertex of H—the cutvertex x—can be embedded into arbitrary faces incident to their cutvertices

in reverse order of removal. Clearly, the total running time of this procedure is linear. The following theorem summarizes our results.

THEOREM 4.15. *Let* $(G, H, \mathcal{H})$ *be a* PEG. *There is a linear-time algorithm that either finds an embedding extension* $\mathcal{G}$ *of* $\mathcal{H}$ *or concludes that such an embedding does not exist.*

5. APPLICATIONS AND EXTENSIONS

In this section, we discuss several extensions of the problem PARTIALLY EMBEDDED PLANARITY. Additionally, we show that PEP has some connections to the problem of finding a *simultaneous embedding with fixed edges* of a pair of graphs. In particular, the results of this work can be used to solve this problem for a restricted class of inputs.

Problem extensions. Several generalizations of the PARTIALLY EMBEDDED PLANARITY problem naturally arise. In all of the following generalizations (denoted by G1 through G4), the input is still a PEG $(G, H, \mathcal{H})$. For the first two generalizations, we readily conclude that they are NP-complete since they contain as special cases CROSSING NUMBER and MAXIMUM PLANAR SUBGRAPH, respectively: (G1) deciding if $\mathcal{H}$ can be extended to a drawing of G with at most k crossings and (G2) deciding if at least k edges of $E(G)\backslash E(H)$ can be added to $\mathcal{H}$ while preserving planarity.

The following two additional problems generalize PEP in different directions: (G3) deciding whether G has a planar embedding $\mathcal{G}$ in which at least k edges of H are embedded as in $\mathcal{H}$ and (G4) deciding whether there is a set $F \subseteq E(H)$ of at most k edges such that $(G\backslash F, H\backslash F, \mathcal{H}\backslash F)$ is a planar PEG. We show that problems G3 and G4, called MINIMUM REROUTING PARTIALLY EMBEDDED PLANARITY and MAXIMUM PRESERVED PARTIALLY EMBEDDED PLANARITY, respectively, are NP-hard.

THEOREM 5.1. MINIMUM REROUTING PARTIALLY EMBEDDED PLANARITY *and* MAXIMUM PRESERVED PARTIALLY EMBEDDED PLANARITY *are NP-hard.*

PROOF. The proof is by reduction from STEINERTREE in planar graphs, which is known to be NP-hard [Garey and Johnson 1977]. The problem STEINERTREE in planar graphs takes as an input a planar graph $G = (V, E)$, a set $T \subset V$ of *terminals*, and an integer k and asks whether a tree $T^* = (V^*, E^*)$ exists such that (1) $V^* \subseteq V$, (2) $E^* \subseteq E$, (3) $T \subseteq V^*$, and (4) $|E^*| \leq k$.

We show how to construct an equivalent instance $(G', H, \mathcal{H}, k')$ of MAXIMUM PRESERVED PARTIALLY EMBEDDED PLANARITY, given an instance $(G = (V, E), T, k)$ of STEINERTREE in planar graphs. First, choose an embedding Γ of G and let H be the dual of Γ, with embedding $\mathcal{H}$. For each terminal $t \in T$, we add a new vertex v_t to H and prescribe it inside the face that is dual to t. This completes the construction of H and $\mathcal{H}$. Graph G' has the same vertex set as H, and its edge set is $E(H) \cup S$, where S is the edge set of any connected planar graph G_S spanning the vertices v_t. Finally, we set $k' = k$.

Now consider the problem of finding a set F of k edges of H such that $(G'\backslash F, H\backslash F, \mathcal{H}\backslash F)$ is a planar PEG. Clearly, G_S can be drawn in a planar way if and only if we choose F in such a way that all vertices v_t lie in the same face of $\mathcal{H}\backslash F$. This is equivalent to the property that the set $F^\star$ of edges dual to F is a Steiner tree in G with terminal set T. Hence, $(G', H, \mathcal{H}, k')$ is a positive instance of MAXIMUM PRESERVED PARTIALLY EMBEDDED PLANARITY if and only if (G, T, k) is a positive instance of STEINERTREE. This shows that MAXIMUM PRESERVED PARTIALLY EMBEDDED PLANARITY is NP-hard.

The reduction from an instance (G, T, k) of STEINERTREE in planar graphs to an instance $(G', H, \mathcal{H}, k')$ of MINIMUM REROUTING PARTIALLY EMBEDDED PLANARITY is analogous to the one for MAXIMUM PRESERVED PARTIALLY EMBEDDED PLANARITY. In particular, G', H, and $\mathcal{H}$ are constructed in exactly the same way; however, in such a case, we have

$k' = |E| - k$. Then, it is sufficient to observe that (i) graph G_S can be drawn in a planar way if and only if a set F of edges can be deleted in such a way that all vertices v_t lie in the same face of $\mathcal{H}\backslash F$; (ii) the set $F^\star$ of edges dual to F is a Steiner tree in G with terminal set T; and (iii) since G_S is a connected component of G', the edges of F can be reinserted without crossings into the drawing—that is, to have all vertices v_t lie in the same face of $\mathcal{H}$, it is sufficient to reroute (instead of delete) the edges in F. This shows that Minimum Rerouting Partially Embedded Planarity is NP-hard. □

In the case of Maximum Preserved Partially Embedded Planarity, we can even make H connected as follows. We connect each vertex v_t to an arbitrary vertex of its prescribed face, and we let G_S be a star graph on the vertices v_t. Thus, Maximum Preserved Partially Embedded Planarity is NP-hard even if the prescribed graph H is connected. However, this strategy does not work for Minimum Rerouting Partially Embedded Planarity, as the reduction for this problem relies on the property that every edge of each face can be removed and reinserted after drawing G_S. This is not the case if H is connected. We leave open the question whether Minimum Rerouting Partially Embedded Planarity is NP-hard if the graph H with prescribed embedding is connected.

Application to simultaneous embedding with fixed edges. The results presented in this work can be used to solve special cases of the problem simultaneous embedding with fixed edges. A *simultaneous embedding with fixed edges* (in the following called Sefe, for short) of a pair $G_1 = (V, E_1), G_2 = (V, E_2)$ of graphs on the same vertex set is a pair (Γ_1, Γ_2) of drawings such that (i) Γ_i is a planar drawing of G_i, for each $i = 1, 2$; (ii) each vertex $v \in V$ is drawn on the same point in Γ_1 and in Γ_2; and (iii) each edge $(u, v) \in E_1 \cap E_2$ is represented by the same curve in Γ_1 and in Γ_2. The problem can also be generalized to simultaneous embedding of more than two graphs.

The Sefe problem is a well-studied problem in graph drawing. A lot of research has been devoted to find pairs of graph classes that always admit a Sefe and to determine how many bends are necessary for constructing a Sefe of pairs of graphs that admit one (e.g., see Angelini et al. [2012b], Di Giacomo and Liotta [2007], Erten and Kobourov [2005], Fowler et al. [2011], and Frati [2006]). Additionally, a lot of work is concerned with the algorithmic aspects of the Sefe problem. In particular, it is known that Sefe is NP-hard for two geometric graphs, where edges are restricted to be straight-line segments [Estrella-Balderrama et al. 2007] and that Sefe is NP-hard for three (or more) graphs [Gassner et al. 2006]. Polynomial-time algorithms have been designed for deciding the existence of a Sefe of two graphs, if some further assumptions are made on the input, such as if the intersection of the two input graphs is biconnected [Angelini et al. 2012a; Haeupler et al. 2013], if the input graphs are biconnected and the common graph is connected [Bläsius and Rutter 2013b], and if the connected components of the common graph are biconnected or have low degree [Bläsius et al. 2013a; Bläsius and Rutter 2013a; Schaefer 2013]. Refer to Bläsius et al. [2013b] for a comprehensive survey on this topic. Despite a large amount of research, the complexity status of the Sefe problem for two graphs remains open.

The results presented in this work allow us to solve in linear time an interesting case of the Sefe problem. Namely, Jünger and Schulz [2009] showed that two graphs $G_1 = (V, E_1)$ and $G_2 = (V, E_2)$ admit a Sefe if and only if they admit planar embeddings $\mathcal{E}_1$ and $\mathcal{E}_2$, respectively, that coincide on the intersection graph. This result, together with the results we presented on the Pep problem, implies the following theorem.

Theorem 5.2. *Let G_1 and G_2 be two graphs with the same n vertices, and let $\mathcal{G}_{1\cap 2}$ be a planar embedding of their intersection graph $G_{1\cap 2} := G_1 \cap G_2$. There exists a linear-time algorithm to decide whether G_1 and G_2 admit a* Sefe *in which the embedding of $G_{1\cap 2}$ coincides with $\mathcal{G}_{1\cap 2}$.*

PROOF. By Jünger and Schulz [2009], G_1 and G_2 admit a SEFE in which the embedding of $G_{1\cap2}$ coincides with $\mathcal{G}_{1\cap2}$ if and only if each of G_1 and G_2 admits a planar embedding that coincides with $\mathcal{G}_{1\cap2}$ when restricted to the vertices and edges of $G_{1\cap2}$. In other words, G_1 and G_2 admit a SEFE in which the embedding of $G_{1\cap2}$ coincides with $\mathcal{G}_{1\cap2}$ if and only if $(G_1, G_{1\cap2}, \mathcal{G}_{1\cap2})$ and $(G_2, G_{1\cap2}, \mathcal{G}_{1\cap2})$ are both YES-instances of PEP. □

Theorem 5.2 implies that deciding whether two graphs have a SEFE is a linear-time solvable problem if one of the graphs has a fixed embedding, such as if one of the two graphs is triconnected.

6. CONCLUDING REMARKS

In this article, we showed that PARTIALLY EMBEDDED PLANARITY (PEP) is a linear-time solvable problem. Problem PEP asks whether a partially embedded graph (PEG) is planar—that is, whether a planar drawing $\mathcal{H}$ of a subgraph H of a planar graph G can be extended to a planar drawing of G. To derive our linear-time algorithm, we first presented a combinatorial characterization of planar PEGs in terms of conditions on the structure of the triconnected, biconnected, and connected components of the input graph. This characterization immediately implies a polynomial-time algorithm for testing the planarity of a given PEG. The second part of the article was devoted to a careful implementation of the algorithm following from the characterization, resulting in an algorithm for PEP with optimal linear running time. Although edge compatibility exhibits a very local behavior and hence is not too difficult to enforce in linear time, numerous steps are necessary to handle cycle compatibility in linear time as well. In addition, we showed that our testing algorithm can be made constructive—that is, it can be implemented so that it finds an embedding extension for the input PEG, if one exists. Altogether, from a purely algorithmic point of view, this completely settles the problem PEP.

Further, we considered several generalizations of PEP and proved that they are NP-hard. Additionally, we showed that PEP exhibits strong connections with another well-known graph drawing problem: the SEFE problem. The results in this work immediately imply a linear-time algorithm for solving SEFE when the embedding of the intersection graph is fixed, which holds, for example, if one of the input graphs is triconnected.

In a subsequent paper, Jelínek et al. [2013] characterize the planar PEGs via forbidden substructures in the spirit of Kuratowski's theorem. In conjunction with the results presented in this work, this gives an efficient algorithm that, for a given PEG, either finds a planar extension or decides that the PEG is nonplanar and extracts an obstruction.

Open problems. The problem PEP asks for determining the extendability of planar combinatorial embeddings or, equivalently, of topological drawings of planar graphs. An obvious research direction is to consider the complexity of the extendability question for other drawing styles. It is known that completing partial straight-line drawings is NP-hard [Patrignani 2006], and it seems that the NP-hardness proof generalizes easily to poly-line drawings that admit a fixed number of bends per edge. Subsequent to the conference version of this article [Angelini et al. 2010], the problem of extending partial representations has been considered for function graphs and permutation graphs [Klavík et al. 2012a], for subclasses of chordal graphs [Klavík et al. 2012b], for interval graphs [Klavík et al. 2011], and for proper and unit interval graphs [Klavík et al. 2012c]. However, it might be interesting to consider the problems of extending, for example, orthogonal drawings [Tamassia 1987] or Manhattan-geodesic drawings [Katz et al. 2010].

A different direction for generalizing PEP would be to relax the strict condition to have a fixed embedding $\mathcal{H}$ for a subgraph H of the entire planar graph G. Gutwenger et al. [2008] consider the problem of testing the planarity of a graph with the further constraint that every vertex has an associated PQ-tree representing the possible rotations that are allowed for that vertex. The common generalization of PEP and this problem assumes that only a subgraph is constrained by such PQ-trees and the remaining edges can be inserted arbitrarily. Is it possible to decide planarity of a partially PQ-constrained graph G in polynomial time? A positive answer to the previous question has been provided in Bläsius and Rutter [2013b] for the case in which G is biconnected.

ACKNOWLEDGMENTS

This work began at the BICI Workshop on Graph Drawing, held in Bertinoro, Italy, in March 2009 and was carried out while the authors were at the Department of Applied Mathematics, Charles University, Prague.

REFERENCES

P. Angelini, G. Di Battista, F. Frati, V. Jelínek, J. Kratochvíl, M. Patrignani, and I. Rutter. 2010. Testing planarity of partially embedded graphs. In *Proceedings of SODA'10*. 202–221.

P. Angelini, G. Di Battista, F. Frati, M. Patrignani, and I. Rutter. 2012a. Testing the simultaneous embeddability of two graphs whose intersection is a biconnected or a connected graph. *Journal of Discrete Algorithms* 14, 150–172.

P. Angelini, M. Geyer, M. Kaufmann, and D. Neuwirth. 2012b. On a tree and a path with no geometric simultaneous embedding. *Journal of Graph Algorithms and Algorithms* 16, 1, 37–83.

P. Bertolazzi, G. Di Battista, and W. Didimo. 2000. Computing orthogonal drawings with the minimum number of bends. *IEEE Transactions on Computers* 49, 8, 826–840.

T. Bläsius, A. Karrer, and I. Rutter. 2013a. Simultaneous embedding: Edge orderings, relative positions, cutvertices. In *Graph Drawing*. Lecture Notes in Computer Science, Vol. 8242. Springer, 220–231.

T. Bläsius, S. G. Kobourov, and I. Rutter. 2013b. Simultaneous embedding of planar graphs. In *Handbook of Graph Drawing and Visualization*, R. Tamassia (Ed.). CRC Press.

T. Bläsius and I. Rutter. 2013a. Disconnectivity and relative positions in simultaneous embeddings. In *Graph Drawing*. Lecture Notes in Computer Science, Vol. 7704. Springer, 31–42.

T. Bläsius and I. Rutter. 2013b. Simultaneous pq-ordering with applications to constrained embedding problems. In *Proceedings of SODA'13*. 1030–1043.

J. M. Boyer and W. J. Myrvold. 2004. On the cutting edge: Simplified $O(n)$ planarity by edge addition. *Journal of Graph Algorithms and Applications* 8, 3, 241–273.

H. de Fraysseix, P. O. de Mendez, and P. Rosenstiehl. 2006. Trémaux trees and planarity. *International Journal on Foundations of Computer Science* 17, 1017–1030.

G. Demoucron, Y. Malgrange, and R. Pertuiset. 1964. Reconnaissance et construction de représentations planaires topologiques. *Rev. Francaise Recherche Opérationelle* 8, 33–34.

G. Di Battista and R. Tamassia. 1996. On-line planarity testing. *SIAM Journal on Computing* 25, 956–997.

E. Di Giacomo and G. Liotta. 2007. Simultaneous embedding of outerplanar graphs, paths, and cycles. *International Journal of Computational Geometry and Applications* 17, 2, 139–160.

C. Dornheim. 2002. Planar graphs with topological constraints. *Journal of Graph Algorithms and Applications* 6, 1, 27–66.

C. Erten and S. G. Kobourov. 2005. Simultaneous embedding of planar graphs with few bends. *Journal of Graph Algorithms and Applications* 9, 3, 347–364.

A. Estrella-Balderrama, E. Gassner, M. Jünger, M. Percan, M. Schaefer, and M. Schulz. 2007. Simultaneous geometric graph embeddings. In *Graph Drawing*. Lecture Notes in Computer Science, Vol. 4875. Springer, 280–290.

J. Fiala. 2003. NP-completeness of the edge precoloring extension problem on bipartite graphs. *Journal of Graph Theory* 43, 2, 156–160.

J. Fowler, M. Jünger, S. G. Kobourov, and M. Schulz. 2011. Characterizations of restricted pairs of planar graphs allowing simultaneous embedding with fixed edges. *Computational Geometry* 44, 8, 385–398.

F. Frati. 2006. Embedding graphs simultaneously with fixed edges. In *Graph Drawing*. Lecture Notes in Computer Science, Vol. 4372. Springer, 108–113.

M. Garey and D. Johnson. 1977. The rectilinear Steiner tree problem is NP-complete. *SIAM Journal on Applied Mathematics* 32, 4, 826–834.

E. Gassner, M. Jünger, M. Percan, M. Schaefer, and M. Schulz. 2006. Simultaneous graph embeddings with fixed edges. In *Graph-Theoretic Concepts in Computer Science*. Lecture Notes in Computer Science, Vol. 4271. Springer, 325–335.

M. Grötschel, L. Lovász, and A. Schrijver. 1988. Stable sets in graphs. In *Geometric Algorithms and Combinatorial Optimization*. Springer, 273–303.

C. Gutwenger, K. Klein, and P. Mutzel. 2008. Planarity testing and optimal edge insertion with embedding constraints. *Journal of Graph Algorithms and Applications* 12, 1, 73–95.

C. Gutwenger and P. Mutzel. 2000. A linear time implementation of SPQR-trees. In *Graph Drawing*. Lecture Notes in Computer Science, Vol. 1984. Springer, 77–90.

B. Haeupler, K. R. Jampani, and A. Lubiw. 2013. Testing simultaneous planarity when the common graph is 2-connected. *Journal of Graph Algorithms and Applications* 17, 3, 147–171.

J. Hopcroft and R. E. Tarjan. 1974. Efficient planarity testing. *Journal of the ACM* 21, 4, 549–568.

V. Jelínek, J. Kratochvíl, and I. Rutter. 2013. A Kuratowski-type theorem for planarity of partially embedded graphs. *Computational Geometry: Theory and Applications* 46, 4, 466–492. DOI:http://dx.doi.org/10.1016/j.comgeo.2012.07.005

M. Jünger and M. Schulz. 2009. Intersection graphs in simultaneous embedding with fixed edges. *Journal of Graph Algorithms and Applications* 13, 2, 205–218.

M. Juvan and B. Mohar. 2005. 2-restricted extensions of partial embeddings of graphs. *European Journal of Combinatorics* 26, 3–4, 339–375.

B. Katz, M. Krug, I. Rutter, and A. Wolff. 2010. Manhattan-geodesic embedding of planar graphs. In *Graph Drawing*. Lecture Notes in Computer Science, Vol. 5849. Springer, 207–218.

P. Klavík, J. Kratochvíl, T. Krawczyk, and B. Walczak. 2012a. Extending partial representations of function graphs and permutation graphs. In *Algorithms—ESA 2012*. Lecture Notes in Computer Science, Vol. 7501. Springer, 671–682.

P. Klavík, J. Kratochvíl, Y. Otachi, I. Rutter, T. Saitoh, M. Saumell, and T. Vyskočil. 2012c. Extending partial representations of proper and unit interval graphs. *CoRR* abs/1207.6960.

P. Klavík, J. Kratochvíl, Y. Otachi, and T. Saitoh. 2012b. Extending partial representations of subclasses of chordal graphs. In *Algorithms and Computation*. Lecture Notes in Computer Science, Vol. 7676. Springer, 444–454.

P. Klavík, J. Kratochvíl, and T. Vyskočil. 2011. Extending partial representations of interval graphs. In *Theory and Applications of Models of Computation*. Lecture Notes in Computer Science, Vol. 6648. Springer, 276–285.

L. Kowalik and M. Kurowski. 2003. Short path queries in planar graphs in constant time. In *Proceedings of STOC'03*. 143–148.

J. Kratochvíl and A. Sebo. 1997. Coloring precolored perfect graphs. *Journal of Graph Theory* 25, 207–215.

K. Kuratowski. 1930. Sur le problème des courbes gauches en topologie. *Fundamenta Mathematicae* 15, 271–283.

B. Mohar. 1999. A linear time algorithm for embedding graphs in an arbitrary surface. *SIAM Journal on Discrete Mathematics* 12, 1, 6–26.

P. Mutzel. 2003. The SPQR-tree data structure in graph drawing. In *Automata, Languages and Programming*. Lecture Notes in Computer Science, Vol. 2719. Springer, 35–46.

M. Patrignani. 2006. On extending a partial straight-line drawing. *International Journal of Foundations of Computer Science* 17, 5, 1061–1069.

J. A. La Poutré. 1994. Alpha-algorithms for incremental planarity testing. In *Proceedings of STOC'94*. 706–715.

M. Schaefer. 2013. Toward a theory of planarity: Hanani-tutte and planarity variants. *Journal of Graph Algorithms and Applications* 17, 4, 367–440.

R. Tamassia. 1987. On embedding a graph in the grid with the minimum number of bends. *SIAM Journal on Computing* 16, 3, 421–444.

R. Tamassia. 1996. On-line planar graph embedding. *Journal of Algorithms* 21, 2, 201–239.

R. Tamassia. 1998. Constraints in graph drawing algorithms. *Constraints* 3, 1, 87–120.

R. Tamassia, G. Di Battista, and C. Batini. 1988. Automatic graph drawing and readability of diagrams. *IEEE Transactions on Systems, Man, and Cybernetics* 18, 1, 61–79.

R. E. Tarjan. 1972. Depth first search and linear graph algorithms. *SIAM Journal on Computing* 2, 146–160.

J. Westbrook. 1992. Fast incremental planarity testing. In *Automata, Languages and Programming*. Lecture Notes in Computer Science, Vol. 623. Springer, 342–353.

Received March 2013; revised April 2014; accepted April 2014