Journal of Graph Algorithms and Applications
http://jgaa.info/ vol. 17, no. 3, pp. 147–171 (2013)
DOI: 10.7155/jgaa.00289

Testing Simultaneous Planarity when the Common Graph is 2-Connected

Bernhard Haeupler[1] *Krishnam Raju Jampani*[2] *Anna Lubiw*[3]

[1]Computer Science and Artificial Intelligence Laboratory,
Massachusetts Institute of Technology, USA.
[2]Department of Mathematics and Statistics,
University of Guelph, Canada.
[3]David R. Cheriton School of Computer Science,
University of Waterloo, Canada.

Abstract

Two planar graphs G_1 and G_2 sharing some vertices and edges are *simultaneously planar* if they have planar drawings such that a shared vertex [edge] is represented by the same point [curve] in both drawings. It is an open problem whether simultaneous planarity can be tested efficiently. We give a linear-time algorithm to test simultaneous planarity when the shared graph is 2-connected. Our algorithm extends to the case of k planar graphs where each vertex [edge] is either common to all graphs or belongs to exactly one of them, and the common subgraph is 2-connected.

Keywords: simultaneous embedding, planar graph, PQ tree, graph drawing, planarity test

Submitted:	Reviewed:	Revised:	Accepted:	Final:
December 2011	November 2012	December 2012	February 2012	February 2012

Published:
March 2012

Article type:	Communicated by:
Regular paper	J. S. B. Mitchell

E-mail addresses: haeupler@mit.edu (Bernhard Haeupler) rjampani@uoguelph.ca (Krishnam Raju Jampani) alubiw@uwaterloo.ca (Anna Lubiw)

1 Introduction

Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be two graphs sharing some vertices and edges. The simultaneous planar embedding problem asks whether there exist planar embeddings for G_1 and G_2 such that, in the two embeddings, each vertex $v \in V_1 \cap V_2$ is mapped to the same point and each edge $e \in E_1 \cap E_2$ is mapped to the same curve. We show that this problem can be solved efficiently when the *common* graph $(V_1 \cap V_2, E_1 \cap E_2)$ is 2-connected.

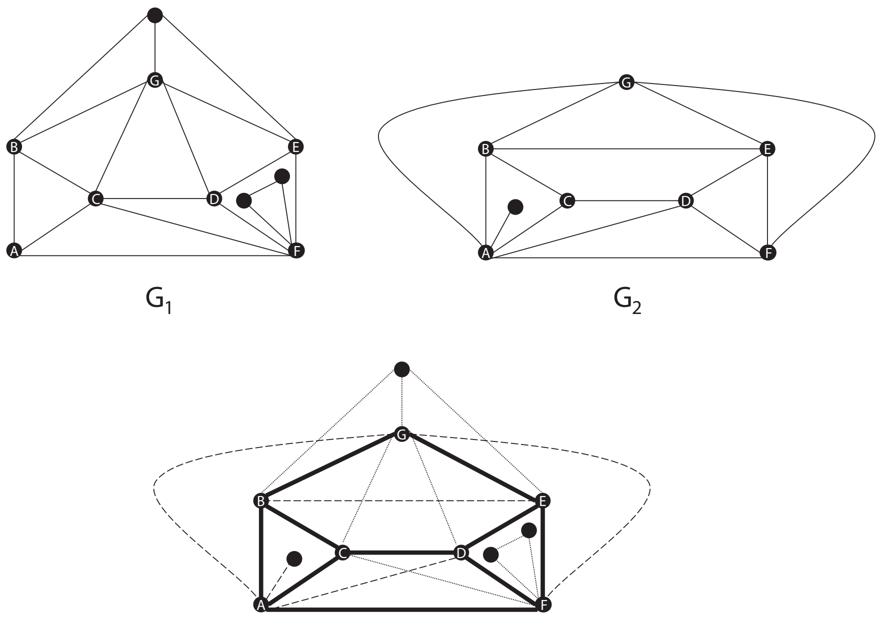

Figure 1: Two planar graphs G_1 and G_2 and their simultaneous planar embedding. The common graph (drawn in bold) is 2-connected and the only edge crossings in the simultaneous embedding occur between a private edge of G_1 and a private edge of G_2.

The study of planar graphs has a long history and has generated many deep results [31–33]. There is hope that some of the structure of planarity may carry over to simultaneous planarity. A possible analogy is with matroids, where optimization results generalize from one matroid to the intersection of two matroids [12]. On a more practical note, simultaneous planar embeddings are valuable for visualization purposes when two related graphs need to be displayed. For example, the two graphs may represent different relationships on the same node set, or they may be the "before" and "after" versions of a graph that has changed over time.

Over the last few years there has been a lot of work on simultaneous planar

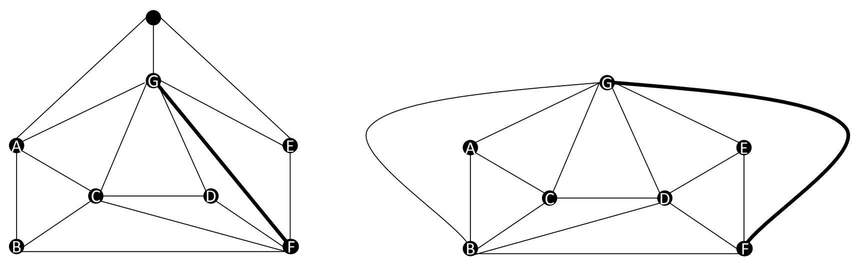

Figure 2: Two planar graphs that do not allow for a simultaneous planar embedding. Note that the two planar embeddings drawn here are not simultaneous planar because the common edge $\{F, G\}$ (drawn in bold) is mapped to two different curves.

embeddings [1, 2, 14, 15, 18–21, 29], often under the name "simultaneous embeddings with fixed edges". We mention a few results here and give a more detailed description in the Background section below. A major open question is whether simultaneous planarity of two graphs can be tested in polynomial time. The problem seems to be right on the feasibility boundary. The problem is NP-complete for three graphs [21] and the version where the planar drawings are required to be straight-line is already NP-hard for two graphs and only known to lie in PSPACE [16]. On the other hand, several classes of (pairs of) graphs are known to always have simultaneous planar embeddings [14, 15, 19, 20, 29] and there are efficient algorithms to test simultaneous planarity for some very restricted graph-classes: biconnected outerplanar graphs [19], the case where one graph has at most one cycle [18] or the case where the common graph is a star [2]. Simultaneous planarity generalizes the problem of whether a planar embedding of a subgraph can be extended to a planar embedding of the whole graph, for which a linear time algorithm was recently given by Angelini et al. [1]. In particular, this problem is equivalent to testing simultaneous planarity between a planar graph and a 3-connected graph.

In this paper we give a linear time algorithm to test simultaneous planarity of any two graphs whose shared subgraph is 2-connected. Our algorithm builds on the planarity testing algorithm of Haeupler and Tarjan [23], which in turn unifies the planarity testing algorithms of Lempel-Even-Cederbaum [30], Shih-Hsu [36] and Boyer-Myrvold [9]. We show how to extend our algorithm to the case of k graphs where each vertex [edge] is either common to all graphs or belongs to exactly one of them, and the common graph is 2-connected. Although we concentrate on the decision version of the problem, we also initiate the quest for a nice simultaneous planar drawing. We show that two simultaneous planar graphs with a connected common graph have a simultaneous planar drawing in which one graph is straight-line planar, and each edge of the other graph is drawn as a polygonal line with at most $|V_1 \cap V_2|$ bends.

Independently (and at the same time as our conference version [22]), Angelini et al. [2] gave an efficient algorithm to test simultaneous planarity of any two graphs whose shared subgraph is 2-connected. Their algorithm is based on SPQR-trees, takes linear time (in the final version [3]), and is restricted to the case where the two graphs have the same vertex set. Recently Bläsius and Rutter [6] have generalized our idea of finding "consistent orderings" between PQ trees that share some common leaves (see sections 2.1 and 4), and shown how to test simultaneous planarity of two biconnected graphs whose shared subgraph is connected. This is a substantial strengthening of our result. Bläsius and Rutter [5] also solve the case where the shared graph is disconnected but the embedding of each connected component is fixed.

Our paper is organized as follows: Section 1.1 gives more background and related work. In Section 2 we review and develop some techniques for PQ-trees, which are needed for our algorithm, and in Section 3 we review the Haeupler-Tarjan planarity testing algorithm. Section 4 contains our simultaneous planarity testing algorithm, including the extension to k graphs, and the result on drawing two simultaneous planar graphs.

1.1 Background

Several versions of simultaneous planarity have received much attention in recent years. Brass et al. [10] introduced the concept of *simultaneous geometric embeddings* of a pair of graphs—these are planar straight-line drawings such that any common vertex is represented by the same point. Note that a common edge will necessarily be represented by the same line segment. It is NP-hard to test if two graphs have simultaneous geometric embeddings [16]. For other work on simultaneous geometric embeddings see [4] and its references.

The generalization to planar drawings where edges are not necessarily drawn as straight line segments, but any common edge must be represented by the same curve was introduced by Erten and Kobourov [15] and called *simultaneous embedding with consistent edges.* Most other papers follow the conference version of Erten and Kobourov's paper and use the term *simultaneous embedding with fixed edges (SEFE).* In our paper we use the more self-explanatory term "simultaneous planar embeddings." A further justification for this nomenclature is that there are combinatorial conditions on a pair of planar embeddings that are equivalent to simultaneous planarity. Specifically, Jünger and Schulz give a characterization in terms of "compatible embeddings" [Theorem 4 in [29]]. Specialized to the case where the common graph is connected, their result says that two planar embeddings are simultaneously planar if and only if the cyclic orderings of common edges around common vertices are the same in both embeddings.

Several papers [14, 15, 20] show that pairs of graphs from certain restricted classes always have simultaneous planar embeddings, the most general result being that any planar graph has a simultaneous planar embedding with any tree [20]. On the other hand, there is an example of two outerplanar graphs that have no simultaneous planar embedding [20]. The graphs that have simultaneous planar embeddings when paired with any other planar graph have been characterized [19]. In addition, Jünger and Schulz [29] characterize the common graphs that permit simultaneous planar embeddings no matter what pairs of planar graphs they occur in.

There are efficient algorithms to test simultaneous planarity for pairs of biconnected outerplanar graphs [19], and for pairs consisting of a planar graph and a graph that is either 3-connected [1] or has at most one cycle [18]. Testing simultaneous planarity when the common graph is a tree was shown to be equivalent to a book-embedding problem which together with [24] implies that testing simultaneous planarity when the common graph is a star can be done in linear time [2].

There is another, even weaker form of simultaneous planarity, where common vertices must be represented by common points, but the planar drawings are otherwise completely independent, with edges drawn as Jordan curves. Any set of planar graphs can be represented this way by virtue of the result that a planar graph can be drawn with any fixed vertex locations [34].

The idea of "simultaneous graph representations" has also been applied to intersection representations. For the case of interval graphs, the problem is to find representations for two graphs as intersection graphs of intervals in the real line, with the property that a common vertex is represented by the same interval in both representations. This can be done in polynomial time [26]. There are also polynomial time algorithms to find simultaneous representations of pairs of chordal, comparability and permutation graphs [27].

2 PQ-trees

Many planarity testing algorithms in the literature [9, 23, 30, 36] use PQ-trees (or a variation) to obtain a linear-time implementation. PQ-trees were discovered by Booth and Lueker [8] and are used, not only for planarity testing, but for many other applications like recognizing interval graphs or testing matrices for the consecutive-ones property. We first review PQ-trees and then in Subsection 2.1 we show how to manipulate pairs of PQ-trees.

A PQ-tree represents the permutations of a set of elements satisfying a family of constraints. Each constraint specifies that a certain subset of elements must appear consecutively in any permutation. PQ-trees are rooted trees with internal nodes being labeled 'P' or 'Q', and are drawn using a circle or a rectangle (or double circles in [25]), respectively. The leaves of a PQ-tree correspond to the elements whose orders are represented. PQ-trees are equivalent under arbitrary reorderings of the children of a P-node and reversals of the order of children of a Q-node. We consider a node with two children to be a Q-node. A leaf-order of a PQ-tree is the order in which its leaves are visited in an in-order traversal of the tree. The set of permutations represented by a PQ-tree is the set of leaf-orders of equivalent PQ-trees. Given a PQ-tree T on a set U of elements, adding a consecutivity constraint on a set $S \subseteq U$, *reduces* T to a PQ-tree T', such that the leaf-orders of T' are precisely the leaf-orders of T in which the elements of S appear consecutively. Booth and Lueker [8] gave an efficient implementation of PQ-trees that supports this operation in amortized $O(|S|)$ time. Their implementation furthermore allows an efficient complement-reduction, i.e., adding the constraint on a set S, given the set $U \setminus S$, in amortized $O(|U \setminus S|)$ time [23].

Although PQ-trees were invented to represent linear orders, they can be reinterpreted to represent circular orders as well [23]: Given a PQ-tree we imagine that there is a new special leaf s attached as the "parent" of the root. A circular leaf order of the augmented tree is a circular order that begins at the special leaf, followed by a linear order of the remaining PQ-tree and ending at the special leaf. Again a PQ-tree represents all circular leaf-orders of equivalent PQ-trees. It is easy to see that a consecutivity constraint on such a set of circular orders directly corresponds to a consecutivity constraint on the original set of linear leaf-orders and a reduction on the circular orders corresponds to a standard or complement reduction. Note that using PQ-trees for circular orders requires solely this different view on PQ-trees but does not need any change in their implementation. As such, it turns out that PC-trees introduced in [25] are the exact same data structure as PQ-trees albeit with this circular interpretation.

2.1 Intersection and projection of PQ-trees

In this section we develop simple techniques to obtain consistent orders from two PQ-trees. More precisely when two PQ-trees share some but not necessarily all leaves, we want to find a permutation represented by each of them with a consistent ordering on the shared leaves.

Note that the set of such consistent orderings is not representable by a PQ-tree. For example, consider the ground set $\{1, 2, 3, 4\}$ and the constraint that $\{2, 3\}$ must be consecutive among $\{1, 2, 3\}$. Element 4 may appear anywhere, and there is no way to capture this with a PQ-tree. This is why we restrict our goal to finding one permutation of each PQ-tree such that the two permutations are consistent on the shared leaves.

The idea is to first *project* both PQ-trees to the common elements, intersect the resulting PQ-trees, pick one remaining order and finally "lift" this order back. We now describe the individual steps of this process in more detail.

The *projection* of a PQ-tree on a subset of its leaves S is a PQ-tree obtained by deleting all elements not in S and simplifying the resulting tree. Simplifying a tree means that we (recursively) delete any internal node that has no children, and delete any node that has a single child by making the child's grandparent become its parent. This can easily be implemented to run in linear time.

Given two PQ-trees on the same set of leaves (elements) we define their *intersection* to be the PQ-tree T that represents exactly all orders that are leaf-orders in both trees. This intersection can be computed in polynomial time as follows.

1. Initialize T to be the first PQ-tree.
2. For each P-node in the second PQ-tree, reduce T by adding a consecutivity constraint on all the P-node's descendant leaves.

3. For each Q-node in the second tree, and for each pair of adjacent children of it, reduce T by adding a consecutivity constraint on all the descendant leaves of the two children.

Using the efficient PQ-tree implementation, computing such an intersection can be sped up to amortized linear time in the size of the two PQ-trees. This is a relatively straight-forward extension of the PQ-tree reduction and is described in detail in Booth's thesis [7].

These two operations are enough to achieve our goal. Given two PQ-trees T_1 and T_2 defined on different element (leaf) sets, we define S to be the set of common elements. Now we first construct the projections of both PQ-trees on S and then compute their intersection T as described above. Any permutation of S represented by T can now easily be "lifted" back to permutations of T_1 and of T_2 that respect the chosen ordering of S. Furthermore, any two permutations of T_1 and T_2 that are consistent on S can be obtained this way.

We note that techniques to "merge" PQ-trees were also presented by Jünger and Leipert [28] in work on level planarity. Their merge is conceptually and technically different from ours in that the result of their merge is a single PQ-tree whereas we compute two orderings that are consistent on common elements. The set of all these consistent orderings cannot be captured by a PQ-tree.

3 Planarity testing

In this section, we review the algorithm of Haeupler and Tarjan [23] for testing the planarity of a graph. Later, we will extend it to an algorithm for testing simultaneous planarity. We first begin with some basic definitions.

Let $G = (V, E)$ be a connected graph on vertex set $V = \{v_1, \cdots, v_n\}$ and let $\mathcal{O}$ be an ordering of the vertices of V. An edge v_iv_j is an *in-edge* of v_i (in $\mathcal{O}$) if v_j appears before v_i in $\mathcal{O}$, and v_iv_j is an *out-edge* of v_i if v_j appears after v_i in $\mathcal{O}$.

An st-ordering of G is an ordering $\mathcal{O}$ of the vertices of G, such that the first vertex of $\mathcal{O}$ is adjacent to the last vertex of $\mathcal{O}$ and every intermediate vertex has at least one in-edge and at least one out-edge. It is well-known that G has an st-ordering if and only if it is 2-connected. Further, an st-ordering can be computed in linear time [17].

The planarity test of Haeupler and Tarjan embeds vertices (together with their adjacent edges) one at a time. Edges with two, one or no endpoint(s) embedded are called (fully-)embedded, half-embedded and non-embedded respectively. The idea is to keep track of all possible partial planar embeddings at each stage. A partial planar embedding is a planar drawing of all embedded vertices and edges that also contains a straight line for each half-embedded edge starting at its embedded vertex and ending in an adjacent face. An example of such a partial planar embedding can be seen in Figure 3. For the correctness of the algorithm it is crucial that the vertices are added in a leaf-to-root order of a spanning-tree. This guarantees that, at any time, the non-embedded vertices induce a connected subgraph. When extending a partial planar embedding these vertices will therefore end up in a single face of this embedding and it is convenient to think of this face as the outer face. With this as a precursor we can restrict attention to partial planar embeddings in which all half-embedded edges end in the outer face. Now the first crucial observation is that (for the sake of extending this embedding) a partial embedding is completely characterized by describing, for each connected component, the cyclic order of the half-embedded edges around it. Secondly, maintaining one PQ-tree per connected component leads to a natural and efficient way to describe the set of all possible cyclic orders and thus to a concise description of the set of all possible partial planar embeddings. Indeed, the main operation needed when embedding a new vertex is to restrict the set of circular half-embedded edge orders of a connected component to those in which the edges going to the newly embedded vertex are consecutive. This is neccessary as otherwise embedding the vertex leads to a non-extendable embedding with half-embedded edges enclosed in two or more faces. It also directly corresponds to the reduction operation supported by the PQ-tree data structure.

Before describing in more detail how to update the PQ-trees when a vertex is added, we first discuss the choice of the specific embedding order we use. In general using either a leaf-to-root order of a depth-first spanning tree or an st-order leads to particularly simple implementations that run in linear-time. Indeed these two orders are essentially the only two orders in which the algorithm runs in linear-time using the standard PQ-tree implementation. Our simultaneous planarity algorithm will use a mixture of the two orders: We first add the vertices that are contained in only one of the graphs by a leaf-to-root order of a depth-first search tree and then add the common vertices using an st-ordering. Note that if all vertices are common, both G_1 and G_2 will be 2-connected and have a common st-order. In this case we only use this st-order. We now give an overview of how the update steps of the planarity test work for each of these orderings.

Leaf-to-root order of a depth-first spanning tree:
Let $v_1, v_2, \cdots, v_n$ be a leaf-to-root order of a depth-first spanning tree of G. Note that at stage i, the vertices $\{v_1, \cdots, v_i\}$ may induce several connected components. We maintain a PQ-tree for each component representing the set of possible circular orderings of its out-edges. Using a depth-first spanning tree, in contrast to an arbitrary spanning tree, has the advantage that we can easily maintain the invariant that the edge to the smallest node greater than i will be the special leaf, which is used to represent circular orderings.

Adding v_{i+1} can lead to merging several components into one. It is easy to see, e.g., in Figure 3 or the left part of Figure 5, that after embedding v_{i+1} the edges of each merging component that go to v_{i+1} have to be consecutive since otherwise a half-embedded edge would be enclosed in the newly formed component. To go to the next stage, we thus first reduce each PQ-tree corresponding to such a merging component by adding a consecutivity constraint that requires the set of out-edges that are incident to v_{i+1} to be consecutive. We then delete

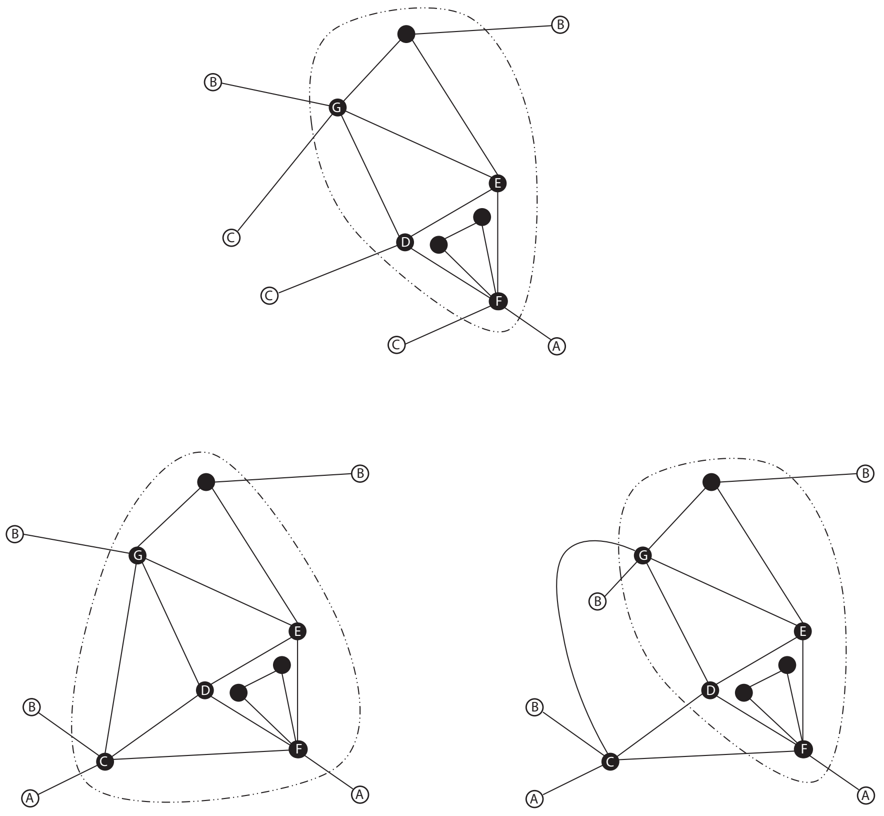

Figure 3: Above is a partial planar embedding of the graph G_1 from Figure 1 with vertices A, B and C being non-embedded. All half-embedded edges (drawn with hollow vertices for their non-embedded endpoints) lie in the outer face and their cyclic order can be observed by following the dotted curve. The half-embedded edges incident with vertex C are consecutive in the ordering, which allows the embedding to be extended by adding vertex C as shown in the lower left figure. Again all half-embedded edges lie in the new outside face. On the lower right, the initial partial embedding is altered by flipping the order of the two half-embedded edges incident to vertex G, as can be observed by following the dotted curve. Now the half-embedded edges incident with vertex C are not consecutive in the ordering, and the attempt to add vertex C results in a half-embedded edge (the one from G to B) enclosed in an inside face. Partial embeddings like this are eliminated by the PQ-reduction of the planarity test.

these edges since they are now fully embedded. By the invariant stated above the special leaf is among these edges. Note that the resulting PQ-tree for a component now represents the set of all possible linear-orders of the out-edges that are not incident to v_{i+1}. Now we construct the PQ-tree for the new merged component including v_{i+1} as follows:

Let v_l be the parent of v_{i+1} in the depth-first spanning tree. The PQ-tree for the new component consists of the edge $v_{i+1}v_l$ as the special leaf and a new P-node as a root and whose children are all the remaining out-edges of v_{i+1} and the roots of the PQ-trees of the reduced components (similar to the picture in Figure 4). Note that by choosing the edge $v_{i+1}v_l$ as the special leaf we again maintain the above mentioned invariant.

It is easy to verify that these operations capture exactly all possible circular orders of half-embedded edges around the new embedded connected component. Note that if the reduction step fails at any stage then the graph must be non-planar. Otherwise the algorithm concludes that the graph is planar.

st-order:
Let $v_1, v_2, \cdots, v_n$ be an st-order of G. This order is characterized by the fact that at any stage $i \in \{1, \cdots, n-1\}$ not just the non-embedded vertices but also the embedded vertices $\{v_1, \cdots, v_i\}$ induce a connected component. This results in the algorithm having to maintain only one PQ-tree T_i to capture all possible circular orderings of out-edges around this one component. Furthermore since v_1v_n is an out-edge at every stage, it can stay as the special leaf of T_i for all i.

The update step for embedding the next vertex is the same as above. At stage 1, the tree T_1 consists of the special leaf v_1v_n and a P-node whose children are all other out-edges of v_1. Now suppose we are at a stage $i \in \{1, \cdots, n-2\}$ and want to go to the next stage. (Since we only maintain the orderings of out-edges of the partially embedded graph at each stage, we do not have to consider stage n). Again it holds that after embedding v_{i+1} the in-edges incident to v_{i+1} have to be consecutive. We call the set of leaves of T_i that correspond to these edges the *black* leaves. To go to the next stage, we thus first reduce T_i so that all black edges appear together. This ensures that either the black edges appear as the (leaf) descendants of a P-node or the (leaf) descendants of a consecutive sequence of child nodes of a Q-node. A non-leaf node in the reduced PQ-tree is said to be black if all its leaf descendants are black edges. We next create a new P-node p_{i+1} and add all the out-edges of v_{i+1} as its children. Now T_{i+1} is constructed from T_i as follows:
Case 1: *T_i contains a black node x that is an ancestor of all the black leaves.* We obtain T_{i+1} from T_i by replacing x and all its descendants with p_{i+1}.
Case 2: *T_i contains a (non-black) Q-node containing a (consecutive) sequence of black children.* We obtain T_{i+1} from T_i by replacing these black children (and their descendants) with p_{i+1}.

This captures again exactly all possible circular orders of half-embedded edges around the newly embedded connected component. Note that as before the graph is non-planar if and only if the reduction step fails at any stage.

4 Simultaneous planarity when the common graph is 2-connected

This section contains our main result, a linear time algorithm for testing simultaneous planarity of two graphs whose shared subgraph is 2-connected.

Let $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ be two planar connected graphs with $|V_1| = n_1$ and $|V_2| = n_2$. Note that $G_1[V_1 \cap V_2]$ need not be the same as $G_2[V_1 \cap V_2]$. Let $G = (V_1 \cap V_2, E_1 \cap E_2)$ be 2-connected and $n = |V_1 \cap V_2|$. Let $v_1, v_2, \ldots, v_n$ be an st-ordering of the vertices of G. We call the edges and vertices of G *common* and all other vertices and edges *private*.

We say two linear or circular orderings of elements with some common elements are *compatible* if the common elements appear in the same relative order in both orderings. Similarly we say two combinatorial planar embeddings of G_1 and G_2, respectively, are *compatible* if for each common vertex the two circular orderings of edges incident to it are compatible.

If G_1 and G_2 have simultaneous planar embeddings, then clearly they have combinatorial planar embeddings that are compatible with each other. The converse also turns out to be true, if the common edges form a connected graph. This can be easily proved as follows. Let $\mathcal{E}_1$ and $\mathcal{E}_2$ be the compatible combinatorial planar embeddings of G_1 and G_2 respectively. Let $\mathcal{E}_p$ be the partial embedding of $\mathcal{E}_1$ [or $\mathcal{E}_2$] obtained by restricting G_1 [resp. G_2] to the common subgraph. (Note that since $\mathcal{E}_1$ and $\mathcal{E}_2$ are compatible, the partial embedding of $\mathcal{E}_1$ restricted to the common subgraph is the same as the partial embedding of $\mathcal{E}_2$ restricted to the common subgraph). Now we can find a planar embedding of the common subgraph that corresponds to $\mathcal{E}_p$ and iteratively extend it to an embedding of $\mathcal{E}_1$ and an embedding of $\mathcal{E}_2$ (see Lemma 2 of Jünger and Schulz [29] for a proof). The two planar embeddings thus obtained are simultaneous planar embeddings and thus it is enough to compute a pair of compatible combinatorial planar embeddings.

We will find compatible combinatorial planar embeddings by adding vertices one by one. At any point we will have two sets of PQ-trees representing the partial planar embeddings of the subgraphs of G_1 and G_2 induced by the vertices added so far. Each PQ-tree represents one connected component of the current subgraph of G_1 or G_2. In the first phase we will add all private vertices of G_1 and G_2, and in the second phase we will add the common vertices in an st-order. When a common vertex is added, it will appear in two PQ-trees, one for G_1 and one for G_2 and we must take care to maintain compatibility. Note that if $V_1 = V_2$, i.e., when all vertices are common, only the second phase is needed and the whole algorithm will solely operate on two PQ-trees – one for each graph.

Before describing the two phases, we give the main idea of maintaining compatibility between two PQ-trees. Recall that in Section 2.1 we found compatible orders for two PQ-trees using projection and intersection of PQ-trees. However, we were unable to store a set of compatible orderings as a PQ-tree, which is what we really need, since planarity testing involves building a sequence of PQ-trees as we add vertices one by one.

To address this issue we introduce a boolean “orientation” variable attached to each Q-node to encode whether it is ordered forward or backward. Compatibility is captured by equations relating orientation variables. At the conclusion of the algorithm, it is a simple matter to see if the resulting set of Boolean equations has a solution. If it does, we use the solution to create compatible orderings of the Q-nodes of the two PQ-trees. Otherwise the graphs do not have simultaneous planar embeddings.

In more detail, we create a Boolean orientation variable $f(q)$ for each Q-node q, with the interpretation that $f(q) = \text{true}$ if and only if q has a “forward” ordering. We record the initial ordering of each Q-node in order to distinguish “forward” from “backward”. During PQ-tree operations, Q-nodes may merge, and during planarity testing, parts of PQ-trees may be deleted. We handle these modifications to Q-nodes by the simple expedient of having an orientation variable for each Q-node, and equating the variables as needed. When Q-nodes q_1 and q_2 merge, we add the equation $f(q_1) = f(q_2)$ if q_1 and q_2 are merged in the same order (both forward or both backward), or $f(q_1) = \neg f(q_2)$ otherwise.

We now describe the two phases of our simultaneous planarity testing algorithm. To process the private vertices of G_1 and G_2 in the first phase we compute for each graph an ordering by contracting G into a single vertex, computing a depth-first search from this vertex and finding a leaf-to-root order of the resulting tree. With these orderings we can now run the algorithm of Haeupler and Tarjan for all private vertices as described in Section 3.

Now the processed vertices induce a collection of components, such that each component has at least one out-edge to a common vertex. Further, for each component, the planarity test provides an associated PQ-tree representing all possible cyclic orderings of out-edges for that component. For each component we look at an arbitrary out-edge that goes to the first common vertex in the st-order and re-root the PQ-tree for this component to have this edge represented by the special leaf. This completes the first phase.

For the second phase we insert the common vertices in an st-order. The algorithm is similar to that described in Section 3 for an st-order but, in addition, has to take care of merging in the private components as well. We first examine the procedure for a single graph. Adding the first common vertex v_1 is a special set-up phase which we describe first; we will describe the general addition below.

Adding v_1 joins some of the private components into a new component C_1 containing v_1. For each of these private components we reduce the corresponding PQ-tree so that all the out-edges to v_1 appear together and then delete those edges. Note that due to the re-rooting at the end of the first phase the special leaf is among those edges. Thus the resulting PQ-tree represents the linear orderings of the remaining edges. We now build a PQ-tree representing the circular orderings around the new component C_1 as follows: we take v_1v_n as the special leaf, create a new P-node as a root and add all the out-edges of v_1 and the roots of the PQ-trees of the merged private components as children of the root (see Figure 4).

Now consider the situation when we are about to add the common vertex v_i, $i \geq 2$. The graph so far may have many connected components but because

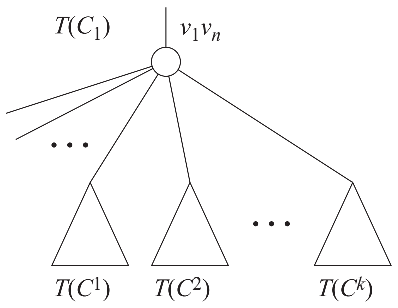

Figure 4: Setting up $T(C_1)$. The P-node's children are the outgoing edges of v_1 and the PQ-trees for the components that are joined together by v_1.

of the choice of an st-ordering, all common vertices embedded so far are in one component C_{i-1}, which we call the *main* component. When we add v_i, all components with out-edges to v_i join together to form the new main component C_i. This includes C_{i-1} and possibly some private components. The other private components do not change, nor do their associated PQ-trees.

We now describe how to update the PQ-tree T_{i-1} associated with C_{i-1} to form the PQ-tree T_i associated with C_i. This is similar to the approach described in Section 3. We first reduce T_{i-1} so that all the *black* edges (the ones incident to v_i) appear together (in any leaf-order of the tree). As before, we call a non-leaf node in the reduced PQ-tree *black* if all its leaf descendants are black. For any private component with an out-edge to v_i, we reduce the corresponding PQ-tree so that all the out-going edges to v_i appear together and then delete those edges. We make all the roots of the resulting PQ-trees into children of a new P-node p_i, and also add all the out-going edges of v_i as children of p_i. It remains to add p_i to T_{i-1} which we do as described below. In the process we also create a *black tree* J_i that represents the set of linear orderings of the black edges.

Case 1: *T_{i-1} contains a black node x such that all black edges are descendants of x.* Let J_i be the subtree rooted at x. We obtain T_i from T_{i-1} by replacing x and all its descendants with p_i and all of its descendants.

Case 2: *T_{i-1} contains a non-black Q-node x that has a sequence of adjacent black children.* We group all the black children of x and add them as children (in the same order) of a new Q-node x'. Let J_i be defined as the subtree rooted at x'. We add an equation relating the orientation variables of x and x'. We obtain T_i from T_{i-1} by replacing the sequence of black children of x (and their descendants) with p_i and all of its descendants (see Figure 5).

Note that we use the orientation variables above for a purpose other than compatibility. (We are only working with one graph so far.) Standard planarity tests would simply keep track of the order of the deleted subtree J_i in relation to its parent. Since we have orientation variables anyway, we use them for this purpose.

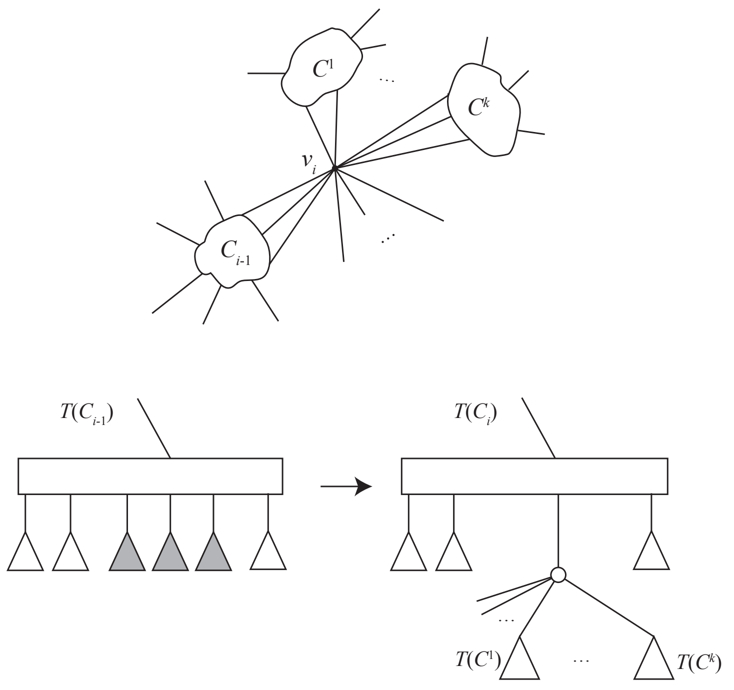

Figure 5: (*above*) Adding vertex v_i which is connected to main component C_{i-1} and to private components $C^1, \ldots, C^k$. (*below*) Creating T_i from T_{i-1} by replacing the black subtree[s] by a P-node whose children are the outgoing edges of v_i and the PQ-trees for the newly joined private components.

We perform the same procedure on graph G_2. We will distinguish the black trees of G_1 and G_2 using superscripts. Thus after adding v_i we have black trees J_i^1 and J_i^2. It remains to deal with compatibility. We claim that it suffices to enforce compatibility between each pair J_i^1 and J_i^2.

To do so, we perform a *unification step* in which we add equations between orientation variables for Q-nodes in the two trees.

Unification step for stage i

We first project J_i^1 and J_i^2 to the common edges, as described in Section 2.1, carrying over orientation variables from each original node to its copy in the projection (if it exists). Next we create the PQ-tree R_i that is the intersection of these two projected trees as described in Section 2.1. Initially R_i is equal to the first tree. The step dealing with Q-nodes (Step 3) is enhanced as follows:

3. For each Q-node q of the second tree, and for each pair a_1, a_2 of adjacent children of q do the following: Reduce R_i by adding a consecutivity constraint on all the descendant leaves of a_1 and a_2. Find the Q-node that is

the least common ancestor of the descendants of a_1 and a_2 in R_i. Add an equation relating the orientation variable of this ancestor with the orientation variable of q (using a negation if needed to match the orderings of the descendants).

Observe that any equations added during the unification step are necessary. Thus if the system of Boolean equations is inconsistent at the end of the algorithm, we conclude that G_1 and G_2 do not have a compatible combinatorial planar embedding. Finally, if the system of Boolean equations has a solution, then we obtain compatible leaf-orders for each pair J_i^1 and J_i^2 as follows: Pick an arbitrary solution to the system of Boolean equations. This fixes the truth values of all orientation variables and thus the orientations of all Q-nodes in all the trees. Subject to this, choose a leaf ordering I of R_i (by choosing the ordering of any P-nodes). The ordering I can then be lifted back to (compatible) leaf-orders of J_i^1 and J_i^2 that respect the ordering of I. The following lemma shows that this is sufficient to obtain compatible combinatorial planar embeddings of G_1 and G_2

Lemma 1 *The system of Boolean equations has a solution if and only if G_1 and G_2 have compatible combinatorial planar embeddings.*

Proof: Since each boolean equation in the system is necessary, if the system of equations does not have a solution then G_1 and G_2 do not have a compatible combinatorial embedding.

For the other direction, assume that the system of boolean equations has a solution. The procedure described above produces compatible leaf orders for all pairs of black trees J_i^1 and J_i^2. Recall that the leaves of J_i^1 (resp. J_i^2) are the out-edges of the component C_{i-1} in G_1 (resp. G_2) and contain all the common in-edges of v_i. Focusing on G_1 individually, its planarity test has succeeded, and we have a combinatorial planar embedding such that the ordering of edges around v_i contains the leaf order of J_i^1. Also, we have a combinatorial planar embedding of G_2 such that the ordering of edges around v_i contains the leaf order of J_i^2.

The embedding of a graph imposes an ordering of the out-edges around every main component. We can show inductively, starting from $i = n$, that the ordering of the out-edges around the main component C_{i-1} in G_1 is compatible with the ordering of the out-edges in the corresponding main component in G_2. Moreover all the common edges incident to v_i belong to either C_{i-1} or C_i. This implies that in both embeddings, the orderings of edges around any common vertex are compatible. Therefore G_1 and G_2 have compatible combinatorial planar embeddings. $\square$

4.1 Simultaneous planarity of k graphs

In this subsection we consider a generalization of simultaneous planarity for k graphs, when each vertex [edge] is either present in all the graphs or present in exactly one of them. Let $G_1 = (V_1, E_1), G_2 = (V_2, E_2), \cdots, G_k = (V_k, E_k)$

be k planar graphs such that $V = V_i \cap V_j$ for all distinct i, j and $E = E_i \cap E_j$ for all distinct i, j. As before, we call the edges and vertices of $G = (V, E)$ *common* and all other edges and vertices *private*. We show that the algorithm of Section 4 can be readily extended to solve this generalized version, when the common graph G is 2-connected.

If $G_1, G_2, \ldots, G_k$ have simultaneous planar embeddings then they clearly have mutually compatible combinatorial planar embeddings. Conversely if G_1, $G_2, \ldots, G_k$ have combinatorial planar embeddings that are mutually compatible, then, as before (see the beginning of Section 4), we can first find the planar embedding of the common subgraph and extend it to the planar embeddings of $G_1, G_2, \ldots, G_k$. Thus once again, the problem is equivalent to finding combinatorial planar embeddings for $G_1, G_2, \ldots G_k$, that are mutually compatible.

Our algorithm for finding such an embedding works as before, inserting private vertices first, followed by common vertices. The only difference comes in the unification step, where we have to take the intersection of k projected trees instead of 2. Doing this is straightforward: We initialize the intersection tree to be the first projected tree, and then insert the constraints of all the other trees into the intersection tree. Finally, Lemma 1 and its proof extend to multiple graphs.

4.2 Running time

We show that our algorithm can be implemented to run in linear time. (In the generalization to k graphs, the run time is linear in the sum of the sizes of the graphs—in other words, a common vertex counts k times.) Computing the leaf-to-root ordering of a depth-first spanning tree and the st-ordering are known to be doable in linear time [17]. The first phase of our algorithm uses PQ-tree based planarity testing with a leaf-to-root order of a depth-first spanning tree [23], which runs in linear time using the efficient PQ-tree implementation of Booth and Lueker [7, 8]. The re-rooting between the two phases needs to be done only once and can easily be done in linear time. The second phase of our algorithm uses PQ-tree based planarity testing with an st-order, as discussed in Section 3. This avoids re-rooting of PQ-trees, and thus also runs in linear time [8, 23, 30]. The other part of the second phase is the unification step, which is only performed on the black trees, i.e. the edges connecting to the current vertex. Note that these edges will get deleted and will not appear in subsequent stages. Thus we can explicitly store the black trees and the intersection tree at every stage and allow the unification step to take time linear in the total size of both black trees. The intersection algorithm can be implemented to run in linear time, as mentioned in Section 2.1. The last thing that needs to be implemented efficiently is the handling of the orientation variables. It is easy to see that once the equations are generated, they can be solved in linear time, by repeatedly fixing the value of a free variable (true or false), finding all the equations that contain the variable and recursively (say in a depth-first way) fixing all the variables so as to satisfy the equations. We now explain how to generate the variable equations in linear time.

Note that in the implementation of PQ-trees (see Booth and Lueker [8]) the children of a Q-node are stored in a doubly-linked list and only the leftmost and rightmost children have parent pointers. Thus when two Q-nodes, one a child of the other, merge, we may not know the variable and the orientation of the parent Q-node. To address this problem, we use labels on certain links of the doubly-linked list as explained below and compute all the equations generated by the reductions of a unification step at the end of the step.

For any two adjacent child nodes c_i and c_{i+1} of a Q-node q, either the links $c_i \rightarrow c_{i+1}$ and $c_{i+1} \rightarrow c_i$ are labeled with l and $\neg l$ (respectively), for some literal l, or they are both unlabeled. The underlying interpretation is that c_i appears before c_{i+1} in the child ordering of q iff l is true. Thus the literals that we encounter when traveling from one end to the other of the doubly-linked list, are all (implicitly) equal. When a Q-node is first created with two child nodes, say x and y, we create a variable associated with it and label the link from x to y with the variable and the link from y to x with the negation of the variable.

During the algorithm, there are two types of equations: (1) Equations consisting of literals appearing in Q-nodes of distinct trees. (These can be PQ-trees of the same graph, as happens in Case 2 of Section 4 or PQ-trees of different graphs, as happens in step 3 of Unification.) (2) Equations consisting of literals appearing in Q-nodes of a single tree (created during PQ-tree reductions).

Note that type 1 equations are essentially equations that constrain the ordering of child nodes across the two Q-nodes, and we handle them as follows. Let c_1, c_2 be any two adjacent child nodes of the first Q-node that are constrained to appear in the same order as child nodes c'_1, c'_2 of the second Q-node. If the links between c_1 and c_2 are unlabeled, we create a new variable x and label the links $c_1 \rightarrow c_2$ and $c_2 \rightarrow c_1$ with x and $\neg x$ respectively. Similarly, we label the links between c'_1 and c'_2, if they are unlabeled. Now we create the equation that equates the literal associated with $c_1 \rightarrow c_2$ with the literal associated with $c'_1 \rightarrow c'_2$.

Type 2 equations happen when two Q-nodes merge. In this case the merged node contains literals from both the Q-nodes. Instead of computing the equation for each merge immediately, we compute the equations in a lazy fashion at the end of each unification step as follows. For every Q-node of the two black trees and the intersection tree obtained from their projections (i.e. the output tree of the unification step on the black trees), we pass from the first child to the last child and equate all the literals encountered in the labels of the links. This clearly takes linear time in the size of the black trees.

Thus we have proved the following Theorem.

Theorem 1 *Let $G_1, G_2, \ldots, G_k$ be k planar graphs such that any two of them share the same 2-connected subgraph. Then we can test whether $G_1, G_2, \ldots, G_k$ are simultaneously planar in $O(\sum_{i=1}^{k} |V(G_i)|)$ time.*

4.3 Simultaneous planar drawings

Our algorithm to test simultaneous planarity finds compatible combinatorial planar embeddings but does not find actual simultaneous drawings. In this section we initiate the study of how to compute "nice" simultaneous drawings. Our results are preliminary and more theoretical than practical. Two natural goals are to minimize the number of bends along the polygonal curves representing the edges and to minimize the number of crossings between the private edges of the two graphs. We will bound both measures, and will achieve polynomial time, but the area bound for our drawings is exponential. We limit our discussion to the case where the common graph is connected.

Theorem 2 *Given compatible combinatorial embeddings for two graphs G_1 and G_2 whose shared graph is connected and has n vertices, we can find planar drawings for G_1 and G_2 in polynomial time such that each pair of private edges intersects at most once and each edge has at most n bends. Furthermore, any edge of G_1 and any private edge of G_2 that joins private vertices of G_2 will be drawn as a straight line segment.*

Take a straight-line planar drawing of the first graph, G_1. This includes a straight-line drawing of the common graph G, which we will extend to a planar drawing of G_2.

The main idea is to draw the private parts of G_2 inside the faces of the drawing of G using shortest paths. (Of course, we must offset the shortest paths slightly to avoid coincident features.) A shortest path inside a polygon has two crucial properties: it bends only at reflex vertices of the polygon, and it intersects any line segment contained in the polygon at most once.

Any bend on an edge of G_2 will occur in the neighborhood of a common vertex (a vertex of G). We will maintain the following invariants: (1) each edge of G_2 has at most one bend per common vertex; (2) the boundary of each face of G_2 has at most one reflex bend per common vertex; and (3) any edge of G_1 intersects any face of G_2 in a single line segment. Property (1) implies that each edge of G_2 has at most n bends. Property (2) allows us to maintain property (1) as we draw more of G_2. Property (3) ensures that private edges of G_1 cross private edges of G_2 at most once. Initially, the only part of G_2 that is drawn is G, and the properties are satisfied.

We first draw the private edges of G_2 whose endpoints are in G. See Figure 6. We draw the edges one by one preserving the invariants. Consider one such edge e. The combinatorial embedding specifies the face, f, of the partial embedding of G_2 that contains the edge. Face f is drawn as a polygon, possibly as the outside of a polygon in case f is the outer face. Also note that the polygon of f may have repeated vertices since the common graph may have cut vertices. In such a case, note that the combinatorial embedding specifies the endpoints of e unambiguously. The edge e cuts f into two faces. By property (2), any reflex vertex of the polygon of f is in the neighborhood of a unique common vertex. Draw the edge e as a shortest path in f, pulling it slightly away from the boundary of f. Observe that this satisfies our invariants.

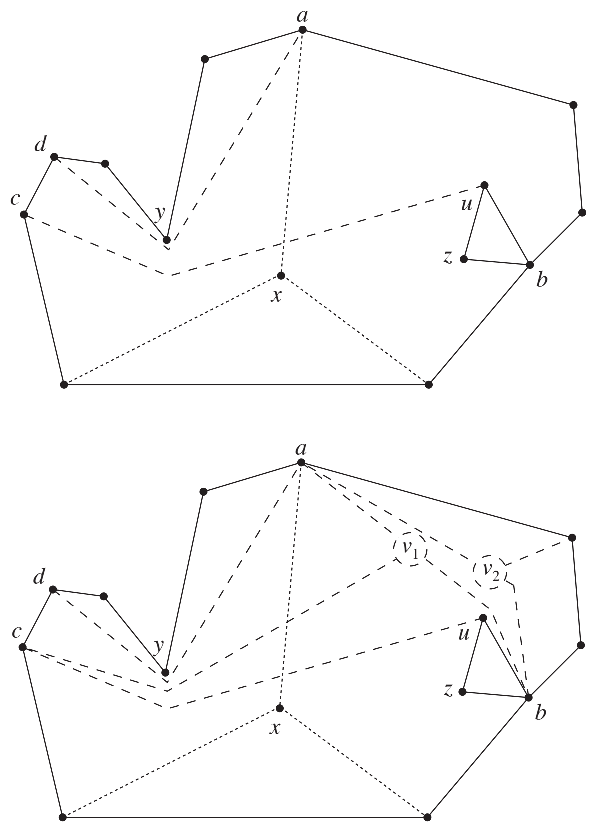

Figure 6: Extending a drawing of G to a drawing of G_2. Solid lines indicate a face of G with a repeated vertex at b (the triangle bzu is outside the face). Private edges of G_1 are drawn with dotted lines. (*above*) Adding two private edges of G_2 (drawn with dashed lines) that have their endpoints in G, specifically (d, a) and (c, u), each drawn with a bend in the neighborhood of the common vertex y. Observe that although there are two bends near y, each face of G_2 has only one reflex bend near y. (*below*) Adding private vertices of G_2. Two connected components in the private part of G_2 are contracted to vertices v_1 and v_2 and these vertices are drawn along a shortest path between two neighbours—in this case the same two neighbours a and b. The corresponding connected components are then drawn in small disjoint discs around v_1 and v_2.

We note that without assuming that the common graph is connected, face f could be a polygon with holes, and edge e would not necessarily cut f into two regions. This could then violate property (3) which is why a different construction is needed for this case.

We now insert into the drawing the private vertices of G_2 and their incident edges. This is done in two stages. In the first stage we contract every connected component of $G_2 - (V_1 \cap V_2)$ to a single vertex and add these vertices to the drawing. In the second stage we expand these vertices to the original connected components.

For the first stage, we consider the connected components of $G_2 - (V_1 \cap V_2)$ one by one. Let C be such a connected component consisting of some private vertices and their induced edges. We will also be concerned with the edges that join C to the common graph G. The combinatorial embedding specifies the face, f, of the partial embedding of G_2 that contains C. Face f is drawn as a polygon and, by property (2), any reflex vertex of the polygon is in the neighborhood of a unique common vertex. We will contract C to a single vertex v and draw v as a point together with all the edges joining it to common vertices, while preserving our invariants. Note that the ordering of edges incident to v is determined from the orderings of edges incident to the vertices of C. If v is adjacent to only one common vertex, we can augment G_2 to make it adjacent to another common vertex. Let a and b be two common vertices adjacent to v. Draw a shortest path inside f from a to b (pulling it away from reflex vertices as before) and place v at some point interior to a line segment of that path. Thus the edges (v, a) and (v, b) form $180°$ angles at v. Draw the remaining edges incident to v as shortest paths in the polygon, pulling the paths slightly away from the boundary of f and from each other to avoid coincident features. Observe that v is not a reflex vertex in any of the resulting faces. This construction satisfies our invariants: (1) each edge has at most one bend per reflex vertex of f; (2) the boundary of each resulting face has at most one reflex bend per vertex of f; and (3) any edge of G_1 intersects any face of G_2 in a single line segment.

For the second stage, we take a disc D around each point v, small enough so that the discs are pairwise disjoint, and so that each disc is disjoint from the rest of the drawing. See Figure 6 for an example. It remains to find a straight-line drawing of component C inside disc D while connecting the appropriate vertices of C to the edges that leave the disc. The basic idea is to fix the points where edges leave D, and add those points as new vertices, yielding an augmented graph C' whose outer face must be drawn as a fixed convex polygon. This can be done via Tutte's graph drawing algorithm. See Figure 7. We now fill in the details of this approach. First note that a single edge e from v to a common vertex, say a, may represent multiple edges from vertices of C to a. Suppose e represents some number $t > 1$ of edges. Consider the path representing e in the drawing and suppose it exits disc D at point p. We create t copies of p closely spaced on D, and replace the path from p to a by t paths that go from the copies of p to a.

We are now in the situation where every edge e from a vertex x of C to a common vertex a is represented by a path from a point $p(e)$ on D to a.

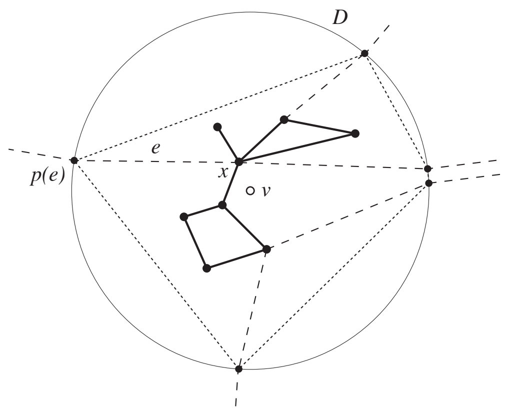

Figure 7: Drawing a connected component C (solid edges) of $G_2 - (V_1 \cap V_2)$ inside a disc D. Points $p(e)$ on the boundary of the disc D are fixed and become bends in the edges (dashed) from C to the common graph. The augmented graph C' consists of C together with vertices $p(e)$ each joined to the appropriate vertex of C and a cycle through points $p(e)$ (dotted).

Furthermore, the ordering of the points around D is the same as the ordering of edges leaving component C. We augment the graph C and its combinatorial planar embedding by adding, for each such edge e, a vertex $p(e)$ and an edge $(x, p(e))$. In case D only has two points on it we add a third dummy vertex. Finally, we add a cycle through the vertices $p(e)$ in the order around D. Let C' be the resulting graph. Augment C' to a 3-connected graph. By Tutte's graph drawing algorithm [35], there is a planar straight-line drawing of C' with the vertices of the outer face drawn as the fixed convex polygon on the $p(e)$'s.

Note that we have introduced one extra bend on each edge that goes from a vertex of C to a common vertex a. We claim that the bound of n bends per edge is still valid. We claimed earlier that the edge from v to a was drawn with at most one bend per reflex vertex of the enclosing face f, which gave the upper bound of n. However, note that a itself is a vertex of f, and does not introduce a bend. Thus the earlier bound can be tightened to $n-1$ bends, and one extra bend is fine. This completes the proof.

5 Conclusions

We have given a linear-time algorithm to test simultaneous planarity of two graphs whose shared subgraph is 2-connected. Our algorithm does not require the two graphs to have the same vertex set. Furthermore, our algorithm works for the more general case when there are k graphs, any two of which have the same 2-connected subgraph in common.

We also showed how to construct simultaneous planar drawings while limiting the number of bends and crossings. Our method applies when the common graph is connected although we believe the result holds more generally. We leave as an open question the construction of simultaneous planar drawings on a small grid.

We conjecture that simultaneous planarity of multiple graphs can be tested in polynomial time when any vertex/edge is either private to a single graph or common to all graphs. Our algorithm solves the special case when the common graph is 2-connected.

Note that simultaneous planarity is known to be NP-hard for three graphs in general [21]. The three graphs constructed in the reduction have edges in all possible combinations: edges private to each graph, edges common to all three graphs, and edges common to a pair of graphs but not the third, for every pair. Another interesting case of simultaneous planarity for multiple graphs is when the graphs intersect in *layers*: the graphs are ordered $G_1, \ldots, G_k$, and every vertex/edge occurs in a consecutive subsequence of the graphs. This layered structure arises from a graph changing over time, so long as a vertex/edge does not re-appear after it vanishes. The complexity of layered simultaneous planarity is open, even for three graphs. For three graphs the layered structure is equivalent to the condition that if a vertex/edge is in G_1 and G_3 then it is also in G_2.

A weaker version of simultaneous planarity for two graphs requires only that each common vertex be represented by the same point—a common edge may be represented by different curves in the two planar drawings. Any pair of planar graphs can be represented this way: after choosing arbitrary vertex positions, each graph can be drawn independently, as shown by Pach and Wenger [34]. It would be interesting to generalize our algorithm to the case where some designated common edges are allowed to be represented by different curves in the two drawings. There is also a natural optimization version of the problem: given two planar graphs with some common vertices and edges, find planar drawings of the graphs so that every common vertex is drawn as the same point in the two drawings, and maximize the number of common edges that are drawn as the same curve in the two drawings.

Lastly, both SPQR-trees [13] and PQ-trees have been used for many planarity related problems and both give rise to distinct representations of all planar embeddings of a (2-connected) planar graph [11, 13, 23]. Understanding the strength of both representations and how they relate is an important question. Comparing the PQ-tree approach to simultaneous planarity taken here with the SPQR-tree approach in [2] or [18] might be a good start. The recent work [6] makes very interesting progress in this direction.

Acknowledgment We want to thank Bob Tarjan. Some of the ideas used here go back to joint research that was done for a journal version of [23]. We would also like to thank the anonymous referees for many comments that improved the presentation of this paper. In particular we want to thank one referee for help with the drawing method in Section 4.3.

References

[1] P. Angelini, G. Di Battista, F. Frati, V. Jelínek, J. Kratochvíl, M. Patrignani, and I. Rutter. Testing planarity of partially embedded graphs. In *ACM-SIAM Symposium on Discrete Algorithms '10*, pages 202–221, 2010. URL: `http://dl.acm.org/citation.cfm?id=1873601.1873620`.

[2] P. Angelini, G. Di Battista, F. Frati, M. Patrignani, and I. Rutter. Testing the simultaneous embeddability of two graphs whose intersection is a biconnected graph or a tree. In *International Workshop on Combinatorial Algorithms '10*, volume 6460 of *LNCS*, pages 212–225, 2011. `doi:10.1007/978-3-642-19222-7_22`.

[3] P. Angelini, G. Di Battista, F. Frati, M. Patrignani, and I. Rutter. Testing the simultaneous embeddability of two graphs whose intersection is biconnected or a connected graph. *Journal of Discrete Algorithms*, 14:150–172, 2012. `doi:10.1016/j.jda.2011.12.015`.

[4] P. Angelini, M. Geyer, M. Kaufmann, and D. Neuwirth. On a tree and a path with no geometric simultaneous embedding. *Journal of Graph Algorithms and Applications*, 16(1):37–83, 2012. `doi:10.7155/jgaa.00250`.

[5] T. Bläsius and I. Rutter. Disconnectivity and Relative Positions in Simultaneous Embeddings. In W. Didimo and M. Patrignani, editors, *Graph Drawing 2012*, volume 7704 of *LNCS*, pages 31–42. 2013. `doi:10.1007/978-3-642-36763-2_4`.

[6] T. Bläsius and I. Rutter. Simultaneous PQ-Ordering with Applications to Constrained Embedding Problems. In *ACM-SIAM Symposium on Discrete Algorithms '13*, 2013.

[7] K. Booth. *PQ Tree Algorithms*. PhD thesis, University of California, Berkeley, 1975.

[8] K. Booth and G. Lueker. Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms. *Journal of Computer and System Sciences*, 13:335—379, 1976. `doi:10.1016/S0022-0000(76)80045-1`.

[9] J. Boyer and W. Myrvold. On the cutting edge: Simplified O(n) planarity by edge addition. *Journal of Graph Algorithms and Applications*, 8(3):241–273, 2004. `doi:10.7155/jgaa.00091`.

[10] P. Brass, E. Cenek, C. Duncan, A. Efrat, C. Erten, D. Ismailescu, S. G. Kobourov, A. Lubiw, and J. Mitchell. On simultaneous planar graph embeddings. *Computational Geometry: Theory and Applications*, 36(2):117–130, 2007. `doi:10.1016/j.comgeo.2006.05.006`.

[11] N. Chiba, T. Nishizeki, A. Shigenobu, and T. Ozawa. A linear algorithm for embedding planar graphs using PQ-trees. *Journal of Computer and System Sciences*, 30(1):54–76, 1985. doi:10.1016/0022-0000(85)90004-2.

[12] W. J. Cook, W. H. Cunningham, W. R. Pulleyblank, and A. Schrijver. *Combinatorial Optimization.* Wiley-Interscience, 1997.

[13] G. Di Battista and R. Tamassia. On-line planarity testing. *SIAM Journal on Computing*, 25(5):956–997, 1996. doi:10.1137/S0097539794280736.

[14] E. Di Giacomo and G. Liotta. Simultaneous embedding of outerplanar graphs, paths, and cycles. *International Journal of Computational Geometry and Applications*, 17(2):139–160, 2007. doi:10.1142/S0218195907002276.

[15] C. Erten and S. G. Kobourov. Simultaneous embedding of planar graphs with few bends. *Journal of Graph Algorithms and Applications*, 9(3):347–364, 2005. doi:10.7155/jgaa.00113.

[16] A. Estrella-Balderrama, E. Gassner, M. Junger, M. Percan, M. Schaefer, and M. Schulz. Simultaneous geometric graph embeddings. In *Graph Drawing '07*, volume 4875 of *LNCS*, pages 280–290, 2008. doi:10.1007/978-3-540-77537-9_28.

[17] S. Even and R. Tarjan. Computing an *st*-numbering. *Theoretical Computer Science*, 2(3):339–344, 1976. doi:10.1016/0304-3975(76)90086-4.

[18] J. Fowler, C. Gutwenger, M. Jünger, P. Mutzel, and M. Schulz. An SPQR-tree approach to decide special cases of simultaneous embedding with fixed edges. In *Graph Drawing '08*, volume 5417 of *LNCS*, pages 157–168, 2009. doi:10.1007/978-3-642-00219-9_16.

[19] J. J. Fowler, M. Jünger, S. G. Kobourov, and M. Schulz. Characterizations of restricted pairs of planar graphs allowing simultaneous embedding with fixed edges. *Computational Geometry*, 44(8):385 – 398, 2011. doi:10.1016/j.comgeo.2011.02.002.

[20] F. Frati. Embedding graphs simultaneously with fixed edges. In *Graph Drawing '06*, volume 4372 of *LNCS*, pages 108–113, 2007. doi:10.1007/978-3-540-70904-6_12.

[21] E. Gassner, M. Jünger, M. Percan, M. Schaefer, and M. Schulz. Simultaneous graph embeddings with fixed edges. In *Workshop on Graph-Theoretic Concepts in Computer Science '06*, volume 4271 of *LNCS*, pages 325—335, 2006. doi:10.1007/11917496_29.

[22] B. Haeupler, K. R. Jampani, and A. Lubiw. Testing simultaneous planarity when the common graph is 2-connected. In *International Symposium on Algorithms and Computation '10*, volume 6507 of *LNCS*, pages 410–421, 2011. doi:10.1007/978-3-642-17514-5_35.

[23] B. Haeupler and R. E. Tarjan. Planarity algorithms via PQ-trees (extended abstract). *Electronic Notes in Discrete Mathematics*, 31:143–149, 2008. doi:10.1016/j.endm.2008.06.029.

[24] S. Hong and H. Nagamochi. Two-page book embedding and clustered graph planarity. Technical Report 2009-004, Dept. of Applied Mathematics and Physics, University of Kyoto, Japan, 2009.

[25] W. Hsu and R. McConnell. PC-trees and circular-ones arrangements. *Theoretical Computer Science*, 296(1):99–116, 2003. doi:10.1016/S0304-3975(02)00435-8.

[26] K. Jampani and A. Lubiw. Simultaneous interval graphs. In *International Symposium on Algorithms and Computation '10*, volume 6506 of *LNCS*, pages 206–217, 2011. doi:10.1007/978-3-642-17517-6_20.

[27] K. R. Jampani and A. Lubiw. The simultaneous representation problem for chordal, comparability and permutation graphs. In *Algorithms and Data Structures Symposium '10*, volume 5664 of *LNCS*, pages 387–398, 2009. doi:10.1007/978-3-642-03367-4_34.

[28] M. Jünger and S. Leipert. Level planar embedding in linear time. *Journal of Graph Algorithms and Applications*, 6(1):67–113, 2002. doi:10.7155/jgaa.00045.

[29] M. Jünger and M. Schulz. Intersection graphs in simultaneous embedding with fixed edges. *Journal of Graph Algorithms and Applications*, 13(2):205—218, 2009. doi:10.7155/jgaa.00184.

[30] A. Lempel, S. Even, and I. Cederbaum. An algorithm for planarity testing of graphs. In *Theory of Graphs: International Symposium '67*, pages 215–232, 1967.

[31] B. Mohar and C. Thomassen. *Graphs on Surfaces*. Johns Hopkins University Press, 2001.

[32] T. Nishizeki and N. Chiba. *Planar Graphs: Theory and Algorithms*. North Holland, 1988.

[33] T. Nishizeki and M. S. Rahman. *Planar Graph Drawing*. World Scientific, 2004.

[34] J. Pach and R. Wenger. Embedding planar graphs at fixed vertex locations. *Graphs and Combinatorics*, 17(4):717–728, 2001. doi:10.1007/PL00007258.

[35] W. T. Tutte. How to Draw a Graph. *Proceedings of the London Mathematical Society*, s3-13(1):743–767, 1963. doi:10.1112/plms/s3-13.1.743.

[36] S. Wei-Kuan and H. Wen-Lian. A new planarity test. *Theoretical Computer Science*, 223(1-2):179 – 191, 1999. doi:10.1016/S0304-3975(98)00120-0.