Testing Simultaneous Planarity When the Common Graph Is 2-Connected

Bernhard Haeupler[1], Krishnam Raju Jampani[2], and Anna Lubiw[2]

[1] CSAIL, Dept. of Computer Science,
Massachusetts Institute of Technology, Cambridge, MA 02139
haeupler@mit.edu
[2] David R. Cheriton School of Computer Science,
University of Waterloo, Waterloo, ON, Canada, N2L 3G1
{krjampan,alubiw}@uwaterloo.ca

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 two graphs share a 2-connected subgraph. 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.

Keywords: Simultaneous Embedding, Planar Graph, PQ Tree, Graph Drawing.

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.

The study of planar graphs has a long history and has generated many deep results [24,23,25]. 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 carry over from one matroid to the intersection of two matroids [7]. 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 embeddings, which have been called "simultaneous embeddings with fixed

O. Cheong, K.-Y. Chwa, and K. Park (Eds.): ISAAC 2010, Part II, LNCS 6507, pp. 410–421, 2010.

edges" [8,9,12,13,14,15,21]. 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 [15] 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 [10]. On the other hand several classes of (pairs of) graphs are known to always have simultaneous planar embeddings [9,8,14,13,21] and there are efficient algorithms to test simultaneous planarity for some very restricted graph-classes: biconnected outerplanar graphs [13], and the case where one graph has at most one cycle [12].

This paper shows how to efficiently test simultaneous planarity of any two graphs that share a 2-connected subgraph and thus greatly extends the classes of graph pairs for which a testing algorithm is known. Our algorithm builds on the planarity testing algorithm of Haeupler and Tarjan [17], which in turn unifies the planarity testing algorithms of Lempel-Even-Cederbaum [22], Shih-Hsu [27] and Boyer-Myrvold [5].

The 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. We present our algorithm for simultaneous planarity in Section 3. We also show that our algorithm can be extended to solve a generalization of simultaneous planarity for k graphs, whose common graph is 2-connected.

1.1 Background

Versions of simultaneous planarity have received much attention in recent years. Brass et al. [6] 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 [10]. For other work on simultaneous geometric embeddings see [2] 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 [9] 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 Schultz give a characterization in terms of "compatible embeddings" [Theorem 4 in [21]]. 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 [9,8,14] 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 [14]. On the other hand, there is an example of two outerplanar graphs that have no simultaneous planar embedding [14].

The graphs that have simultaneous planar embeddings when paired with any other planar graph have been characterized [13]. In addition, Jünger and Schultz [21] 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 biconnected outerplanar graphs [13] and for a pair consisting of a planar graph and a graph with at most one cycle [12].

Simultaneously and independently of this work Angelini et al. [1] showed how to test simultaneous planarity of two graphs when the common graph is 2-connected. Their algorithm is based on SPQR-trees, takes $O(n^3)$ time and is restricted to the case where the two graphs have the same vertex set.

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 [26].

The idea of "simultaneous graph representations" has also been applied to intersection representations [19,18].

2 PQ-Trees

Many planarity testing algorithms in the literature use PQ-trees (or a variation) to obtain a linear-time implementation. PQ-trees were discovered by Booth and Lueker [4] 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. The leaves of a PQ-tree correspond to the elements of the set, and internal nodes are labeled 'P' or 'Q', and are drawn using a circle or a rectangle, respectively. 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 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 [4] gave an efficient implementation of PQ-trees that supports this operation in amortized $O(|S|)$ time.

Although PQ-trees were invented to represent linear orders, they can be reinterpreted to represent circular orders as well [17]: 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. 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.

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. The idea for this 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 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 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 its 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 such an intersection can be computed in time linear in the size of the two PQ-trees (see Booth's thesis [3]).

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 [20] 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 our structure of two orderings that are consistent on common elements cannot be captured by a single PQ tree.

3 Planarity

In this Section, we review the recent algorithm of Haeupler and Tarjan [17] for testing the planarity of a graph. Next we extend it to an algorithm for testing simultaneous planarity. We first begin with some basic definitions.

Let $G = (V, E)$ be a 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 an in-edge and an 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 [11].

A *combinatorial embedding* of G, denoted by $\mathcal{C}(G)$, is defined as a clockwise circular ordering of the incident edges of v_i, for each $i \in \{1, \cdots, n\}$, with respect to a planar drawing of G. If $\mathcal{C}$ is a combinatorial embedding of G, we use $\mathcal{C}(v_i)$ to denote the clockwise circular ordering of edges incident with v_i in $\mathcal{C}$.

3.1 Planarity Testing Using PQ-Trees

Let $G = (V, E)$ be a connected graph. The planarity testing algorithm of Haeupler and Tarjan embeds vertices (and their edges) one at a time and maintains all possible partial embeddings of the embedded subgraph at each stage. 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 the remaining vertices induce a connected graph and hence lie in a single face of the partial embedding at any time. 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 algorithm uses a mixture of the two orders: We first add the vertices that are contained in only one of the graphs using a depth-first search order and then add the common vertices using an st-ordering. We now give an overview of how the simple planarity test works for each of these orderings.

st-order:
Let $v_1, v_2, \cdots, v_n$ be an st-order of G. At any stage $i \in \{1, \cdots, n-1\}$ the vertices $\{v_1, \cdots, v_i\}$ form a connected component and the algorithm maintains all possible circular orderings of out-edges around this component using a PQ-tree T_i. Since v_1v_n is an out-edge at every stage, it can stay as the special leaf of T_i for all i. 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 $i \in \{1, \cdots, n-2\}$. We call the set of leaves of T_i that correspond to edges incident to v_{i+1}, the *black* leaves. To go to the next stage, we first reduce T_i so that all the black edges appear together. A non-leaf node in the reduced PQ-tree is said to be black if all its 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}.

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.

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 components. We maintain a PQ-tree for each component representing the set of 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. Adding v_{i+1} can lead to merging several components into one.

To go to the next stage, we first reduce each PQ-tree corresponding to such a component by adding a consecutivity constraint that requires the set of out-edges that are incident to v_{i+1} to be consecutive and then deleting these edges. 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 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 1). Note that by choosing the edge $v_{i+1}v_l$ as the special leaf we again maintain the above mentioned invariant.

As before, if the reduction step fails for any component, then the graph is non-planar. Otherwise the algorithm concludes that the graph is planar.

3.2 Simultaneous Planarity

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 $|V_2| = n_2$. Let $G = (V_1 \cap V_2, E_1 \cap E_2)$ be 2-connected and $n = |V_1 \cap V_2|$. Let $v_1, v_2, \cdots, v_n$ be an st-ordering of $V_1 \cap V_2$. 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 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, they have combinatorial embeddings that are compatible with each other. If the common edges form a connected graph the converse is also true and is a special case of Theorem 4 of Jünger and Schultz [21]. Thus it is enough to compute a pair of compatible combinatorial embeddings.

We will find compatible combinatorial embeddings by adding vertices one by one, iteratively constructing two sets of PQ-trees, representing the partial planar embeddings of G_1 and of G_2 respectively. Each PQ-tree represents one connected component 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-ordering. 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.

Before describing the two phases, we give the main idea of maintaining compatibility between two PQ-trees. In Section 2.1 we found compatible orders for two PQ-trees using projection and intersection of PQ-trees, but we were unable to store a set of compatible orderings, which is what we really need, since planarity testing involves a sequence of PQ-trees.

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 $f(q) =$ True iff 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 of them a reverse depth-first ordering by contracting G into a single vertex and then running a depth-first search from this vertex. With these orderings we can now run the algorithm of Haeupler and Tarjan for all private vertices as described in Section 3.1.

Now the processed vertices induce a collection of components, such that each component has an out-edge to a common vertex. Further, the planarity test provides us for each component with an associated PQ-tree representing all possible cyclic orderings of out-edges for that component. For each component we look at the 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 3.1 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; 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 Fig. 1).

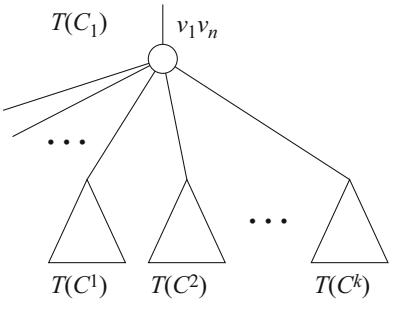

Fig. 1. Setting up $T(C_1)$. The P-node's children are the outgoing edges of v_1 and the PQ-trees for the components that are joined together by v_1.

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 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 3.1. We first reduce T_{i-1} so that all the *black* edges (the ones incident to v_i) appear together. As before, we call a non-leaf node in the reduced PQ-tree *black* if all its descendants are black leaves. 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.

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 (see Figure 2).

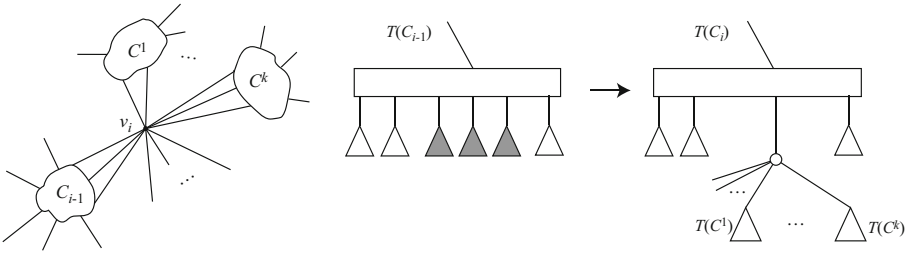

Fig. 2. (*left*) Adding vertex v_i which is connected to main component C_{i-1} and to private components $C^1 \dots C^k$. (*right*) Creating T_i from T_{i-1} by replacing the black subtree by a P-node whose children are the outgoing edges of v_i and the PQ-trees for the newly joined private components.

Note that we use 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.

We perform a similar 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. 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 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). 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 embeddings of G_1 and G_2.

Lemma 1. *If the system of Boolean equations has a solution then G_1 and G_2 have compatible combinatorial embeddings.*

Proof. 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. Focussing on G_1 individually, its planarity test has succeeded, and we have a combinatorial embedding such that the ordering of edges around v_i contains the leaf order of J_i^1. Also, we have a combinatorial 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 embeddings.

A generalization of simultaneous planarity to k graphs. 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. Our algorithm of section 3.2 can be readily extended to solve this generalized version, when the common graph is 2-connected (see the full paper [16] for more details).

3.3 Running Time

We show that our algorithm can be implemented to run in linear time. Computing the reverse depth-first ordering and the st-ordering are known to be feasible in linear-time [11]. The first phase of our algorithm uses PQ-tree based planarity testing with a reverse depth-first search order [17], which runs in linear time using the efficient PQ-tree implementation of Booth and Lueker [3,4]. 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 3.1. This avoids re-rooting of PQ-trees, and thus also runs in linear time [17,4,22]. 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. 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 complete size of both black trees. The intersection algorithm needs to be implemented with a little bit more care but again, using the standard PQ-tree implementation and the intersection algorithm described in Booth's thesis [3], linear time is possible. The last thing that needs to be implemented efficiently is the handling of the orientation variables. Note that a Q-node is implemented as a doubly-linked list of its children [4]. By storing the variable and orientation information of a Q-node in one of the links of the doubly-linked list and generating equations lazily at the end of each unification step, we can generate all variable equations in linear time (see full paper for details). Once generated, the equations can clearly be solved in linear time.

Acknowledgment. We want to thank Bob Tarjan. Some of the ideas used here go back to joint research that was done for a complete version of [17].

References

1. Angelini, P., Di Battista, G., Frati, F., Patrignani, M., Rutter, I.: Testing the simultaneous embeddability of two graphs whose intersection is a biconnected graph or a tree. In: IWOCA. LNCS (2010)
2. Angelini, P., Geyer, M., Kaufmann, M., Neuwirth, D.: On a tree and a path with no geometric simultaneous embedding. In: CoRR, abs/1001.0555 (2010)
3. Booth, K.: PQ Tree Algorithms. PhD thesis, University of California, Berkeley (1975)
4. Booth, K., Lueker, G.: 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)
5. Boyer, J., Myrvold, W.: On the cutting edge: Simplified O(n) planarity by edge addition. Journal of Graph Algorithms and Applications 8(3), 241–273 (2004)
6. Brass, P., Cenek, E., Duncan, C., Efrat, A., Erten, C., Ismailescu, D., Kobourov, S.G., Lubiw, A., Mitchell, J.: On simultaneous planar graph embeddings. Computational Geometry: Theory and Applications 36(2), 117–130 (2007)
7. Cook, W.J., Cunningham, W.H., Pulleyblank, W.R., Schrijver, A.: Combinatorial Optimization. Wiley Interscience, Hoboken (1997)

8. DiGiacomo, G., Liotta, G.: Simultaneous embedding of outerplanar graphs, paths, and cycles. International Journal of Computational Geometry and Applications 17(2), 139–160 (2007)
9. Erten, C., Kobourov, S.G.: Simultaneous embedding of planar graphs with few bends. Journal of Graph Algorithms and Applications 9(3), 347–364 (2005)
10. Estrella-Balderrama, A., Gassner, E., Junger, M., Percan, M., Schaefer, M., Schulz, M.: Simultaneous geometric graph embeddings. In: Hong, S.-H., Nishizeki, T., Quan, W. (eds.) GD 2007. LNCS, vol. 4875, pp. 280–290. Springer, Heidelberg (2008)
11. Even, S., Tarjan, R.: Computing an st-Numbering. Theor. Comput. Sci. 2(3), 339–344 (1976)
12. Fowler, J., Gutwenger, C., Junger, M., Mutzel, P., Schulz, M.: An SPQR-tree approach to decide special cases of simultaneous embedding with fixed edges. In: Tollis, I.G., Patrignani, M. (eds.) GD 2008. LNCS, vol. 5417, pp. 1–12. Springer, Heidelberg (2009)
13. Fowler, J., Jünger, M., Kobourov, S.G., Schulz, M.: Characterizations of restricted pairs of planar graphs allowing simultaneous embedding with fixed edges. In: Broersma, H., Erlebach, T., Friedetzky, T., Paulusma, D. (eds.) WG 2008. LNCS, vol. 5344, pp. 146–158. Springer, Heidelberg (2008)
14. Frati, F.: Embedding graphs simultaneously with fixed edges. In: Kaufmann, M., Wagner, D. (eds.) GD 2006. LNCS, vol. 4372, pp. 108–113. Springer, Heidelberg (2007)
15. Gassner, E., Junger, M., Percan, M., Schaefer, M., Schulz, M.: Simultaneous graph embeddings with fixed edges. In: Fomin, F.V. (ed.) WG 2006. LNCS, vol. 4271, pp. 325–335. Springer, Heidelberg (2006)
16. Haeupler, B., Jampani, K.R., Lubiw, A.: Testing simultaneous planarity when the common graph is 2-connected (2010)
17. Haeupler, B., Tarjan, R.E.: Planarity algorithms via PQ-trees (extended abstract). Electronic Notes in Discrete Mathematics 31, 143–149 (2008)
18. Jampani, K.R., Lubiw, A.: The simultaneous representation problem for chordal, comparability and permutation graphs. In: WADS. LNCS, vol. 5664, pp. 387–398. Springer, Heidelberg (2009)
19. Jampani, K.R., Lubiw, A.: Simultaneous interval graphs (2010) (submitted)
20. Jünger, M., Leipert, S.: Level planar embedding in linear time. J. Graph Algorithms Appl. 6(1), 67–113 (2002)
21. Jünger, M., Schulz, M.: Intersection graphs in simultaneous embedding with fixed edges. Journal of Graph Algorithms and Applications 13(2), 205–218 (2009)
22. Lempel, A., Even, S., Cederbaum, I.: An algorithm for planarity testing of graphs. In: Rosenstiehl, P. (ed.) Theory of Graphs: International Symposium, pp. 215–232 (1967)
23. Mohar, B., Thomassen, C.: Graphs on Surfaces. Johns Hopkins University Press, Baltimore (2001)
24. Nishizeki, T., Chiba, N.: Planar graphs: theory and algorithms. Elsevier, Amsterdam (1988)
25. Nishizeki, T., Rahman, M.S.: Planar graph drawing. World Scientific, Singapore (2004)
26. Pach, J., Wenger, R.: Embedding planar graphs at fixed vertex locations. Graphs and Combinatorics 17(4), 717–728 (2001)
27. Shih, W.K., Hsu, W.-L.: A new planarity test. Theoretical Computer Science 223(1-2), 179–191 (1999)