Subgraph Induced Planar Connectivity Augmentation
(Extended Abstract)

Carsten Gutwenger[2], Michael Jünger[1], Sebastian Leipert[2], Petra Mutzel[3], Merijam Percan[1], and René Weiskircher[3]

[1] Universität zu Köln, Institut für Informatik,
Pohligstraße 1, 50969 Köln, Germany
{mjuenger,percan}@informatik.uni-koeln.de
Partially supported by the Future and Emerging Technologies programme of the EU under contract number IST-1999-14186 (ALCOM-FT).
[2] caesar research center,
Ludwig-Erhard-Allee 2, 53175 Bonn, Germany
{gutwenger,leipert}@caesar.de
[3] Technische Universität Wien E186, Favoritenstraße 9–11, 1040 Wien, Austria,
{mutzel,weiskircher}@ads.tuwien.ac.at

Abstract. Given a planar graph $G = (V, E)$ and a vertex set $W \subseteq V$, the subgraph induced planar connectivity augmentation problem asks for a minimum cardinality set F of additional edges with end vertices in W such that $G' = (V, E \cup F)$ is planar and the subgraph of G' induced by W is connected. The problem arises in automatic graph drawing in the context of c-planarity testing of clustered graphs. We describe a linear time algorithm based on SPQR-trees that tests if a subgraph induced planar connectivity augmentation exists and, if so, constructs a minimum cardinality augmenting edge set.

1 Introduction

For an undirected graph $G = (V, E)$, a subset of vertices W of V, and E_W the subset of E that contains only edges with end vertices in W let $G_W = (W, E_W)$ be the subgraph of G induced by W. If G is planar, a *subgraph induced planar connectivity augmentation* for W is a set F of additional edges with end vertices in W such that the graph $G' = (V, E \cup F)$ is planar and the graph G'_W is connected.

We present a linear time algorithm based on the SPQR data structure that tests if a subgraph induced planar connectivity augmentation exists and, if so, constructs a minimum cardinality augmenting edge set. The difficulty of the subgraph induced planar connectivity augmentation problem arises from the fact that the computation of an appropriate planar embedding is part of the problem. Once the embedding is fixed, the decision becomes trivial.

The subgraph induced planar connectivity augmentation problem arises for example in the context of c-planarity testing of clustered graphs. A *clustered* *graph* is an undirected graph together with a nested family of vertex subsets whose members are called *clusters*. The concept of *c*-planarity is a natural extension of graph planarity for clustered graphs and plays an important role in automatic graph drawing. While the complexity status of the general problem is unknown, *c*-planarity can be tested in linear time if the graph is *c*-connected, i.e., all cluster induced subgraphs are connected [3,2]. In approaching the general case, it appears natural to augment the clustered graph by additional edges in order to achieve *c*-connectivity without losing *c*-planarity.

The results presented in this paper are the basis for a first step towards this goal. Namely, the algorithm presented here leads to a linear time algorithm that tests "almost" *c*-connected cluster graphs, i.e., cluster graphs in which at most one cluster is disconnected, for *c*-planarity [6,8].

In general, connectivity augmentation problems consist of adding a set of edges in order to satisfy certain connectivity constraints such as k-connectivity or k-edge-connectivity. The first results in this area are due to Lovász [12]. Since then, augmentation results for many different connectivity properties have been proved. For more information on connectivity augmentation problems see the surveys, e.g., by Frank [5] and Khuller [11].

Planar connectivity augmentation problems have been introduced by Kant and Bodlaender [10]. They have shown that the problem of augmenting a planar graph to a planar biconnected graph with the minimum number of edges is NP-hard. Moreover, they have given a 2-approximation algorithm for the planar biconnectivity augmentation problem. Fialko and Mutzel [4] have given a $\frac{5}{3}$-approximation algorithm. Polyhedral investigations of this problem have been conducted, e.g., in [13] and [14].

To our knowledge, this is the first time that subgraph induced planar connectivity augmentation is investigated. It is a generalization of planar connectivity augmentation. The subgraph induced connectivity augmentation problem is a special case of a certain connectivity augmentation problem considered, e.g., by Stoer [15] in the context of network survivability.

This paper is organized as follows. Section 3 describes the easy case in which the embedding of the given graph is fixed for example, when the graph is triconnected. The most interesting case is the treatment of biconnected graphs in Section 4 with the help of the SPQR-tree data structure and the results of the previous section. Finally, Section 5 is concerned with the non-biconnected case in which we construct the block-cut tree of G and use the algorithm for biconnected graphs for each block along with a planarity testing step. All sections are illustrated by an application to an example graph (see Figure 2 - Figure 5). The proofs omitted in this extended abstract and the information about the implementation will be given in the full version [7].

2 Preliminaries

SPQR-trees have been introduced by Di Battista and Tamassia [1]. They represent a decomposition of a planar biconnected graph according to its split pairs

(pairs of vertices whose removal splits the graph or vertices connected by an edge). The construction of the SPQR-tree works recursively (see Figure 2). At every node v of the tree, we split the graph into the split components of the split pair associated with that node. The first split pair of the decomposition is an edge of the graph and is called the *reference edge* of the SPQR-tree. We add an edge to each of the split components to make sure that they are biconnected and continue by computing their SPQR-tree and making the resulting trees the subtrees of the node used for the splitting. Every node of the SPQR-tree has two associated graphs:

- The *skeleton* of the node associated with a split pair p is a simplified version of the whole graph where some split-components are replaced by single edges.
- The *pertinent graph* of a node v is the subgraph of the original graph that is represented by the subtree rooted at v.

The two vertices of the split pair that are associated with a node v are called the *poles* of v. There are four different node types in an SPQR-tree (S-,P-,Q- and R-nodes) that differ in the number and structure of the split components of the split pair associated with the node (see Figure 1). The Q-nodes form the leaves of the tree, and there is one Q-node for each edge in the graph. The skeleton of a Q-node consists of the poles connected by two edges. The skeletons of S-nodes are cycles, while the skeletons of R-nodes are triconnected graphs. P-node skeletons consist of the poles connected by at least three edges. Figure 1 shows examples for skeletons of S-, P- and R-nodes.

Skeletons of adjacent nodes in the SPQR-tree share a pair of vertices. In each of the two skeletons, one edge connecting the two vertices is associated with a corresponding edge in the other skeleton. These two edges are called *twin edges.* The edge in a skeleton that has a twin edge in the parent node is called the *virtual edge* of the skeleton.

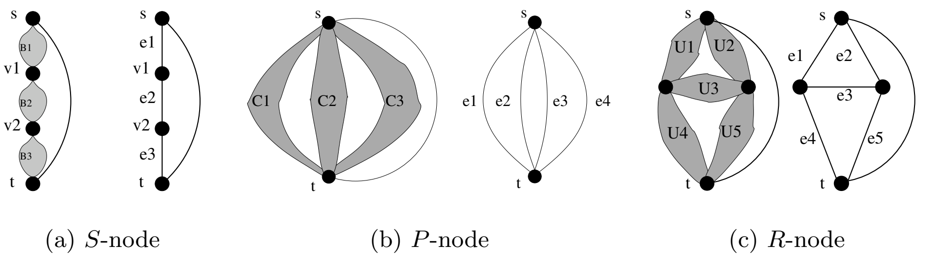

Fig. 1. Decomposition of biconnected graphs and the skeletons of the corresponding nodes in the SPQR-tree. Here the three cases are illustrated.

Each edge e in a skeleton represents a subgraph of the original graph. This graph together with e is the *expansion graph* of e.

All leaves of the SPQR-tree are Q-nodes and all inner nodes S-, P- or R-nodes. When we regard the SPQR-tree as an unrooted tree, then it is unique for every biconnected planar graph. Another important property of these trees is that their size (including the skeletons) is linear in the size of the original graph and that they can be constructed in linear time [1,9]. As described in [1, 9], SPQR-trees can be used to represent the set of all combinatorial embeddings of a biconnected planar graph. Every combinatorial embedding of the original graph defines a unique combinatorial embedding for a skeleton of each node in the SPQR-tree. Conversely, when we define an embedding for the skeleton of each node in the SPQR-tree, we define a unique embedding for the original graph. The skeletons of S- and Q-nodes are simple cycles, so they have only one embedding. But the skeletons of R-and P-nodes have at least two different embeddings. Therefore, the embeddings of the R- and P-nodes determine the embedding of the graph and we call these nodes the *decision nodes* of the SPQR-tree.

3 An Easy Case: Fixed Embedding

We consider the case that the planar graph G is given together with a fixed embedding. In the following, we call the vertices belonging to W *blue vertices*. Our task is to insert edges so that the induced subgraph G_W is connected and G is still planar after edge insertion.

Our algorithm looks at each face f in G that has at least two non-adjacent blue vertices on its boundary. We start at an arbitrary blue vertex v on the boundary of f and introduce a new edge through f that connects it to the next blue vertex on the boundary. Thus we step through the blue vertices on the boundary of f, connecting each to its successor on the boundary until we come back to v. We call the resulting graph G'. Note that we only introduce a linear number of edges in this step and that G' is planar. Another important property of G' is that G'_W is connected if and only if there is a planar augmentation for W in G.

Then we compute the graph G'' by deleting all vertices from G' that are not blue. We assign value 1 to all edges introduced in the first step and 0 to all other edges. We can find a minimum spanning tree in G'' in linear time because there are only two different weights on the edges. One way to do this is to use Prim's Algorithm where we use two lists instead of the priority queue. The algorithm may now determine that G'' is not connected. Then we know that there is no planar augmentation for G. Otherwise, the edges of weight 1 in the minimum spanning tree are our solution. Thus, we can solve the problem in linear time.

4 The Algorithm for the Biconnected Case

In this section we present an algorithm for the case that the given graph G is biconnected. First, we compute the SPQR-tree $\mathcal{T}$ of G. In Section 4.1, we present a recursive algorithm for coloring all edges in all skeletons of the SPQR-tree. This coloring stores information about the position of the blue vertices in each skeleton of the SPQR-tree by assigning three different colors to the edges. The coloring enables us to test if an augmentation is possible by examining the colors in each skeleton. If an augmentation is possible, we compute an embedding of the graph that allows an augmentation (see Section 4.2). Then we can apply the algorithm of the previous section to this fixed embedding in order to compute the list of edges needed to solve the augmentation problem. Algorithm 1 gives an overview of the algorithm for biconnected graphs.

Algorithm 1: The algorithm `BiconnectedAugmenter` computes a planar connectivity augmentation for a planar biconnected graph G and a subset W of the vertices, if it exists.

```
Input: A biconnected planar graph $G$ and a subset $W$ of its vertices,
Result : true if and only if there is a planar augmentation for $W$; in the positive
         case an embedding $\Pi$ and a minimum cardinality augmenting edge
         set will be computed.
Calculate the SPQR-tree $\mathcal{T}$ of $G$;
Make an arbitrary node $r$ which is not a $Q$-node the root of $\mathcal{T}$;
MarkEdgesPhase1($r, W$);
MarkEdgesPhase2($r, W$);
BiconnectivityFeasibilityCheck($\mathcal{T}$);
Embedding $\Pi$ = CalculateEmbedding;
return FixedEmbeddingAugmenter($\Pi, W$);
```

4.1 The Coloring Algorithm

Again we call a vertex blue, if it is contained in W and black otherwise. Figure 2 shows an example graph, in which the set W is given by $\{3, 8, 10\}$ (shown by bold circles). The skeletons are shown in Figure 3, which displays essentially the SPQR-tree without the Q-nodes. The blue vertices in the tree are marked by circles.

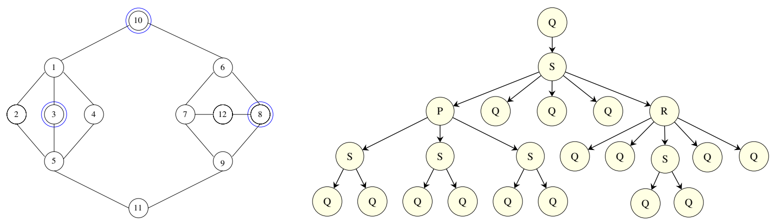

Fig. 2. Example: A graph G and its SPQR-tree.

We assign one of two colors to each edge in each skeleton: blue or black. We call an edge in a skeleton blue, if its expansion graph contains blue vertices and black otherwise. For example, in Figure 5 in the skeleton of the S-node of the SPQR-tree, the edge between the vertices 1 and 5 is blue because the blue vertex 3 is contained in its expansion graph. It is represented by a dashed line.

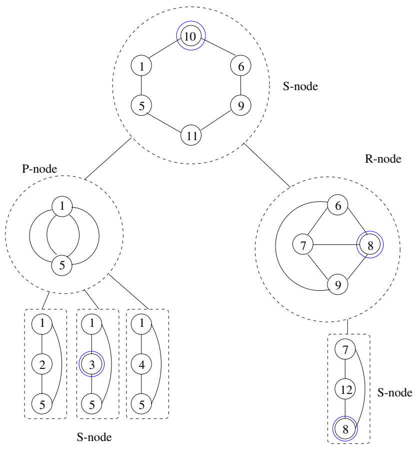

Fig. 3. Continuation of Figure 2: Q-nodes are omitted; assigning blue to the vertices of the subset W.

Additionally, we assign the attribute *permeable* to some blue edges. Intuitively, an edge is permeable if we can construct a path connecting only blue vertices through its expansion graph. Therefore, in Figure 5 the edge between the vertices 1 and 5 in the skeleton of the root node does not get the attribute permeable. But we assign for example the attribute permeable to the edge between the vertices 1 and 5 in the skeleton of the P-node whose expansion graph contains the vertex 10. In Figure 5, the permeable edges are represented by a dotted line. Let $G(e)$ be the expansion graph of edge e in skeleton $\mathcal{S}$. In any planar embedding $G(e)$, there are exactly two faces that have e on their boundary. This follows from the fact that in a planar biconnected graph, every edge is on the boundary of exactly two faces in every embedding. We call the edge e in $\mathcal{S}$ permeable with respect to W, if there is an embedding Π of $G(e)$ and a list of at least two faces $L = (f_1, \ldots, f_k)$ in Π that satisfies the following properties:

1. The two faces f_1 and f_k are the two faces with e on their boundary.
2. For any two faces f_i, f_{i+1} with $1 \leq i < k$, there is a blue vertex on the boundary between f_i and f_{i+1}.

We call a skeleton $\mathcal{S}$ of a node v of $\mathcal{T}$ permeable if the pertinent graph of v together with the virtual edge of $\mathcal{S}$ have the two properties stated above. So $\mathcal{S}$

is permeable if the twin edge of its virtual edge is permeable. For example, in Figure 5 the skeleton that contains the vertex 3 is permeable.

We develop an algorithm that marks each edge in every skeleton of the SPQR-tree $\mathcal{T}$ of G with the colors black or blue and assigns the attribute permeable depending on the expansion graph of the edge. The algorithm works recursively. We assume that $\mathcal{T}$ is rooted at node r and r is not a Q-node.

First we mark all the edges of the skeletons of the children of r recursively black or blue and assign the permeable attribute by treating them as the roots of subtrees. Each edge in the skeleton $\mathcal{S}$ of r except the reference edge corresponds to a child of r. Let e be such an edge in $\mathcal{S}$, v the corresponding child of r and $\mathcal{S}'$ the skeleton of v. If $\mathcal{S}'$ contains a blue edge or vertex, we mark e blue and otherwise black. The permeability of e depends on the type of v:

Q-node: We mark e permeable if the skeleton $\mathcal{S}'$ contains a blue vertex.

S-node: If the skeleton $\mathcal{S}'$ contains a blue vertex or a permeable edge, we mark e permeable.

P-node: If the skeleton $\mathcal{S}'$ contains only permeable edges or a blue vertex, we mark e permeable.

R-node: We consider a graph H where the vertices are the faces of $\mathcal{S}'$ and there is an edge between two vertices if there is a permeable edge or a blue vertex on the boundary that separates the two faces. Let s and t be the two faces left and right of the virtual edge of $\mathcal{S}'$. If there is a path in H connecting s and t, we mark e permeable (see Figure 4).

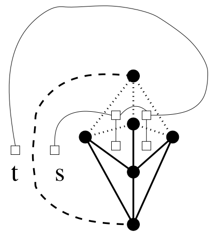

Fig. 4. A permeable R-node with the graph H: permeable edges are represented by dotted lines, the virtual edge of the skeleton by a dashed line.

After executing this algorithm, which we call `MarkEdgesPhase1`, all edges of the skeleton of the root node r are marked, because we have seen all the blue vertices of the graph. All other skeletons except the skeleton of r contain one edge that is not yet marked: the virtual edge of the skeleton.

The algorithm `MarkEdgesPhase2` works top down by traversing $\mathcal{T}$ from the root to the leaves. The edges of the skeleton of the root r of $\mathcal{T}$ are already marked in the first step, therefore we can proceed to the children and mark the virtual edges of its children. Let v be a node in $\mathcal{T}$ where the skeleton $\mathcal{S}'$

of the parent node is already completely marked. We mark the virtual edge e in the skeleton $\mathcal{S}$ of v blue if there is a blue edge or vertex in the skeleton of the parent. The permeability of e again depends on the type of the skeleton in exactly the same way as in the algorithm `MarkEdgesPhase1` (see Figure 5). Note that the case Q-node is irrelevant here because the Q-nodes form the leaves of the tree. In Figure 5 we see a result of the two algorithms `MarkEdgesPhase1` and `MarkEdgesPhase2` for our example graph. Permeable edges are represented by dotted lines, blue edges that are not permeable by dashed lines and blue vertices by a circle around them.

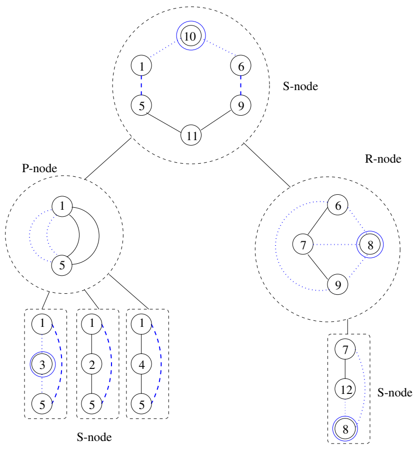

Fig. 5. Continuation of Figure 3: After calling the algorithms `MarkEdgesPhase1` and `MarkEdgesPhase2`.

The two algorithms `MarkEdgesPhase1` and `MarkEdgesPhase2` can both be implemented in linear time because the size of the SPQR-tree of a planar biconnected graph including all skeletons is linear in the size of the graph [1].

Lemma 1. *Let e be an edge in a skeleton of an inner node and $G(e)$ its expansion graph. Then the coloring algorithm marks e blue if and only if $G(e)$ contains a vertex of W. Furthermore, e is marked permeable if an only if there is an embedding Π of $G(e)$ together with a sequence of faces $f_1, \dots, f_k$ with the following property:*

(*) The two faces f_1 and f_k are the two faces with e on their boundary and for any two faces f_i, f_{i+1} with $1 \leq i < k$, there is a blue vertex on the boundary between f_i and f_{i+1}.

4.2 The Embedding Algorithm

Let $\mathcal{S}$ be a skeleton of a P-node. We call the embedding of $\mathcal{S}$ *admissible* if all blue edges are consecutive and the blue edges that are not permeable (if they exist) are at the beginning and at the end of the sequence (see Figure 6 for three examples of admissible orderings). For example, in Figure 5 the skeleton of the P-node has already an admissible embedding.

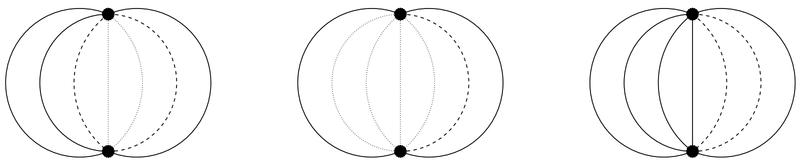

Fig. 6. Admissible embeddings of a P-node skeleton (permeable edges are dotted, blue edges that are not permeable are dashed).

Our algorithm for finding an augmentation or proving that no augmentation exists works in two phases:

1. Using the colors and attributes of the edges in each skeleton, we fix an embedding for every P- and R-node skeleton and thus determine an embedding for G.
2. We use the algorithm of Section 3 for fixed embeddings to determine whether an augmentation is possible.

The embedding computed in the first step has the property that it allows an augmentation if and only if there is an embedding of G that allows an augmentation.

We set the embedding for the skeletons of the R- and P-nodes recursively using the structure of the SPQR-tree. We assume that the vertices in G are numbered and that all edges are directed from the vertex with lower number to the vertex with higher number.

For simplicity, we consider whether a special case is present where a planar connectivity augmentation cannot exist.

Theorem 1. *Let G be a biconnected series-parallel planar graph and W a subset of its vertices. There exists a planar connectivity augmentation for W in G if and only if all P-nodes of the SPQR-tree of G contain at the most two edges that are blue but not permeable.*

The conclusion of the theorem is that if there exists a P-node that contains a skeleton with more than two blue edges in a biconnected graph G, there cannot exist a planar connectivity augmentation. For example, in Figure 5 there are no edges that are blue but not permeable in the skeleton of the P-node.

First, we test whether the biconnected graph contains only P-nodes with skeletons that have at the most two edges. Then we can sort the two blue edges as mentioned in Figure 6 to obtain an admissible embedding. Note that the two blue but not permeable edges are not consecutive in an admissible embedding if there are additionally black and permeable edges. Therefore, there is nothing to do for the skeleton of the P-node in Figure 5.

Next, we construct an algorithm that marks edges in the skeletons with a new attribute that can have three different values: `left`, `right` and `nil`. If the virtual edge (the edge whose twin edge is in the parent of v) in a skeleton of node v is marked `left`, then the pertinent graph of v must be embedded in such a way that there is a blue vertex on the boundary of the face left of the virtual edge. If the edge is marked `right`, a blue vertex must be on the boundary of the face right of the virtual edge. If the virtual edge is marked `nil`, there is no restriction on the embedding of the pertinent graph of v.

For each node v in the SPQR-tree, where the embedding of the parent node has already been fixed, we perform two steps:

1. We determine an embedding using the attribute of the virtual edge and the colors and attributes of the other edges.
2. We determine the attribute for the virtual edge in the skeleton of each child of v.

Only the skeletons of R- and P-nodes have more than one embedding, so the first step is only important for these node types. First we consider the case where v is an R-node. In this case, its skeleton has two embeddings. If the attribute on the virtual edge is `nil`, we can choose any of the two embeddings. Otherwise, we choose an embedding where there is a blue edge or vertex on the face left (right) of the virtual edge if the attribute was `left` (`right`). If none of the two embeddings has this property, no augmentation is possible. If v is a P-node, we can only choose among the admissible embeddings of the skeleton. Again, we choose an embedding according to the attribute and if no suitable embedding exists, there can be no augmentation.

Now we have to determine the attribute for the virtual edge of each child of v. Let $\mathcal{S}$ be the skeleton of v. For all edges in $\mathcal{S}$ that are either black or permeable, we pass `nil` to the corresponding child. Let L be the set of edges in $\mathcal{S}$ that are blue but not permeable.

First we consider the case that v is an S-node. If the attribute of the virtual edge is `nil`, we pass `nil` to every child that corresponds to an edge in L. Otherwise, the attribute on the virtual edge designates one of the two faces of the S-node skeleton as the one that must have a blue vertex on the boundary. For each edge e in L, we pass `left` to the corresponding child if this face is right of e and `right` otherwise.

To make the following descriptions more concise, we call a face *permeable*, if it has a permeable edge or a blue vertex on its boundary. If v is a P-node, L contains at most two edges. For each of edge e in L, there are three possible cases:

1. Exactly one of the faces with e on its boundary contains a blue edge. If this is the face on the right of e, we pass `left` to the child corresponding to e and `right` otherwise.

2. One of the faces with e on its boundary is permeable and the other is not. If the permeable face is left of e, we pass `right` to the child corresponding to e and `left` otherwise.
3. The faces left and right of e are both permeable or both contain only black edges and vertices. In this case, we pass `nil` to the child.

If v is an R-node, we also have to consider the faces left and right of each edge e in L. The same cases as for P-nodes apply, but there is one additional case that cannot occur in a P-node: Both faces left and right of e are not permeable but both contain a blue edge except e (if this happens in a P-node, there can be no augmentation). In this case, we pass `nil` to the corresponding child. For example, in Figure 5 the attributes of all virtual edges are `nil`.

To start the process, we choose an arbitrary P- or R-node as the root of the SPQR-tree. If we choose an arbitrarily R-node, we select one of the two embeddings of the skeleton. If we select a P-node, we choose an arbitrary admissible embedding. Now we can compute the attributes we pass to the children of the node as stated above and compute an embedding for each skeleton of the SPQR-tree by applying the algorithm in depth first or breadth first sequence to all inner nodes of the tree.

This algorithm defines an embedding for each R- and P-node in the SPQR-tree and thus for the graph G. Since we touch each skeleton only once and the operations we perform for each skeleton can be done in time linear in the size of the skeleton, the embedding can be computed in linear time. Then we apply the algorithm from Section 3 to the fixed embedding. This algorithm either computes the list of edges that constitutes the planar connectivity augmentation or it signals that no augmentation is possible for this embedding.

Theorem 2. *Let Π be the embedding of G determined by the algorithm described above. G has a planar connectivity augmentation with respect to W if and only if there exists a planar connectivity augmentation for G with respect to the embedding Π.*

The proof uses structural induction over the SPQR-tree. We first show that the claim holds for graphs whose SPQR-tree has only one inner node and then use induction to show that it holds for graphs whose SPQR-tree has more than one inner node. Because of space considerations we show the proof in [7].

Theorem 3. *Given a biconnected planar graph $G = (V, E)$ and a subset of vertices $W \subseteq V$. The algorithm* `BiconnectedAugmenter` *tests correctly whether a subgraph induced planar connectivity augmentation exists and, if so, constructs a minimum cardinality augmenting edge set. It runs in time $O(|V|)$.*

5 The Algorithm for the Connected Case

Using BC-trees, this algorithm can be extended for connected graphs. Because of space considerations we omit a detailed description that is available in [7]. Using this result for non-connected planar graphs, the problem becomes easy.

References

1. Di Battista, G., Tamassia, R. (1996) On-line planarity testing. Journal on Computing **25** (5), 956–997
2. Cohen, R.-F., Eades, P., Feng, Q.-W. (1995) Planarity for clustered graphs. In P. Spirakis (ed.) Algorithms – ESA '95, Third Annual European Symposium, Lecture Notes in Computer Science 979, Springer-Verlag, 213–226
3. Dahlhaus, E. (1998) Linear time algorithm to recognize clustered planar graphs and its parallelization (extended abstract). In C. L. Lucchesi (ed.) LATIN '98, 3rd Latin American symposium on theoretical informatics, Campinas, Brazil, Lecture Notes in Computer Science 1380, 239–248
4. Fialko, S., Mutzel, P. (1998) A new approximation algorithm for the planar augmentation problem. In Proceedings of the Ninth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA '98), San Francisco, California, ACM Press, 260–269
5. Frank, A. (1992) Augmenting graphs to meet edge-connectivity requirements. SIAM J. Discrete Mathematics **5** (1), 25–53
6. Gutwenger, C., Jünger, M., Leipert, S., Mutzel, P., Percan, M., Weiskircher, R. (2002) Advances in c-planarity testing of clustered graphs. Technical report, Institut für Informatik, Universität zu Köln, zaik2002–436.
7. Gutwenger, C., Jünger, M., Leipert, S., Mutzel, P., Percan, M., Weiskircher, R. (2002) Subgraph induced planar connectivity augmentation. Technical report, Institut für Informatik, Universität zu Köln, zaik2002–435.
8. Gutwenger, C., Jünger, M., Leipert, S., Mutzel, P., Percan, M., Weiskircher, R. (2003) Advances in c-planarity testing of clustered graphs. In M. T. Goodrich and S. G. Kobourov (eds.) Graph Drawing (Proc. 2002), Lecture Notes in Computer Science 2528, Springer-Verlag, 220–235
9. Gutwenger, C., Mutzel, P. (2001) A linear time implementation of SPQR-trees. In J. Marks (ed.) Graph Drawing (Proc. 2000), Lecture Notes in Computer Science 1984, Springer-Verlag, 77–90
10. Kant, G., Bodlaender, H. L. (1991) Planar graph augmentation problems. In Proc. 2nd Workshop Algorithms Data Struct., Lecture Notes in Computer Science 519, Springer-Verlag, 286–298
11. Khuller, S. (1997) Approximation Algorithms for Finding Highly Connected Subgraphs. In Hochbaum, D. S. (ed.), PWS Publishing Company, Boston, 236–265
12. Lovász, L. (1979) Combinatorial Problems and Exercises. North-Holland.
13. Mutzel, P. (1995) A polyhedral approach to planar augmentation and related problems. In P. Spirakis (ed.) Algorithms – ESA '95, Third Annual European Symposium, Lecture Notes in Computer Science 979, Springer-Verlag, 494–507
14. De Simone, C., Jünger, M. (1997) On the two-connected planar spanning subgraph polytope. Discrete Applied Mathematics **80** (229), 223–229
15. Stoer, M. (1992) Design of Survivable Networks, Lecture Notes in Mathematics 1531. Springer-Verlag, Berlin