Alcom-FT Technical Report Series
ALCOMFT-TR-02-126

# 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,
Friedensplatz 16, 53111 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 an minimum cardinality augmenting edge set.

## 1 Introduction

For an undirected graph $G = (V, E)$, $W \subseteq V$, and $E_W = \{(v_1, v_2) \in E : \{v_1, v_2\} \subseteq W\}$ 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 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*. $c$-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 [?,?]. In approaching the general case, it appears natural to augment the clustered graph by additional edges in order to achieve $c$-connectivity without loosing $c$-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 [?].

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 on this area are due to Lovasz [?]. 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 [?] and Khuller [?].

Planar connectivity augmentation problems have been introduced by Kant and Bodlaender [?]. 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 [?] have given a $\frac{5}{3}$-approximation algorithm. Polyhedral investigations of this problem have been conducted, e.g., in [?] and [?].

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 [?] 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, e.g., 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 resullts 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. The proofs omitted in this extended abstract will be given in the full version [?].

## 2 Preliminaries

SPQR-trees have been suggested by Di Battista and Tamassia [?]. 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. 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 them 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:

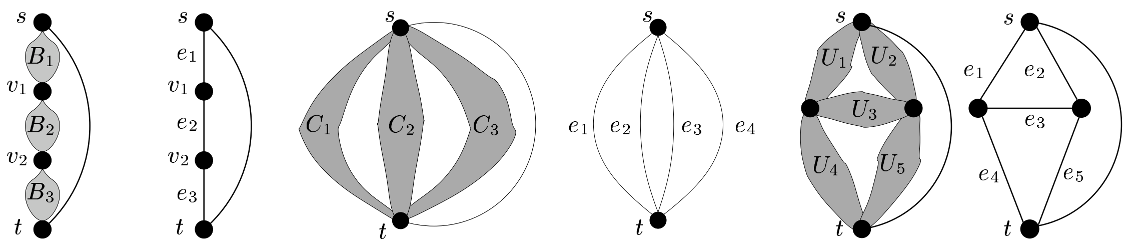
- The *skeleton* of the node associated with a split pair $p$ is a simplified version of the whole graph where some split-components are replaced by single edges.
- The *pertinent graph* of a node $v$ is the subgraph of the original graph that is represented by the subtree rooted at $v$.

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. The $Q$-nodes form the leaves of the tree, and their 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.

**Definition 1 (Expansion Graph).** *Let $e = (u, v)$ be an edge in a skeleton $\mathcal{S}$ of a node $\mu$ of the SPQR-tree $\mathcal{T}$ of $G$. Since $e$ is an edge in a skeleton, the vertices $u$ and $v$ are a split-pair of $G$. Then we define the* expansion graph *of $e$ as follows:*

1. *If $\mu$ is the $Q$-node for edge $e'$ in $G$, then the expansion graph $G(e)$ of $e$ is defined as follows: if $e$ is the virtual edge of $\mu$ then we define $G(e)$ as $G(e) = (V, (E - \{e'\}) \cup \{e\})$. Otherwise, we define $G(e)$ as $G(e) = (\{u, v\}, \{e, e'\})$. Therefore the expansion graph is either isomorphic to $G$ or to $\mathcal{S}$.*
2. *If $\mu$ is an $R$-node or $S$-node, then the expansion graph of $e$ is the union of all the split components of the split pair $\{u, v\}$ in $G$ that contain no vertices of $\mathcal{S}$ except $u$ and $v$ together with edge $e$.*
3. *If $\mu$ is a $P$-node, then there are at least three split components of the pair $\{u, v\}$ in $G$. In the construction of $\mathcal{T}$, all edges $e_i$ of $\mathcal{S}$ except the virtual edge are associated with a subgraph $G_i$ of $G$. Thus we define the expansion graph of each $e_i$ as the graph $G_i$ together with edge $e$. The expansion graph of the virtual edge is defined as the split component of $\{u, v\}$ that contains the reference edge of $\mathcal{T}$ (the edge of $G$ that is used to start the decomposition) together with edge $e$.*




(a) Case $S$-node (b) Case $P$-node (c) Case $R$-node

**Fig. 1.** The structure of biconnected graphs and the skeleton of the root of the corresponding SPQR-tree.

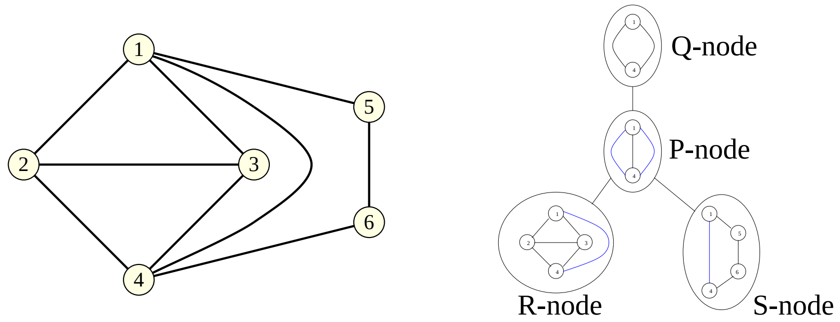



**Fig. 2.** A graph $G$ and its SPQR-tree (the $Q$-nodes of the $R$- and $S$-node are omitted).

All leaves of the SPQR-tree are $Q$-nodes and all inner nodes $S$-, $P$- or $R$-nodes. When we see 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 [?] [?]. As described in [?] [?], 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 each skeleton of a node in the SPQR-tree. Conversely, when we define an embedding for each skeleton of a node in the SPQR-tree, we define a unique embedding for the original graph. The skeleton of $S$- and $Q$-nodes are simple cycles, so they have only one embedding. But the skeletons of $R$-and $Q$-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 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 his successor on the boundary until we come back to $v$. We call the resulting graph $G'$. Note that we only introduced 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. All edges introduced in the first step receive value 1 and all others value 0. 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 suggest an algorithm in the case that the given graph $G$ is biconnected. Firts, 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. This testing algorithm is explained in Section 4.2. If an augmentation is possible, we compute an embedding of the graph that allows an augmentation (see Section 4.3). Then we can apply the algorithm of the previous section to this fixed embedding in order to compute the list of edges we have to solve the augmentation problem (see Algorithm 1 for 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 the minimum cardinality augmenting edge set will be returned.
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. 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.

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. 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 $\Pi$ of $G(e)$ and a list of at least two faces $L = (f_1, \dots, f_k)$ in $\Pi$ 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 $v$ and 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.

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 assign the attribute permeable depending on the expansion graph of the edge. The algorithms works recursively. We assume that $\mathcal{T}$ is rooted at node $r$.

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 $\mathcal{S}'$ 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 $\mathcal{S}'$ 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 of 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 3).

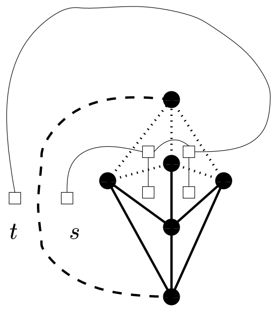



**Fig. 3.** A permeable $R$-node: permeable edges are represented by dotted lines, the virtual edge of the skeleton by a dashed line

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`. Note that the case $Q$-node is irrelevant here because the $Q$-nodes form the leaves of the tree.

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 [**?**].

**Lemma 1.** *Let $G$ be a planar biconnected graph and $W$ a subset of its vertices. Let $e$ be an edge in the skeleton $\mathcal{S}$ of the node $v$ in the SPQR-tree $\mathcal{T}$ of $G$ and $G(e)$ the expansion graph of $e$. After applying the coloring algorithms* `MarkEdgesPhase1` *and* `MarkEdgesPhase2`*, the following statements are true:*

1. *The edge $e$ is colored blue if and only if $G(e)$ contains a vertex of $W$.*
2. *The edge $e$ is marked permeable if and only if it is permeable.*

### 4.2 Checking Feasibility

The goal of this step is to use the coloring of the edges in the skeletons of the SPQR-tree to determine, if a planar connectivity augmentation of $G$ for the subset $W$ of vertices is possible and to compute the list of

**Algorithm 2:** Algorithm `MarkEdgesPhase1` that marks all edges in the skeletons except the virtual edges.

```
Input: An node v in 𝒯 and the set W of vertices
Result : All edges in S(v) except the virtual are marked.
Let L be the set of children of v;
foreach v' ∈ L do
    MarkEdgesPhase1(v', W);
    Let e be the edge that S(v') shares with S(v);
    if S(v') contains a blue or permeable edge or a vertex of W then
        Mark e blue
    else
        Mark e black;
    switch type of node v' do
        case P-node
            if All edges in S(v') except e are permeable or one of the vertices in S(v') belongs to W then
                Mark e as permeable
        case S-node
            if S(v') contains a vertex of W or an edge marked permeable then
                Mark e permeable
        case R-node
            Let e be the edge shared by S(v') and S(v);
            if findPath(e, v') returns true then
                Mark e as permeable
if v is a Q-node then
    Let e be the non-virtual edge of S(v);
    if S(v) contains a vertex of W then
        Mark e permeable;
    else
        Mark e black;
```

**Algorithm 3:** Algorithm `MarkEdgesPhase2` marks all the virtual edges of the skeletons in the subtree rooted at $v$ if all the edges in the skeleton of $v$ are marked.

**Input**: A node $v$ in the SPQR-tree $\mathcal{T}$ where all edges are marked
**Result** : All edges in the subtree rooted at $v$ are marked

```
Let S be the skeleton of v;
if v is a Q-node then
    Let v1 and v2 be the two vertices in S;
    Let e be the virtual edge in the child of v;
    if {v1, v2} ∩ W ≠ ∅ then
        Mark e permeable;
    else
        Mark e black;
else
    Let L be the list of edges in S whose twin edge is contained in a skeleton of a child of v;
    if S does not contain a blue or permeable edge or a vertex of W then
        Mark all the twin edges of the edges in L black;
    else
        Mark all the twin edges of the edges in L blue;
        switch type of v do
            case S-node
                if S contains a vertex of W or an edge marked permeable then
                    Mark every twin edge of the edges in L permeable;
            case P-node
                if All edges in S are marked permeable then
                    Mark all twin edges of the edges in L permeable
            case R-node
                foreach edge e in L do
                    Let v' be the child of v that contains e;
                    if findPath(e, v') returns true then
                        Mark e in S(v') as permeable;
forall the children w of v do
    MarkEdgesPhase2(w, W);
```

**Algorithm 4:** Algorithm `findPath` checks if the expansion graph of the twin edge of an edge in an $R$-node skeleton is permeable.

**Input**: An edge $e$ and an $R$-node $v$ in $\mathcal{T}$ with the property that $e$ is contained in the skeleton $\mathcal{S}$ of $v$ and all edges in $\mathcal{S}$ are marked
**Result** : "true" if the expansion graph of the twin edge of $e$ is permeable

```
Compute an arbitrary embedding Π of S;
Compute the dual graph D of S with respect to Π;
foreach edge e' of D do
    if the primal edge of e' is permeable then
        Set the cost of e' to zero;
    else
        Set the cost of e' to one;
Set the cost of the dual edge of e to one;
Let v1 and v2 be the two vertices in D that correspond to the two faces in Π with e on their boundary;
Compute the shortest path p from v1 to v2 in D;
if the cost of p is zero then
    return "true";
else
    return "false";
```

necessary edges. First we present a lemma that holds for series-parallel biconnected planar graphs. These are the biconnected planar graphs whose SPQR-tree does not contain an $R$-node.

**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.*

*Proof.* The SPQR-tree of series-parallel graphs contains no $R$-node. $P$-nodes with at least three blue and not permeable edges cannot be augmented (see Figure 4 showing a graph whose SPQR-tree contains a $P$-node with three blue but not permeable edges) and therefore no planar connectivity augmentation exists.

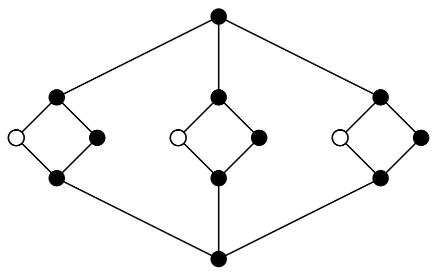


**Fig. 4.** A graph whose SPQR-tree contains a $P$-node with three blue edges that are not permeable (blue vertices are represented by hollow circles)

If every $P$-node skeleton contains at most two edges that are blue but not permeable, we can always find an embedding of the graph where we can connect the blue vertices. The embedding of $G$ is determined by the embedding of each $P$-node skeleton.

### 4.3 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 5 for three examples of admissible orderings).

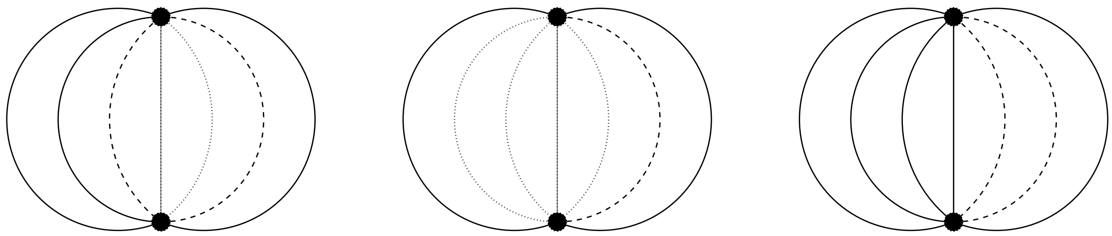


**Fig. 5.** Admissible embeddings of a $P$-node skeleton (permeable edges are dotted, blue edges that are not permeable are dashed)

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 from Section 3 for fixed embeddings to determine if 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. The algorithm 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 that $v$ 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 $P$-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 *preferred*, 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 preferred and the other is not. If the preferred 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 preferred 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 can not occur in a $P$-node: Both faces left and right of $e$ are not preferred 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.

To start the process, we choose an arbitrary $P$- or $R$-node as the root of the SPQR-tree. If we choose an $R$-node, we select one of the two embeddings of the skeleton at random. If we select a $P$-node, we choose an arbitrary admissible embedding at random. 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. The 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 $\Pi$ be the embedding of $G$ determined by the above described algorithm. $G$ 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 $\Pi$.*

This can be shown using 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.

**Theorem 3.** *Given a planar graph $G = (V, E)$ and a subset of vertices $W \subseteq V$. The algorithm* `Biconnected-Augmenter` *tests correctly if 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

In the last section we have dealt with the problem of finding an induced subgraph of $W$ in a biconnected graph. Hence we already know how to deal with the biconnected blocks of the graph $G$ using the SPQR-tree. To solve the connected case, we first build the BC-tree of $G$. This tree has two types of nodes: The $c$-nodes correspond to cut-vertices of $G$ and the $b$-nodes to biconnected components (blocks). There is an edge connecting a $c$-node and a $b$-node, if the cut-vertex is contained in the block.

We say a $b$-node is blue if it contains blue vertices. It is not hard to see that we only have to deal with the smallest subtree $bc$ of the BC-tree of $G$ that contains all blue blocks. We say a cut-vertex is red if it is in $bc$. Note that the leaves of a BC-tree are blocks. First we observe that there can only be a planar connectivity augmentation if there is an embedding of $G$ where all red vertices which are contained in non-blue blocks are on the boundary of the same face.

This can be seen as follows: If a blue block is connected to another blue block via a blue cut-vertex, we can use this cut-vertex to connect blue vertices in all neighboring faces. If the cut-vertex is not blue, this is not possible. Therefore, for every red cut-vertex in a blue block must be a blue vertex on the boundary of the same face in every embedding that allows a planar connectivity augmentation.

Figure 6 shows a graph where we cannot place all red vertices into the same face and no augmentation is possible. To test if we can embed the red vertices which are contained in a non-blue block into the same face, we introduce a new dummy vertex and connect it to these red vertices. If the resulting graph is planar, we know that we can put these red vertices into the same face. Otherwise, no augmentation is possible.

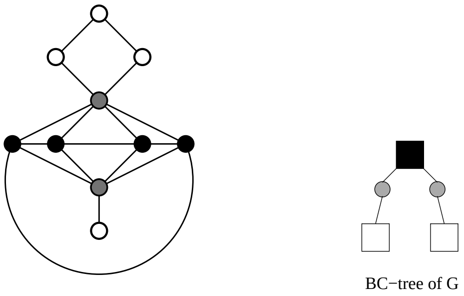



**Fig. 6.** No augmentation is possible because the two red cut-vertices can not be placed inside the same face (the blue vertices are represented by hollow circles and the red vertices by grey circles, the blue blocks as hollow squares)

To fix an embedding for $G$, we first use the same algorithm we have developed for the biconnected case. The only difference is that we treat each red vertex just like a blue vertex but do not give the label permeable to edges whose expansion graph contains only red vertices. In this way, we compute an embedding for each blue block. The embedding of blocks that are not blue is irrelevant. As described in the last section, we compute a planar connectivity augmentation for each block. Note that we also insert edges that connect a red vertex to a blue vertex.

We start traversing $bc$ from the leaves which are blocks with only one cut vertex. Beginning from the leaf $B_1$, if the cut vertex of $B_1$ is blue we only join the block $B_1$ with the parent node of the cut vertex in $bc$ which is block $B_2$ together in the way that we embed $B_1$ in a face of $B_2$ that contain the cut vertex and another blue node. If the cut vertex is red we embed one edge between the red vertex and a blue vertex in $B_1$ in the outer face. Then we embed block $B_1$ in a face of block $B_2$ which contain also an edge between the red vertex and a blue vertex. As there exists a planar connectivity augmentation in all blue blocks in $bc$ such an edge will exist. We proceed recursively until all blocks in $bc$ are inserted. Note that the non-blue blocks can be embedded arbitrarily.

Then we remove all the new edges where one end-vertex is a red vertex and use the algorithm from Section 3 to compute a planar connectivity augmentation for the fixed embedding.

For non-connected graphs we apply the algorithm of the connected case to each connected component. Then we transform the connected components so that they have at least one blue vertex on the outer face. Finally we connect the blue vertices on the outer face.