Linear Planar Augmentation Algorithms for Outerplanar Graphs

Goos Kant

RUU-CS-91-47
December 1991

Utrecht University

Department of Computer Science

Padualaan 14, P.O. Box 80.089,
3508 TB Utrecht, The Netherlands,
Tel. : ... + 31 - 30 - 531454

Linear Planar Augmentation Algorithms for Outerplanar Graphs

Goos Kant

Technical Report RUU-CS-91-47
December 1991

Department of Computer Science
Utrecht University
P.O.Box 80.089
3508 TB Utrecht
The Netherlands

ISSN: 0024-3275

Linear Planar Augmentation Algorithms for Outerplanar Graphs*

Goos Kant
Dept. of Computer Science, Utrecht University
P.O. Box 80.089, 3508 TB Utrecht, the Netherlands

Abstract

In this paper we show that for outerplanar graphs G the problem of augmenting G by adding a minimum number of edges such that the augmented graph G' is planar and bridge-connected, biconnected or triconnected can be solved in linear time and space.

keywords: *outerplanar graphs, augmentation algorithms, data structures, connectivity.*

1 Introduction

A number of graph problems can be viewed as augmentation problems [3]: Given a graph $G = (V, E)$, find a set of edges E' of minimum size such that $G' = (V, E \cup E')$ satisfies a certain property. In this paper we inspect three properties, considered by Kant & Bodlaender [11]: bridge-connectivity, biconnectivity and triconnectivity, while preserving planarity, i.e., G is planar and G' also must be planar. The problems are called *the planar bridge-connectivity, biconnectivity* and *triconnectivity augmentation problem*, respectively [11]. These planar graph augmentation problems derive their significance from the need to draw planar graphs elegantly. Several efficient planar graph drawing algorithms require the input graph to fulfill certain connectivity constraints (see e.g., [9, 16, 21]). Thus, some extra (dummy) edges are added such that the resulting graph fulfills these constraints. In order to preserve the structure of G as much as possible, we want to add a minimum number of dummy edges.

As shown in [11], the planar biconnectivity augmentation problem is NP-complete and the planar bridge-connectivity and triconnectivity augmentation problem are

*This work was supported by the ESPRIT Basic Research Actions program of the EC under contract No. 4171 (project ALCOM II).

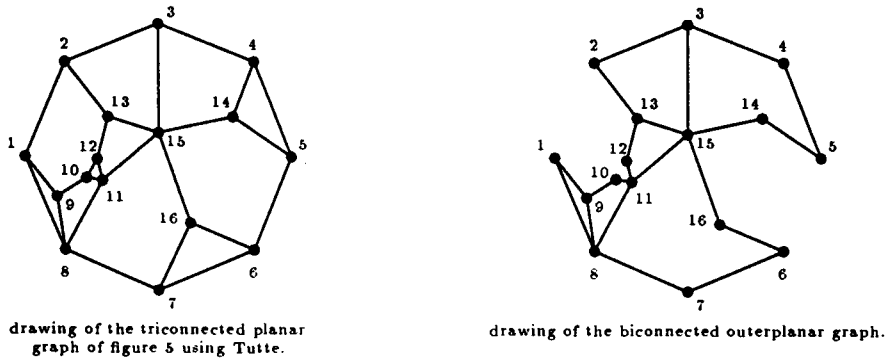


Figure 1: Drawing example of the outerplanar graph of figure 5.

still open. Approximation algorithms, working within $\frac{3}{2}, \frac{3}{2}$ and $\frac{5}{4}$ times optimal, respectively, are included in [11] as well. These problems, however, become efficiently solvable when we restrict our input graphs to trees and outerplanar graphs. We show in this paper that the planar bridge-connectivity augmentation problem for an outerplanar graph G can be solved in linear time, by modifying the general augmentation algorithm to meet bridge-connectivity constraints of Eswaran & Tarjan [2]. Finding an augmentation that biconnects a graph seems to involve difficulties that are not present in the bridge-connectivity augmentation algorithm [3]. Also here, the planar biconnectivity augmentation algorithm in this paper is more complicated, but can still be done in linear time, by modifying the corresponding general linear augmentation algorithm of Hsu & Ramachandran [8] in an elegant way. (Actually, the algorithm of Hsu & Ramachandran is a corrected and simplified version of the algorithm of Rosenthal & Goldner [17].) In the case of arbitrary planar graphs (for which the problem is NP-complete [11]), a strongly related algorithm can be obtained, working in $O(n \log n)$ time and with a performance ratio 2 [11]. Recently a linear algorithm is presented to augment a graph by adding a minimum number of edges to admit triconnectivity [7]. In this paper we show by a totally different approach how to make a (not necessarily biconnected) outerplanar graph triconnected and still planar by a minimum number of edges in linear time, thereby using a variant of the linear planar bridge-connectivity algorithm for outerplanar graphs. This means that several planar drawing algorithms for drawing biconnected and triconnected planar graphs can be used for outerplanar graphs, by augmenting them with as few edges as possible. In figure 1 an example of a drawing of a biconnected outerplanar graph is given, by making it triconnected and applying Tutte's convex drawing algorithm [21]. Observations how to augment G while maintaining outerplanarity are included as well.

Augmentation algorithms seems to have a lot of interest nowadays. Other recent augmentation algorithms, which do not preserve planarity, can be found in [4, 14], but they only consider the edge-connectivity constraints. Other recent algorithms,

dealing with vertex-connectivity, can be found in [6, 12]. In [10] an augmentation algorithm is described to augment planar graphs such that they are triangulated, while preserving degree constraints.

This paper is organized as follows: in section 2 some definitions are given. In section 3 we give some general observations on drawing outerplanar graphs and augmentation techniques. In section 4, 5 and 6 the planar augmentation algorithms for outerplanar graphs to meet bridge-connectivity, biconnectivity and triconnectivity constraints, respectively, are presented.

2 Definitions

Let $G = (V, E)$ be an undirected graph with $|V| = n$ vertices and $|E| = m$ edges. G is *connected* if there is a path between every pair of vertices. If v is a vertex of G such that $G - \{v\}$ is disconnected, then v is called a *cutvertex*. If (x, y) is an edge such that $G' = (V, E - (x, y))$ is disconnected, then (x, y) is called a *bridge*. If G is connected and contains no cutvertices, it is *biconnected*. If G is connected and contains no bridges, it is *bridge-connected*. If G is connected and the deletion of any two vertices with incident edges preserves the connectivity, then G is called *triconnected*. The connected (biconnected, bridge-connected, triconnected) components of a graph are its maximal connected (biconnected, bridge-connected, triconnected) subgraphs. A tree is an undirected, connected, acyclic graph.

A graph is called *planar* if it can be drawn in the plane such that there is no pair of crossing edges. A graph is called *outerplanar* if it can be drawn as a planar graph with all vertices occuring on one face, called the *outerface*. A graph is outerplanar if and only if its biconnected components are outerplanar, and can be recognized in linear time [13].

Central to the algorithms is the concept of the *block* graph of G, denoted by $bc(G)$ (cf. Harary [5]): each biconnected component (or, shortly, block) is represented by a b-vertex and each cutvertex of G is represented by a c-vertex of $bc(G)$ and two vertices u, v of $bc(G)$ are adjacent if and only if the corresponding cutvertex of u in G is contained in the corresponding block of v in G or vice versa. It can easily be shown that $bc(G)$ is always a forest. It is known as the bc-tree of G when G is connected. Every path in $bc(G)$ contains alternating b- and c-vertices. A pendant block is a block which contains exactly one cutvertex. Let $p(v)$ of a cutvertex v denote the number of pendants, connected at v. Let p be the number of pendants of G (thus the number of leaves in $bc(G)$) and let q be the number of isolated vertices in $bc(G)$. Let $d(v)$ denote the number of components of $G - \{v\}$, i.e., the graph after deleting cutvertex v. Each component of $G - \{v\}$ is called a v-block. If G is a tree then for all v, $d(v) = deg(v)$, the degree of vertex v. Let $d = \max_{v \in V}\{d(v)\}$.

When constructing $bc(G)$ for an outerplanar graph G, we assume that the cutvertices $c_1, \ldots, c_k$, connected at a certain blockvertex b in $bc(G)$, appear in the order they appear on the outerface of the corresponding block B of G. This means that

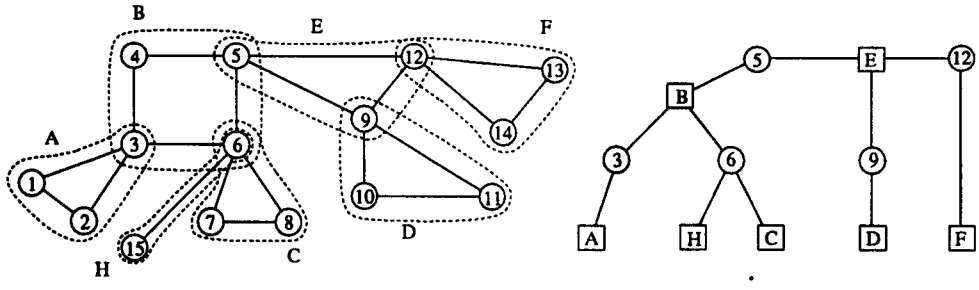


Figure 2: Example of a graph and a block graph (from [8]).

in $bc(G)$, the order of the sons of every blockvertex is fixed, and for a cutvertex, any order of the sons is allowed. Such a "fixed" $bc(G)$ can be constructed in linear time [20] and will be used in the next sections throughout. Note that adding one edge between two arbitrary pendants of an outerplanar graph does not destroy planarity, because we can always embed this edge on the outerface. An added edge will also be called *augmenting* in the following sections. In figure 2 an outerplanar graph G and the corresponding block graph $bc(G)$ are given.

3 Preliminaries

Outerplanar graphs are an interesting class of planar graphs, since all vertices share one common face. Several problems, which are NP-hard for planar graphs, become easily solvable for outerplanar graphs, e.g., the CHROMATIC NUMBER PROBLEM. With respect to augmentation problems, it has been shown by Kant & Bodlaender [11] that every outerplanar graph can be triangulated while minimizing the maximum degree in polynomial time. Since the problem is NP-complete for planar graphs, this again gives a contrast between outerplanar and planar graphs.

Biconnected outerplanar graphs can be drawn by placing all vertices on the cornerpoints of a regular n-gon, and drawing the chords as straight lines inside the cycle. This leads to a convex planar drawing, but the minimum angle can be $O(\frac{1}{n})$ and the ratio between the longest and smallest edge can be $O(n)$. Another algorithm for drawing outerplanar graphs works only for triangulated outerplanar graphs, i.e., all faces inside the cycle are triangles. It starts with drawing a triangle ABC, which has one edge on the outerface, say AB. It draws AB horizontal and draws C above AB such that $AC = BC$ and $\angle BAC = \angle ABC$. From AC and BC recursively the remaining parts of the outerplanar graph are drawn. See figure 3 for an example. Here the drawing can be constructed such that the minimum angle is $\frac{\pi}{2(d+2)}$. However, the ratio between the longest and smallest edge can be $O(2^n)$. This drawing construction has also succesfully been applied by Lin & Skiena [18],

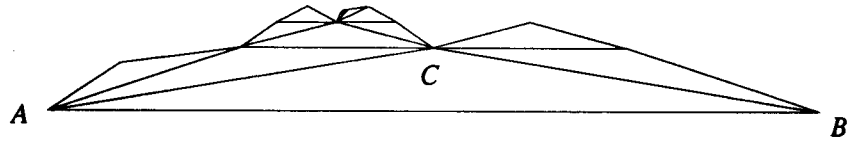


Figure 3: Drawing an outerplanar graph.

to draw a polygon P on a grid whose visibility graph is a triangulated outerplanar graph.

In this section we consider the problem of augmenting an outerplanar graph such that the resulting graph is again outerplanar and bridge- or biconnected. If G is not connected, then we apply the following algorithm of [2]: let t be the number of trees in $bc(G)$. Let $v(i)$, $1 \leq i \leq 2t$ be a set of vertices of $bc(G)$ such that

1. $v(2i-1)$ and $v(2i)$ are each a pendant or an isolated vertex in the ith tree of $bc(G)$, for each i, $1 \leq i \leq t$.

2. $v(2i-1) = v(2i)$ if and only if the ith tree of $bc(G)$ is an isolated vertex.

It now easily follows that $bc(G) \cup \{(v(2i), v(2i+1)) | 1 \leq i < t\}$ is a tree having $p' = p + 2q - 2(t-1)$ pendants and no isolated vertices [2]. We add the edges between arbitrary vertices of the pendants in G, if there is an edge between the corresponding leaves in $bc(G)$. Hence we may assume further that G is connected and outerplanar.

Biconnecting G while maintaining the outerplanarity is obtained as follows by the algorithm of Read [15]: let an embedding of the outerplanar graph be given, i.e., let the edges around each vertex be given in the adjacency list in clockwise order with respect to a planar drawing. Compute the blocks (e.g., by using [19]). We add an edge (u, w) between two consecutive neighbors u and w of cutvertex v, if u and w belong to different blocks. For every vertex v we do this exactly $d(v)-1$ times, since v belongs to $d(v)$ different blocks. After this addition, v is not a cutvertex anymore, because there is a path between the v-blocks, not using v but the new added edges. To see that G is still outerplanar we inspect a cutvertex v. Let u and w be two consecutive neighbors of v in the embedding of G, belonging to different blocks, but no edge (u, w) is added by the augmentation algorithm (since only $d(v) - 1$ edges are added). Notice that (u, v) and (v, w) belong to the outerface before the augmentation, hence also after the augmentation v still belongs to the outerface. This completes the following theorem.

Theorem 3.1 *There is a simple linear-time algorithm to augment an outerplanar graph by adding edges to a biconnected outerplanar graph.*

By changing the algorithm a little it can be proved that the degree of every vertex increases by at most 2 (see Kant & Bodlaender [11]). Since every biconnected graph is also bridge-connected, this algorithm can be used to bridge-connect G as well. Notice that an outerplanar graph G cannot be triconnected, since deleting any pair of vertices u, v, for which (u, v) is a chord of G disconnects G.

Biconnecting (or bridge-connecting) G by a minimum number of edges while maintaining outerplanarity seems to be a much harder problem. Even for trees T with p leaves it is not very difficult to construct examples in which $\frac{p}{2}$ edges are sufficient to biconnect (or bridge-connect) T, and examples in which $p - d$ edges are necessary to biconnect (or bridge-connect) T. Also there exists outerplanar graphs G with b blocks and 2 pendants, for which $b - 1$ edges are necessary to biconnect G. Moreover it seems pretty hard to prove that the solution is also minimum.

Therefore we inspect the problem of augmenting outerplanar graphs such that the augmented graph is bridge-connected, biconnected or triconnected and planar. We show that the number of edges added is equal to the non-planar case, and therefore optimal. Also the algorithms are very simple and can easily be implemented to run in linear time. For drawing biconnected and triconnected planar graphs a lot of advanced algorithms are known, which can be used to draw the (augmented) outerplanar graphs.

4 Bridge-connectivity

In this section we inspect the problem of how to add a minimum number of edges to an outerplanar graph G, such that the augmented graph G is bridge-connected and planar. Bridge-connecting G is equal to bridge-connecting the forest $bc(G)$. The algorithm for bridge-connecting $bc(G)$ is inspired on the general bridge-connecting algorithm of Eswaran & Tarjan [2]. They gave the following lowerbound on the number of edges needed to make $bc(G)$ bridge-connected (recall the definitions of section 2):

Theorem 4.1 ([2]) *At least $\lceil \frac{p}{2} \rceil + q$ edges are needed to make $bc(G)$ bridge-connected.*

If G is not connected, then we apply the algorithm of section 3, hence we may assume further that G is connected and let T be its bc-tree. All that remains is to find a set of $\lceil \frac{p}{2} \rceil$ edges to bridge-connect any tree with p leaves and a fixed order of the leaves given.

This bound is attainable. Hereto we first convert the tree T into a related tree T' by deleting every vertex v of degree 2 and merging its two incident edges into one. T' can be obtained in linear time and every internal vertex has degree ≥ 3. We now pick an arbitrary nonleaf blockvertex as root. We number the vertices in postorder, which means that we traverse the tree by a depth-first search traversal, where we first visit the sons of a vertex for numbering, before numbering the vertex. We visit the sons from left to right, which means that the leftmost descendant leaf

gets number v_1 and the root gets number v_n. Let $v(1), \ldots, v(p)$ be the leaves of T', ordered in increasing v_i-number. The following lemma is easy to prove:

Lemma 4.2 *The descendants of any vertex have consecutive numbers in any post-order numbering.*

The idea of the algorithm is as follows: we visit the vertices in increasing post-order numbering. If v_i is a leaf $v(k)$, then we simply add an edge $(v(k-1), v(k))$, if k is odd. If v_i is an internal vertex, then we test whether there exist an augmenting edge $(v(\alpha), v(\beta))$, with $v(\alpha)$ a descendant of v_i and $v(\beta)$ not a descendant of v_i. If not then we change it such that this holds. The algorithm can now be described more formally as follows:

BRIDGE-CONNECT
  $A := \emptyset$; { A becomes the augmenting set of edges. }
  $x := 1$;
  for $i := 2$ **to** n **do**
    if v_i is a leaf $v(k)$ **then**
      if k is odd **then** $A := A \cup \{(v(k-1), v(k))\}$
    else
      let $v(j_1), \ldots, v(j_h)$ be the descendant leaves of v_i;
      if j_1 is even **and** $x < j_1$ **then**
        $A := A - \{(v(j_1), v(j_2))\} \cup \{(v(x), v(j_1))\}$; $x := j_2$;
  endfor;
  if $x < p$ **then** $A := A \cup \{(v(x), v(p))\}$ **else** $A := A \cup \{(v(1), v(p))\}$

It follows from the algorithm that $|A| = \lceil \frac{p}{2} \rceil$, but to prove that G indeed is bridge-connected and planar, we need the following lemma of [13]:

Lemma 4.3 *Let (v, w) be an edge of $G' = (V, E' \cup A)$. Then (v, w) is a bridge of G' if and only if v is the father of w in T and there is no edge $(\alpha, \beta) \in A$ such that α is a descendant of w in T and β is not a descendant of w in T.*

Lemma 4.4 *$G' = (V, E' \cup A)$ is bridge-connected.*

Proof: Let (v, w) be any edge of G' such that v is the father of w in T. Suppose the leaves of T which are descendants of w are $\{v(j_1), \cdots, v(j_h)\}$. w is not the root thus $j_1 > 1$ or $j_h < p$, assume $j_1 > 1$ (the case $j_h < p$ goes analog). If w is a leaf then $j_1 = j_h$ and w gets and incident edge to another leaf of G', so assume that w is not a leaf, thus $j_h > j_1$. If j_1 is odd then by BRIDGE-CONNECT the edge $(v(j_1 - 1), (v(j_1))$ is added to A and $v(j_1 - 1)$ is not a descendant of w, so assume j_1 is even. If at this moment $x < j_1$ holds, then the edge $(v(x), v(j_1))$ will be added in BRIDGE-CONNECT, and $v(x)$ is not a descendant of w. Let us call such added edges $(v(x), v(j_1))$ in BRIDGE-CONNECT *special*.

Assume finally that j_1 is even and $x \geq j_1$. Since $x = 1$ initially and $x \geq j_1 > 1$ now, x has been changed and, hence, one or more special edges $(v(a), v(b))$ are added. For every special edge $(v(a), v(b))$ holds that a is odd and b is even, and the next special edge starts in $v(b+1)$. The first special edge starts in $v(1)$ and the last one when visiting w ends in $v(x-1)$. Since j_1 is even no special edge can start in $v(j_1)$. Let a be the highest number $< j_1$ such that there starts a special edge $(v(a), v(b))$ in $v(a)$. But now $b \geq j_1$, because otherwise $b \leq j_1 - 2$ and there would start a special edge in $v(b+1)$. Contradiction with the fact that a was the highest number $< j_1$ where a special edge $(v(a), v(b))$ starts. $v(a)$ is not a descendant leaf of w, but $v(b)$ is. By lemma 4.3 we can conclude that (v, w) is not a bridge of G', if $j > 1$. □

Lemma 4.5 *$G' = (V, E' \cup A)$ is planar.*

Proof: It follows directly from BRIDGE-CONNECT that when visiting leaf $v(k)$, that for all leaves $v(i)$, $x < i < k$ holds that $(v(i), v(i-1)) \in A$ with i odd. This means also that there are no edges $(v(\alpha), v(\beta)) \in A$ with $\alpha < x < \beta$. Hence when adding special edge $(v(x), v(j_1))$ to A or when adding $(v(k-1), v(k))$ to A, there are no two edges $(v(\alpha), v(\beta))$ and $(v(\gamma), v(\delta)) \in A$, with $\alpha < \gamma < \beta$ and $\delta < \alpha$ or $\delta > \beta$. Since this holds when visiting any vertex v_i, it follows that G' is planar. □

Constructing the graph $bc(G)$ can easily be obtained in linear time, as well as adding the $t-1$ edges such that the graph is connected. Constructing the bc-tree and the post-order numbering can easily be performed in linear time. At each internal node v we store a pointer to its leftmost leaf. Using this plus one pointer to leaf $v(x)$ the total required time is $O(1)$ when visiting v_i in BRIDGE-CONNECT. This leads to the main theorem of this section:

Theorem 4.6 *The planar bridge-connectivity augmentation problem can be solved in linear time and space for outerplanar graphs.*

5 Biconnectivity

Before introducing the algorithm for biconnecting outerplanar graphs, we first study some properties of the block graph. The following definitions are coming from [8].

Definition 5.1 *A vertex v of $bc(G)$ is called* massive *if and only if v is a c-vertex with $d(v) - 1 > \lceil \frac{p}{2} \rceil$. A vertex v of $bc(G)$ is* critical *if and only if v is a c-vertex with $d(v) - 1 = \lceil \frac{p}{2} \rceil$. The graph $bc(G)$ is* critical *if and only if there exists a critical c-vertex in $bc(G)$.*

Definition 5.2 *A block graph $bc(G)$ is* balanced *if and only if G is connected and without any massive c-vertex. (Note that $bc(G)$ could have a critical c-vertex.) A graph G is balanced if and only if $bc(G)$ is balanced.*

Definition 5.3 (the leaf-connecting condition) *Two leaves u_1 and u_2 of $bc(G)$ satisfy the leaf-connecting condition if and only if u_1 and u_2 are in the same tree of $bc(G)$ and the path P from u_1 to u_2 in $bc(G)$ contains either (1) two vertices of degree more than 2, or (2) one b-vertex of degree more than 3.*

Some first observations concerning the block graph $bc(G)$ are the following:

Lemma 5.1 ([17]) *There can be at most one massive vertex in $bc(G)$. If there is a massive vertex in $bc(G)$, then there is no critical vertex in $bc(G)$, and there can be at most two critical vertices in $bc(G)$, if $p > 2$.*

The following fact for updating $bc(G')$ from $bc(G)$ is given in [17].

Theorem 5.2 *Given a graph G and its block graph $bc(G)$, adding an edge between two leaves u and v of $bc(G)$ creates a cycle C. Let G' be the graph obtained by adding an edge between u' and v' in G where u' and v' are not cutvertices in the blocks represented by u and v respectively. The following relations hold between $bc(G)$ and $bc(G')$.*

- *Vertices and edges of $bc(G)$ that are not in the cycle C remain the same in $bc(G')$.*
- *All b-vertices in $bc(G)$ that are in the cycle C contract to a single b-vertex b' in $bc(G')$.*
- *Any c-vertex in C with degree equal to 2 is eliminated.*
- *A c-vertex x in C with degree greater than 2 remains in $bc(G')$ with edges incident on vertices not in the cycle. The vertex x also attaches to the b-vertex b' in $bc(G')$.*

Inspect the graph G and $bc(G)$ of figure 2. If we add an edge $(8, 10)$ to G, then all b-vertices on the path between C and D in $bc(G)$ are contracted in one b-vertex X, as shown in figure 4.

Lemma 5.3 ([8]) *Let u_1 and u_2 be two leaves of $bc(G)$ satisfying the leaf-connecting condition (definition 5.3). Let α and β be non-cutvertices in blocks of G represented by u_1 and u_2 respectively. Let G' be the graph obtained from G by adding an edge between α and β and let P represent the path between u_1 and u_2 in $bc(G)$. The following three conditions are true.*

- *$p' = p - 2$.*
- *If v is a c-vertex in P with degree greater than 2 in $bc(G)$, then the degree of v decreases by 1 in $bc(G')$.*

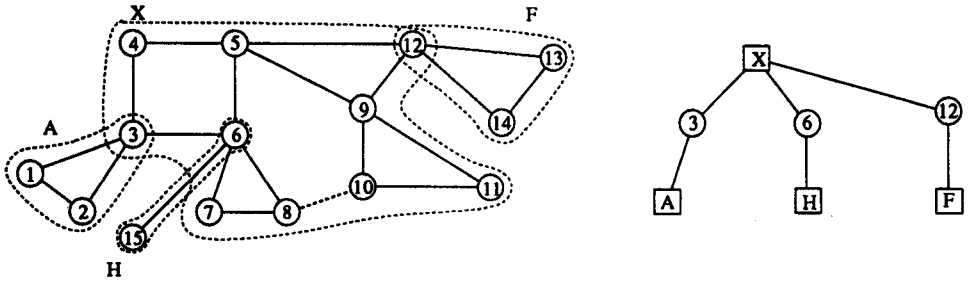


Figure 4: Example to get $bc(G')$ from $bc(G)$ when adding edge $(8, 10)$ in the outerplanar graph of figure 2 (from [8]).

- *If v is a c-vertex in P with degree equal to 2, then v is eliminated in $bc(G')$.*

We first state a lower bound on the number of edges needed to augment a graph to reach biconnectivity.

Theorem 5.4 ([2]) *Let G be an undirected graph with t connected components and let q be the number of isolated vertices in $bc(G)$. Then at least $max\{d+t-2, \lceil \frac{p}{2} \rceil + q\}$ edges are needed to biconnect G, if $q + p > 1$.*

The algorithm to obtain this bound in general (without the requirement of planarity) is based on the following observation [8]: in every step we search for two leaves u_1 and u_2 of $bc(G)$ satisfying the leaf-connecting condition (definition 5.3). Then we add an edge between two non-cutvertices in blocks of G represented by u_1 and u_2. Then the number of leaves will decrease by 2 when updating $bc(G)$. This technique simplifies the original algorithm of Rosenthal & Goldner [17] where the path from u_1 to u_2 should contain the c-vertex with highest degree.

We will show that applying this will lead to the lowerbound of theorem 5.4.

The algorithm of Hsu & Ramachandran [17] consists of three stages. We will describe the three stages here. Stage 1 makes the graph connected; stage 2 eliminates the massive vertices and stage 3 makes the graph biconnected. Stage 1 and 2 can easily be modified for our problem but for stage 3 we have to modify both algorithm and data structure intensively.

5.1 Stage 1

First we have to connect the t components with each other, using $t - 1$ edges. But for this we can use the algorithm described in section 3. This corresponds with the given lowerbound of theorem 5.4, hence we may now further assume that G is connected, and we try to biconnect its bc-tree T by $\max\{d - 1, \lceil \frac{p}{2} \rceil\}$ edges, without destroying planarity.

5.2 Stage 2

Suppose there is a massive vertex v^* in T. (If no massive vertex v^* exists, no action is taken at stage 2.) Let γ be the number of components of $T - v$ containing only one leaf of T. Call such components *1-chains*. There are $d(v^*) - \gamma$ components of which each contains at least 2 leaves, so $p \geq \gamma + 2(d(v^*) - \gamma)$. We pick v^* as the root of T and number the vertices of T in preorder: traverse the tree by a depth-first search traversal, where we first number the vertex, before visiting the sons from left to right in T. Let $v(1), \ldots, v(p)$ be the pendants of T, ordered so that number$(v(i)) <$ number$(v(i+1))$, for $1 \leq i < p$. We now add 2δ edges such that as much as possible 1-chains are coalesced into one by the following algorithm, with $\delta = d(v^*) - 1 - \lceil \frac{p}{2} \rceil$.

$i := 1$; $A := \emptyset$; { A becomes the augmenting set of edges. }
while $|A| < 2\delta$ **and** $i < n$ **do**
  if $v(i)$ and $v(i+1)$ are both 1-chains **then** $A := A \cup \{(v(i), v(i+1))\}$;
  $i := i + 1$
od;
$i := 1$;
while $|A| < 2\delta$ **do**
  if $v(i)$ is a 1-chain **and** $(v(i), v(i+1)) \notin A$ **then** $A := A \cup \{(v(i), v(i+1))\}$;
  $i := i + 1$
od

Let G' be the augmented graph by adding the 2δ edges of A between the corresponding pendants. It easily follows that the edges can be added such that G' is still outerplanar. Let T' be the corresponding bc-tree with p' leaves, with $p' = p - 2\delta$, and let $d'(v)$ denote the d-value of v in T'.

Lemma 5.5 ([17]) *For all cutvertices v of T', $d'(v) - 1 \leq \lceil \frac{p'}{2} \rceil$ holds.*

Proof: For v^*, it holds that $d'(v^*) - 1 = d(v^*) - 1 - 2\delta = \lceil \frac{p}{2} \rceil - \delta = \lceil \frac{p'}{2} \rceil$. Now consider a cutvertex $v \neq v^*$, and suppose that $d'(v) - 1 > \lceil \frac{p'}{2} \rceil$. Now $p \geq d(v^*) + d(v) - 2 = (d(v^*) - 1) + (d(v) - 1) > \lceil \frac{p}{2} \rceil + (\lceil \frac{p'}{2} \rceil + \delta) = \lceil \frac{p}{2} \rceil + (\lceil \frac{p}{2} \rceil - \delta) + \delta = 2\lceil \frac{p}{2} \rceil \geq p$, which is a contradiction. □

5.3 Stage 3

In this stage, we have to deal with a graph G where $bc(G)$ is balanced. The idea is to add an edge between two leaves y and z under the conditions that the path P between y and z passes through all critical vertices and the new block tree has two less leaves if $bc(G)$ has more than 3 leaves. Thus the degree of any critical vertex decreases by 1 and the tree will remain balanced. We also want y and z to be leaves that satisfy the leaf-connecting condition, because then we can use lemma

5.3 and will lead to the desired lowerbound of $\lceil \frac{p}{2} \rceil$ edges to biconnect a balanced graph. Moreover, this path must be such that adding an edge between two non-cutvertices of the blocks represented by y and z does not destroy planarity. We call two leaves y and z *adjacent* if after adding (y, z) to $bc(G)$ all other leaves of $bc(G)$ are either inside or outside the new created cycle C. Hence in every step we look for two adjacent leaves y and z, satisfying the leaf-connecting condition and the path P between them passes through all critical vertices. We will show that these pairs always exists.

Since the vertices with degree 2 are of no interest in the algorithm, we eliminate them from the bc-tree, by contracting their two incident edges into one.

The algorithm can now be described as follows:

```
SEQ_BCA
(* G has at least 3 vertices and bc(G) is balanced; *)
   Let T be bc(G) rooted at an arbitrary b-vertex b*;
   while p ≥ 2 do
      if d = 2 then
         let v be a b-vertex (unequal to b*) with degree > 2
      else
         let v be a c-vertex with the largest degree in T;
      use algorithm PATHFINDER(v) to find a vertex w with degree > 2
      and two adjacent leaves y and z such that the path
      between them passes through v and w;
      find non-cutvertices α and β in the corresponding blocks
      of G represented by y and z respectively;
      add an edge between α and β; update the block graph T
   od;
```

We now describe the procedure PATHFINDER, that finds a vertex w and the two adjacent leaves y and z whose path P between them passes v and w. Recall from section 2 that any order of sons of a cutvertex is allowed, but the sons of a blockvertex may only be swapped from a left-to-right order into a right-to-left order. We construct the following data structure for $bc(G)$ (which is almost equal to the construction of PQ-trees, introduced in [1]):

- Every vertex v in $bc(G)$ is represented by a record. If v is not a leaf, then v has a pointer to its leftmost and rightmost son, called l-son and r-son, respectively.
- The sons of each vertex are stored in a doubly linked list.
- If a vertex is a son of a c-vertex or the left- or rightmost son of a b-vertex, then it has a father-pointer to it, otherwise this pointer is nil.

Since we may permute the children of a c-vertex in any order, we sort the children of each c-vertex v such that all non-leaves occur at one side, say starting from the leftmost son of v. The idea now is to walk from v towards root b^*, until the father-pointer is nil or b^*. Let w be the highest reached vertex from v to b^*, then we change the tree such that we can reach v from w by following only l-son pointers. We reach leaf y now by following l-son pointers from v and reach leaf z by following r-son pointers from the left brother of w (if w has at least two sons), otherwise we follow r-son pointers from w.

If there are only three leaves, then we can reduce $bc(G)$ into a new block tree with two leaves by picking any pair of leaves in $bc(G)$ and connecting them. We know that we can reduce a block tree of 2 leaves into a single vertex by connecting the two leaves. So assume further that $p > 3$, then the algorithm can be described more formally as follows (swap(a, b) changes the contents of a with b and vice versa):

```
PATHFINDER(vertex v);
(* v is the c-vertex with largest degree in T or a b-vertex with degree > 2; *)
   w := v;
   while father(w) ≠ nil and father(w) ≠ b* do
      if w is not leftmost son of father(w) then swap(w, l-son(father(w)));
      w := father(w);
   od;
   y := v;
   while y is not a leaf do
      if leftmost son of y is a leaf then swap(l-son(y), r-son(y));
      y := l-son(y);
   od;
   if w has degree ≥ 2 then z := left brother of w
   else z := rightmost son of w;
   while z is not a leaf do
      if rightmost son of z is a leaf then swap(l-son(z), r-son(z));
      z := r-son(z);
   od
```

Lemma 5.6 *y and z are adjacent.*

Proof: From the father of w there are two paths downwards: one via w and v to y, following only the leftmost son pointers, and one via the left brother of w or via the rightmost son of w to leaf z, following only the rightmost son pointers. Hence all the other leaves are on one side of the cycle, obtained by adding (y, z) to T. □

This means that we can add an edge (α, β) between two non-cutvertices of the blocks of y and z without destroying the planarity.

Lemma 5.7 *y and z satisfy the leaf-connecting condition.*

Proof: v is a vertex with degree > 2. When we stop in the **while**-loop of PATHFINDER, then $father(w)$ is nil, and hence has degree > 2 and is part of path P, or $father(w) = b^*$. Assume w.l.o.g. that $d(b^*) < 3$, thus b^* has degree 1. Of course w has degree > 2. If $w \neq v$ then we have already found another vertex w with degree > 2, so assume $w = v$. If v is a c-vertex and there is another vertex v_2 with degree > 2 in T, then v_2 is the leftmost or rightmost son of v, because the children of each c-vertex in T are sorted such that all non-leaves occur at one side. But path P visits from v both the leftmost and rightmost son of v and, hence, v_2 will be a part of P by PATHFINDER. Assume finally that v is a b-vertex (hence $d = 2$). If v has degree > 3 then $w = v$ satisfies the leaf-connecting condition. Otherwise v has only two sons. PATHFINDER walks downwards through both sons. Hence if both sons are not leaves, then one of them has degree > 2 and will be detected by PATHFINDER. Hence the leaves y and z always satisfy the leaf-connecting condition. □

Lemma 5.8 *For outerplanar graphs G with $bc(G)$ balanced, the algorithm* SEQ_BCA *(using the procedure* PATHFINDER*) finds a set of $\lceil \frac{p}{2} \rceil$ edges, that when added to G, yield a biconnected planar graph.*

Proof: Assume w.l.o.g. that $p > 3$. In this case, a critical vertex must have degree more than 2.
Case 1: If $bc(G)$ has two critical vertices v and w, then all other non-leaves have degree 2 and, hence, are eliminated from the tree. Since PATHFINDER will find another vertex with degree > 2 if present, both v and w will be part of P.
Case 2: If $bc(G)$ has only one critical vertex v, algorithm SEQ_BCA finds it. Because $bc(G)$ is balanced and $p > 3$, there must exist another vertex w with degree more than 2. Otherwise v is massive. PATHFINDER will find a vertex w with degree > 2.
Case 3: The block tree $bc(G)$ has no critical vertex. Then SEQ_BCA will take the c-vertex with highest degree if $d \geq 2$, otherwise it takes a b-vertex with degree ≥ 3. In both cases PATHFINDER will find another vertex w with degree > 2 on the path P.

In all three cases, we can find two vertices of degree more than 2 or a b-vertex of degree more than 3. Thus by lemma 5.3. the number of leaves in the new block tree reduces by two. When v or w is critical, the value of d is reduced by 1. Thus the block tree remains balanced. Hence we can achieve the lower bound of theorem 5.4 by the algorithm. □

For finding the c-vertex with highest degree, we maintain an array *bucket*, and initially we store in $bucket[i]$ all c-vertices with degree i. Using an extra pointer, we can find in $O(1)$ time the c-vertex with highest degree. When the degree of a c-vertex decreases, we can remove it from one entry and store it in another entry in

$O(1)$ time. If all buckets are empty, meaning that there are no c-vertices in T, then we have to take a b-vertex with degree > 2. Notice that either b^* or one of its sons must have degree > 2, hence we can easily find such a vertex in $O(1)$ time.

If $d(b^*) \geq 3$, then the lowest common ancestor of y and z is always a b-vertex b_1. Let w_1, w_2 be two sons of b_1, which are part of P. We now walk from y to z and make every c-vertex a son of b_1 and we make all sons of a b-vertex sons of b_1. We store them between w_1 and w_2 in the order we visit them between y and z. Since all b-vertices on P are now eliminated and every c-vertex on P is now son of P, the degree of several vertices on P is decreased in the updated tree. Since only the degree of vertices on P decreases, we test these vertices for degree 2, because then we eliminate them. If the lowest common ancestor of y and z is a cutvertex c_1, then we take a new b-vertex b_1, and do the same as above, and add finally this b-vertex as son of c_1 in the tree T.

Lemma 5.9 *Algorithm* SEQ_BCA *runs in* $O(n+m)$ *time.*

Proof: The block-tree can be built in $O(n+m)$ time. The total number of vertices in the block tree is $O(n)$. A linear time bucket-sort routine is used to sort the degree of the c-vertices. By the algorithm PATHFINDER, every path P between the leaves y and z can be found in $O(|P|)$ time. By theorem 5.2, the number of times a vertex is visited is no more than its degree. Since the summation of degrees of all vertices in a tree with n vertices is $O(n)$, the lemma is true. □

This lemma completes the following result:

Theorem 5.10 *There is a linear algorithm to augment outerplanar graphs by a minimum number of edges such that the resulting graph is biconnected and planar.*

6 Triconnectivity

6.1 Triconnecting Trees

We now inspect the problem how to augment an outerplanar graph G with a minimum number of edges such that the augmented graph G' is triconnected and planar. Hereto, we first restrict our attention to trees, where every vertex of degree 2 must get one additional edge and every leaf must get two additional edges. This means that at least $\lceil \frac{K}{2} \rceil + L$ edges are required to triconnect a tree, with L the number of leaves and K the number of vertices v with degree 2. This bound is attainable while preserving planarity.

First, if G does not contain a vertex of degree ≥ 3, then G is a path P of vertices $v_1, \ldots, v_p$. Triconnecting this path P is easy by adding the edges $(v_1, v_3), (v_2, v_4), (v_3, v_5), (v_4, v_6), \ldots, (v_{p-2}, v_p), (v_{p-1}, v_1)$. Hence assume from now on that there is at least one vertex r with $\deg(r) \geq 3$, which is the root of T.

Let us call a path P in T with only vertices of degree 2 a *chain*. If $|P|$ (the length of P by counting the number of vertices) is greater than 2 then we handle P as follows. Let $v_1, v_2, \ldots, v_k$ be the vertices of P. Add an edge between v_1 and v_k, v_2 and v_{k-1}, etc., until one or two vertices are left ($v_{\lfloor \frac{k}{2} \rfloor}$ and $v_{\lceil \frac{k}{2} \rceil}$). Adding edges from $v_{\lfloor \frac{k}{2} \rfloor}$ and $v_{\lceil \frac{k}{2} \rceil}$ to vertices, not part of P makes the chain P triconnected, hence we may assume further that every chain has length 1 (a *1-chain*) or length 2 (a *2-chain*). We assume that the order of the children of each vertex in T is fixed, and by a preorder numbering (see section 4.2) we determine the left-to-right order $l_0, \ldots, l_{L-1}$ of the leaves. Let for each nonleaf vertex v_i, $l(v_i)$ be its leftmost descendant leaf, and let for every leaf l_i in T, $V(l_i)$ be the set of vertices v, with $deg(v) = 2$ and $l(v) = l_i$. For each l_i we sort the elements of $V(l_i)$ in increasing order, according to their preorder-number (by bucket-sort).

We now try to add edges between vertices of $V(l_i)$ to achieve triconnectivity while preserving planarity. We must be sure that we do not add edges between two vertices of a same 2-chain. For this we visit the vertices of $V(l_i)$ and add, if allowed, edges between consecutive vertices:

CHAIN-EDGES($V(l_i)$);
  let $V(l_i) = \{v_1, \ldots, v_p\}$; $x := 1$;
  for $i := 3$ **to** p **step** 2 **do**
    if $chain(v_{i-1}) = chain(v_i)$ **then** add edge (v_x, v_{i-1}); $x := i$
    else add edge (v_{i-1}, v_i);
  endfor

The algorithm looks like the algorithm BRIDGE-CONNECT, but we only have to visit the vertices of $V(l_i)$ in increasing order. Similar to lemma 4.4, it can easily be shown that CHAIN-EDGES will preserve planarity but moreover, it follows that there will be no edge added between two vertices of a common chain. This means that all chains, which received an edge, are now become triconnected. If $|V(l_i)|$ is odd, then one vertex remains unmatched, otherwise two. For all sets $V(l_i)$, let $V(l_i)$ contain the unmatched vertices of $V(l_i)$ after CHAIN-EDGES($V(l_i)$), hence $|V(l_i)| \leq 2$ for all leaves. Let $A = \{(l_i, l_{i+1})|, 0 \leq i < L\}$ (with additions modulo L). Now, do the following:

CHANGE-SETS
  for all l_i with $|V(l_i)| = 1$ **do**
    let $V(l_i) = \{v_j\}$;
    $A := A - \{(l_{i-1}, l_i)\} \cup \{(l_{i-1}, v_j)\}$
  od;
  for all l_i with $|V(l_i)| = 2$ **do**
    let $V(l_i) = \{v_{i_1}, v_{i_2}\}$, with v_{i_1} above v_{i_2};
    $A := A - \{(l_{i-1}, l_i)\} \cup \{(l_{i-1}, v_{i_1})\}$
  endfor

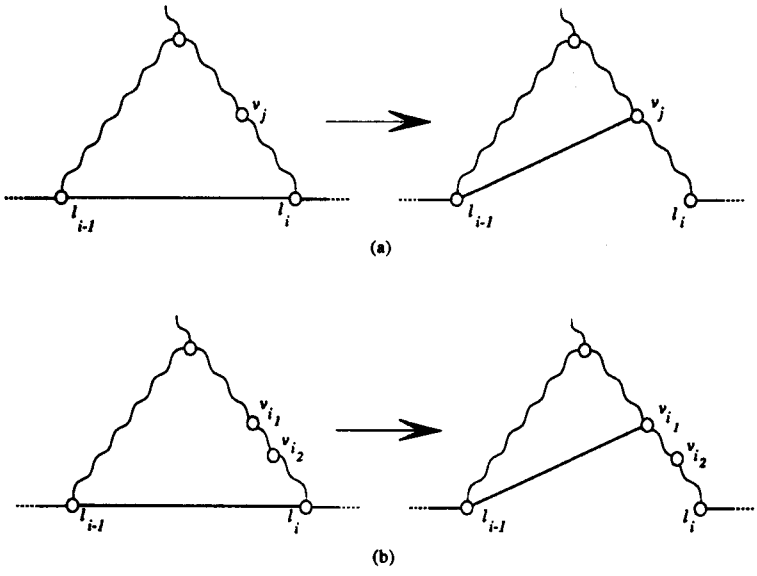


Figure 5: The change when $|V(l_i)| = 1$ (a) and when $|V(l_i)| = 2$ (b).

In figure 5 both changes are given. Now these leaves l_i with $|V(l_i)| = 1$ must get one outgoing edge, and the leaves l_i with $|V(l_i)| = 2$ must get one outgoing edge, as well as the corresponding vertices v_{i_2}, but they may not be connected with each other, because then the graph cannot become triconnected. For these pairs l_i, v_{i_2} we say that chain(l_i) = chain(v_{i_2}). Let $v_1, \ldots, v_k$ be the vertices of T, ordered left to right by increasing preorder number, which must get one additional edge. We again apply the algorithm CHAIN-EDGES on the set vertices $\{v_1, \ldots, v_k\}$. If two vertices remain unmatched, then we add an edge between them, otherwise we add an edge between v_1 and v_k.

Theorem 6.1 *$G' = (V, E \cup A)$ is triconnected and planar.*

Proof: Regarding the planarity it easily follows from Figure 2 and the algorithm CHAIN-EDGES that adding the set A to G does not destroy the planarity. For the triconnectivity constraint we have to show that if we delete two vertices from G', then the graph must still be connected.

If for all $V(l_i)$, $|V(l_i)| = 0$, then there is a cycle C on the leaves of T. For every leaf l_i, with $|V(l_i)| > 0$ there is an edge from l_{i-1} to an edge $v_j \in V(l_i)$ and there is a path from v_j to l_i, hence there is always a cycle C, containing all leaves. Let $v_i, v_j \in G'$ be two arbitrary vertices of G, we now show that $G' - \{v_i, v_j\}$ is connected.

Suppose first that v_i or v_j do not belong to C, hence there is a path P on the leaves. All leaves can reach each other via P and all vertices, not descendants of v_i

and v_j can reach each other via T. Since the degree of the root is 3, there is always a path from some leaves in $G' - \{v_i, v_j\}$ to r, hence via this path all vertices can reach each other.

Assume now that $v_i, v_j \in C$. If they are both leaves, then via T all vertices can reach each other. Otherwise, assume v_i is not a leaf and let l_i be the leftmost descendant of v_i. l_i has an augmenting edge to a vertex v_α of C, not a descendant of v_i. If v has degree 3 in T, there is a path from v_i to a leaf l_k in T, with $k \neq i$. There is a path form l_k via $l_{k+1}, l_{k+2}, \ldots$ to a vertex v_β of C, with $\beta \neq \alpha$. Hence there are two node-disjoint paths P_1, P_2 from v_i to vertices, not descendants of v_i. If $v_j \in P_1$ then via P_2 we can reach from v_i all descendants of v_j, which are not reachable via P_1. Moreover, via P_2, there is a path P_3 to the root r, not crossing v_j, hence via P_2 and P_3 we can reach all vertices, not descendants of v_i or v_j. If v_i has degree 2 in T, but degree 3 after CHAIN-EDGES, then v_i belongs to triconnected component and, hence, has 2 node-disjoint paths P_1 and P_2 via descendants of v_i to vertices, not descendants of v_i. Otherwise, as shown in figure 5 and by applying CHAIN-EDGES on the remaining set of vertices v of degree 2, v_i directly receives an edge to a vertex v_β of C. Since there is also a path via l_i to a vertex v_α of C, with $\alpha \neq \beta$ and v_α, v_β both not descendants of v_i, we again have 2 node-disjoint paths from v_i to all other leaves of C. From this it easily follows that $G' - \{v_i, v_j\}$ is connected for every arbitrary pair of vertices v_i, v_j, hence G' is triconnected. □

Since $|A| = \lceil \frac{K}{2} \rceil + L$, the augmenting set is optimal. Since also $|V(l_0)| + \cdots + |V(l_{L-1})| \leq n$, this leads to the following

Lemma 6.2 *There is a linear time and space algorithm to augment a tree by a minimum number of edges such that the augmented graph is triconnected and planar.*

6.2 Triconnecting Biconnected Outerplanar Graphs

Next we consider outerplanar graphs, but first we inspect biconnected outerplanar graphs, which are cycles with non-intersecting chords. Every vertex with degree 2 must have an additional edge and therefore we walk from an arbitrary startvertex v_1 around the outerface to recognize the chains, i.e., simple paths in which all vertices have degree 2. Every biconnected outerplanar graph has at least two vertices v, with $\deg(v) = 2$ [13]. Let F be an internal face of the outerplanar graph, containing two chains C_1, C_2 of length c_1, c_2, respectively and assume $c_1 \geq c_2$. Then we first add c_2 edges between the two chains, and we add edges between the remaining vertices of C_1, until one or two vertices remain unmatched. If there is only one chain C_1, then we only add edges between vertices of C_1, until one or two remain unmatched. This can easy be done while preserving planarity. When there are more than two chains, belonging to one face, then a similar technique can be applied such that all unmatched vertices belong to one chain. Next we number the vertices $v_1, \ldots, v_n$ around the outerface. We also number all vertices of degree 2 by $v(1), \ldots, v(p)$

around the outerface, visiting in order of increasing v_i-number. To get G triconnected, we apply a similar technique as in section 3: when we visit a vertex $v(k)$ of degree 2, then we test if the "subtree of v_i" is triconnected. In this case we test for each chord (α, β) (an edge, not on the outerface of the outerplanar graph) if there are two edges from one side of (α, β) to the other side. The algorithm can now be described as follows. Assume $v_1 = v(1)$ is a vertex of degree 2.

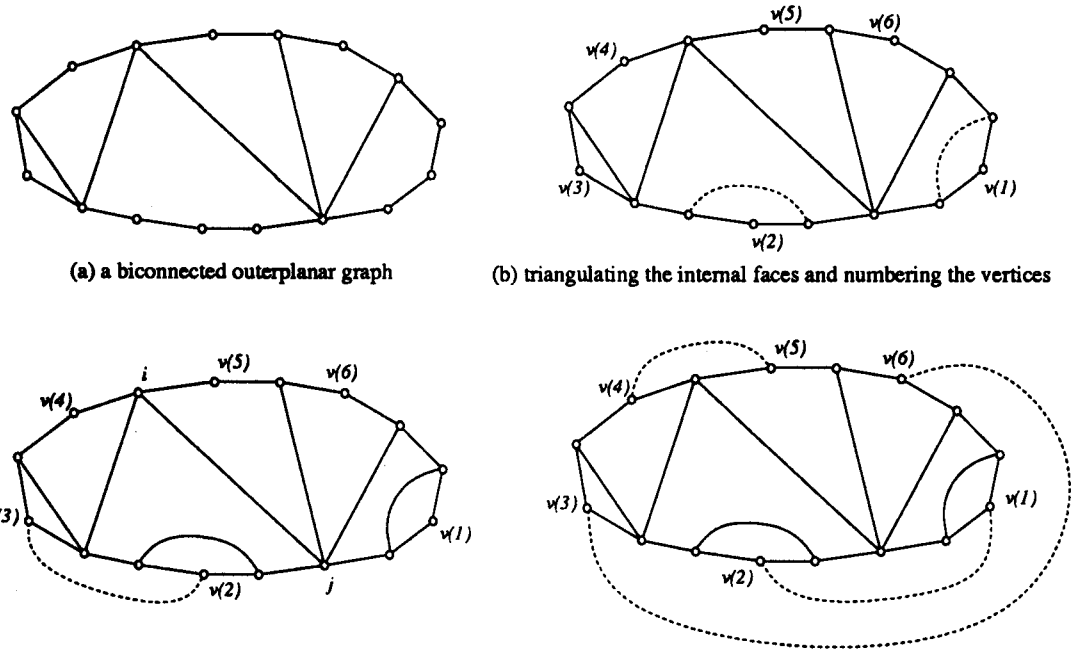
TRICONNECT_BIC_OUTERPLANAR
  $A := \emptyset$; { A becomes the augmenting set of edges. }
  $x := 1$;
  for $i := 2$ **to** n **do**
    if $v_i = v(k)$ for some k **then**
      if k is odd **then** $A := A \cup \{v(k-1), v(k))\}$
    else
      for all chords (v_j, v_i) with $1 \leq j < i$ **do**
        let $v(j_1), \ldots, v(j_p)$ be vertices of degree 2 in G between v_j and v_i;
        if j_1 is even **and** $x < j_1$ **then**
          $A := A - \{(v(j_1), v(j_2))\} \cup \{(v(x), v(j_1))\}$; $x := j_2$;
        if j_p is odd **then**
          $A := A - \{(v(j_{p-1}), v(j_p))\} \cup \{(v(x), v(j_{p-1}))\}$; $x := j_p$;
      endfor
  endfor;
  if $x < p'$ **then** $A := A \cup \{(v(x), v(p))\}$ **else** $A := A \cup \{(v(1), v(p))\}$

Lemma 6.3 *We can augment a biconnected outerplanar graph G by a minimum number of edges such that the augmented graph G' is triconnected and planar.*

Proof: Every vertex of degree 2 get one augmenting edge, plus we have one extra edge if the number of vertices of degree 2 is odd, hence the set of augmenting edges is optimal.

The structure of the algorithm TRICONNECT_BIC_OUTERPLANAR is equal to the algorithm BRIDGE-CONNECT, and by a similar proof as in lemma 4.4 it can be shown that $G' = (V, E \cup A)$ is planar.

Regarding the triconnectivity, inspect two vertices $v_i, v_j \in G'$, with $i > j$. $G' - \{v_i, v_j\}$ splitst the outercycle C in two paths P_1, P_2. If v_i and v_j belong to two different faces in G, then there is a chord in G, connecting the two paths P_1, P_2, hence $G' - \{v_i, v_j\}$ is still connected. Assume now that v_i and v_j share a common face. If v_i and v_j are in the same chain, one easily sees that $G' - \{v_i, v_j\}$ is connected. So assume v_i, v_j are on different chains. Let (v_k, v_l) be a chord in G with $k \leq j$ as large as possible and $l \geq j$ as small as possible. It is well known that for every chord (v_k, v_l) in a biconnected graph, there is a vertex v_α of degree 2, with $k < \alpha < l$. Thus the set vertices $\{v(j_1), \ldots, v(j_p)\}$ of degree 2 in G between v_j and v_i is not empty. If j_1 is even and $x < j_1$, then by adding $(v(x), v(j_1))$ to A, there is an edge from one side of (v_k, v_l) to the other side. Similar if j_p is odd then again via the



(a) a biconnected outerplanar graph

(b) triangulating the internal faces and numbering the vertices

(c) applying TRICONNECTED_BIC_OUTERPLANAR when visiting chord (i, j).

(d) the complete triconnected planar graph.

Figure 6: Example of the algorithm TRICONNECTED_BIC_OUTERPLANAR.

edge $(v(j_p), v(j_p + 1))$ there will be an edge from one side of (v_k, v_l) to the other side. If there is a vertex v_k between $v(j_1)$ and $v(j_p)$ on the outerface, then $v(j_1)$ and $v(j_p)$ do not belong to the same face, because we first added edges in one face such that the remaining vertices of degree 2 belong to one side. Since $v(j_1)$ and $v(j_p)$ have degree 2 and do not belong to one internal face, there is an edge (α, β), with α between $v(j_1)$ and $v(j_p)$ and β between $v(j_1)$ and v_k (or between $v(j_p)$ and v_l).

Since $\beta \neq v(j_1), v_j \neq \beta$ or $v_j \neq v(j_1)$ holds, say $v_j \neq v(j_1)$. But now there is a path P between P_1 and P_2 in $G' - \{v_i, v_j\}$. and via (α, β) and v_k, because $v_i \neq \alpha$. □

In figure 6 an example of the algorithm TRICONNECTED_BIC_OUTERPLANAR is given.

6.3 Triconnecting Outerplanar Graphs

To triconnect outerplanar graphs G, which are not necessarily biconnected, we apply the techniques for triconnecting trees and biconnected outerplanar graphs. If G is not connected, then we can apply the algorithm of section 3 to connect the components with each other, so we may assume that G is connected.

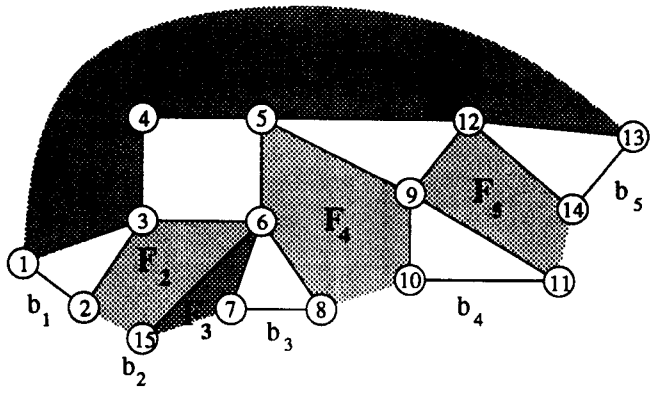


Figure 7: Example of the areas of the outerplanar graph of figure 2.

We now recognize the biconnected components of G and add edges between vertices of chains, sharing a common face inside one biconnected component of the outerplanar graph, such that only one or two vertices remain unmatched in each internal face and (if possible) part of the same side. To be more precise about these biconnected components, which must receive an augmenting edge, we state the following lemma, which can be verified quite easily.

Lemma 6.4 *A maximal biconnected component B_i in an outerplanar graph B_i needs at least one incoming edge in any triconnectivity augmentation if and only if there is vertex of degree 2 in B_i or its corresponding b-vertex has degree 2.*

We now build the bc-tree T of G. Let one arbitrary b-vertex b^* or c-vertex c^* be the root. Again the order of the children of a b-vertex is fixed. We assume that the left-to-right-order of the children of a b-vertex corresponds with walking counter-clockwise around the outerface of the coresponding block. We eliminate all c-vertices of degree 2 in T, which will have no influence on the structure of the graph. Let $l_0, \ldots, l_{L-1}$ be the leaves of T, visited from left to right, corresponding with the pendant blocks $b_0, \ldots, b_{L-1}$ in G. We introduce the following definition:

Definition 6.1 *An area A_i in G is the face which we obtain by adding an edge between the pendants b_{i-1} and b_i.*

Since the graph G is outerplanar, every vertex belongs to at least one area. If a vertex belongs to two areas A_i, A_j, then we assign it to the area A_i for which i is as small as possible. In figure 7 an example of the definition of area is given.

For every leaf l_i in T we define $V(l_i)$ as the set of vertices v in G with degree 2, for which v is assigned to area F_i. If a maximal biconnected component contains no vertices of degree 2, but its corresponding b-vertex has degree 2, then we assign one

vertex v of B_i to $V(l_i)$, if v is on the outerface of B_i and assigned to area A_i (see lemma 6.4).

Notice that if G is a tree, then this assigning to $V(l_i)$ corresponds with the definition of the leftmost descendant leaf. We apply a preorder v_i-numbering on the vertices of G and we sort each set $V(l_i)$ in order of increasing v_i-number. We now apply the algorithm TRICONNECT_BIC_OUTERPLANAR, but we visit all vertices of $V(l_i)$, and we test for chords (v_j, v_i), if both v_j and v_i belong to area F_i. We do not add the last edge to A, which means that from every set $V(l_i)$, one or two vertices remain unmatched. To augment these vertices, we change $V(l_i)$ by the set of unmatched vertices in it, hence now $|V(l_i)| \leq 2$. Let $A = \{(l_i, l_{i+1}) | 0 \leq i < L\}$, then we apply CHAIN-EDGES and additionally we apply CHANGE-SETS to augment every vertex of degree 2 by an extra edge. Every leaf l_i in T receives two extra edges. We assign these edges to those vertices of the corresponding block b_i in G, which have degree ≤ 2. Notice that every outerplanar block has at least two vertices with degree 2, hence these vertices always exist.

Thus every vertex of degree 2 and every remaining b-vertex receives one extra edge and every pendant receives two extra edges. Thus the set of augmenting edges is optimal. Moreover, we can use the proofs of lemma 5.2 and 5.3 to show that the augmented graph indeed is triconnected. The algorithms can easily be implemented to run in linear time and space, thereby completing the following main result of this section:

Theorem 6.5 *There is a linear time and space algorithm to augment an outerplanar graph G by a minimum number of edges such that the resulted graph is triconnected and planar.*

Acknowledgements

The author wishes to thank Tsan-sheng Hsu for some useful comments and for pointing out an error in an earlier version.

References

[1] Booth, K.S., and G.S. Lueker. Testing for the consecutive ones property, interval graphs and graph planarity testing using PQ-tree algorithms, *J. of Comp. and System Sciences* 13 (1976), pp. 335–379.

[2] Eswaran, K.P., and R.E. Tarjan. Augmentation problems, *SIAM J. Comput.* 5 (1976), pp. 653–665.

[3] Frederickson, G.N., and J. JaʼJa. Approximation algorithms for several graph augmentation problems, *SIAM J. Comput.* 10 (1981), pp. 270–283.

[4] Frank, A., Augmenting graphs to meet edge-connectivity requirements, *Proc. 31th Annual IEEE Symp. on Found. on Comp. Science*, St. Louis, 1990, pp. 708–718.

[5] Harary, F., *Graph Theory*, Addison–Wesley Publ. Comp., Reading, Mass., 1969.

[6] Hsu, T., On four-connecting a triconnected graph, in: *Proc. 33st Annual IEEE Symp. on Found. of Comp. Science*, Pittsburgh, 1992 (to appear).

[7] Hsu, T., and V. Ramachandran, A linear time algorithm for triconnectivity augmentation, in: *Proc. 32th Annual IEEE Symp. on Found. on Comp. Science*, Porto Rico, 1991.

[8] Hsu, T., and V. Ramachandran, On finding a smallest augmentation to biconnect a graph, In: *Proc. of the Second Annual Int. Symp. on Algorithms*, Lecture Notes in Comp. Science 557, Springer-Verlag, 1992, pp. 326–335.

[9] Kant, G., Drawing planar graphs using the *lmc*-ordering, in: *Proc. 33st Annual IEEE Symp. on Found. of Comp. Science*, Pittsburgh, 1992 (to appear).

[10] Kant, G., and H.L. Bodlaender, Triangulating planar graphs while minimizing the maximum degree, in: O. Nurmi and E. Ukkonen (Eds.), *Proc. 3rd Scand. Workshop on Algorithm Theory (SWAT'92)*, Lecture Notes in Comp. Science 621, Springer-Verlag, 1992, pp. 258–271.

[11] Kant, G., and H.L. Bodlaender, Planar graph augmentation problems, Extended Abstract in: F. Dehne, J.-R. Sack and N. Santoro (Eds.), *Proc. 2nd Workshop on Data Structures and Algorithms*, Lecture Notes in Comp. Science 519, Springer-Verlag, 1991, pp. 286–298.

[12] Khuller, S., and R. Thurimella, Approximation algorithms for graph augmentation, in: *Proc. 19th Int. Colloquium on Automata, Languages and Programming (ICALP'92)*, Lecture Notes in Comp. Science 623, Springer-Verlag, 1992, pp. 330–341.

[13] Mitchell, S.L., Linear algorithms to recognize outerplanar and maximal outerplanar graphs, *Inform. Process. Lett.* 9 (1979), pp. 229–232.

[14] Naor, D., D. Gusfield and C. Martel, A fast algorithm for optimally increasing the edge-connectivity, in: *Proc. 31st Annual IEEE Symp. on Found. of Comp. Science*, St. Louis, 1990, pp. 698–707.

[15] Read, R.C., A new method for drawing a graph given the cyclic order of the edges at each vertex, *Congr. Numer.* 56 (1987), pp. 31–44.

[16] Rosenstiehl, P., and R.E. Tarjan, Rectilinear planar layouts and bipolar orientations of planar graphs, *Discr. and Comp. Geometry* 1 (1986), pp. 343–353.

[17] Rosenthal, A., and A. Goldner, Smallest augmentations to biconnect a graph, *SIAM J. Comput.* 6 (1977), pp. 55–66.

[18] Lin, Y.-L., and S.S. Skiena, *Complexity Aspects of Visibility Graphs*, Manuscript, Dept. of Comp. Science, State Univ. of New York, Stony Brook, 1992.

[19] Tarjan, R.E., Depth-first search and linear graph algorithms, *SIAM J. Comput.* 1 (1972), pp. 146–159.

[20] Tarjan, R.E., A note on finding the bridges of a graph, *Inform. Process. Lett.* 2 (1974), pp. 160–161.

[21] Tutte, W.T., Convex representations of graphs, *Proc. London Math. Soc.*, vol. 10 (1960), pp. 304–320.