On Triangulating Planar Graphs under the Four-Connectivity Constraint

T. Biedl, G. Kant and M. Kaufmann

UU-CS-1995-27
August 1995

On Triangulating Planar Graphs under the Four-Connectivity Constraint

T. Biedl, G. Kant and M. Kaufmann

Technical Report UU-CS-1995-27
August 1995

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

ISSN: 0024-3275

On Triangulating Planar Graphs under the Four-Connectivity Constraint

Therese Biedl[*] Goos Kant[†] Michael Kaufmann[‡]

Abstract

Triangulation of planar graphs under constraints is a fundamental problem in the representation of objects. Related keywords are graph augmentation from the field of graph algorithms and mesh generation from the field of computational geometry. We consider the triangulation problem for planar graphs under the constraint to satisfy four-connectivity. A four-connected planar graph has no separating triangles, i.e. cycles of length 3 which are not a face.

We show that triangulating embedded planar graphs without introducing new separating triangles can be solved in linear time and space. If the initial graph had no separating triangle, the resulting triangulation is four-connected. If the planar graph is not embedded then deciding whether there exists an embedding with at most k separating triangles is NP-complete. For biconnected graphs a linear time approximation which produces an embedding with at most twice the optimal number is presented. With this algorithm we can check in linear time whether a biconnected planar graph can be made 4-connected while maintaining planarity. Several related remarks and results are included.

1 Introduction

The problem of augmenting a graph to reach a certain connectivity requirement by adding edges has important applications in network reliability [6, 22] and fault tolerant computing. The general version of the augmentation problem is to augment the input graph to reach a given connectivity requirement by adding

[*]RUTCOR, P.O. Box 5062, New Brunswick, NJ 08903–5062, therese@rutcor.rutgers.edu

[†]Department of Computer Science, Utrecht University, Padualaan 14, 3584 CH Utrecht, the Netherlands, goos@cs.ruu.nl. Research was supported by the ESPRIT Basic Research Actions program of the EC under contract No. 7141 (project ALCOM II).

[‡]Wilhelm-Schickard-Institut für Informatik, Universität Tübingen, Sand 13, 72026 Tübingen, Germany, mk@informatik.uni-tuebingen.de.

a smallest set of edges. Recent papers present linear time augmentation algorithms to admit the 2-connectivity constraint [5, 21, 12], and the 3-connectivity constraint [11]. With respect to 4-connectivity Kanevsky et al. [13] presented an $\mathcal{O}(n \cdot \alpha(m,n) + m)$ time algorithm for testing 4-connectivity, and Hsu presented an $\mathcal{O}(n \cdot \alpha(m,n) + m)$ time algorithm to compute the minimal set of edges to augment a 3-connected graph to a 4-connected graph [10] (here $\alpha(m,n)$ is the functional inverse of Ackermann's function). Kant described several algorithms for the augmentation problem with the additional constraint of planarity [15]. This problem has important applications in planar network design and graph drawing algorithms.

In this paper we consider the problem of triangulating a planar graph while achieving 4-connectivity. Notice that this differs significantly from the papers mentioned above, where the objective is to find a minimum set of augmenting edges to reach a certain connectivity constraint. During the last years 4-connected planar graphs received new attention due to their important characteristics: every 4-connected planar graph is hamiltonian, it can be drawn as a visibility representation in a very compact way [17], and if it is triangular it can be represented by a rectangular dual [1, 18]. Visibility representations and rectangular duals are widely used drawing representations, e.g. in industrial environments where rectangular duals are used in floor-planning problems [19].

Unfortunately, not every planar graph can be triangulated with the additional constraint of 4-connectivity. If a planar graph contains a cycle of length 3 which is not a face this is called a *separating triangle.* No graph containing a separating triangle can be made 4-connected while maintaining planarity. Also the *star graph* (the graph consisting of an $n-1$-cycle and one more vertex connected to all other vertices) does not contain a separating triangle, but for $n \geq 5$ any triangulation of it does. So this graph again cannot be made 4-connected.

We present a linear time and space algorithm that, given an embedded planar graph G which does not contain a star graph in some sense, triangulates G without introducing new separating triangles. If the initial graph does not contain a separating triangle the output graph is a 4-connected triangular planar graph.

If the initial planar graph is not embedded the number of initial separating triangles depends on the chosen embedding. We show that it is NP-complete to decide whether a biconnected planar graph can be embedded such that the number of separating triangles is at most k. On the positive side, we present a linear time algorithm that embeds a planar graph such that the number of separating triangles is at most twice the optimum. In particular, it can be found in linear time whether a biconnected planar graph can be embedded without separating triangles at all, and whether it can be made 4-connected.

If this is the case we can make the graph 4-connected and then apply the rectangular dual algorithm of [1, 18]. Another application are straight-line drawings: Very recently, Xin He [9] presented a linear-time algorithm to draw triangulated 4-connected planar graphs with straight lines on a grid of size at most $\frac{1}{2}(n+3) \times \frac{2}{3}(n-1)$, thereby considerably improving the general bound of $(n-2) \times (n-2)$. Our algorithm can be used to extend this results to all biconnected graphs that can be made 4-connected.

If our algorithm gives an embedding of G with s separating triangles then a visibility representation of G can be drawn on a grid of size at most $(n+s-1) \times (n-1)$. For small s this improves the general bound of $(\lfloor\frac{3}{2}n\rfloor - 3) \times (n-1)$ [14].

The paper is organized as follows. In Section 2 we give some necessary definitions. In Section 3 we present the linear time algorithm for triangulating an embedded planar graph without introducing separating triangles. In Section 4 we consider the problem of embedding planar graphs such that the number of separating triangles is minimum, and in Section 5 we give an approximation for this minimization problem. Section 6 shortly deals with the problem of testing 4-connectivity of planar graphs, and gives some concluding remarks.

2 Definitions

In this paper all graphs are assumed to be simple, i.e. without loops and multiple edges. A graph is called *planar* if it can be drawn in the plane with edges intersecting only at vertices, i.e. without edge crossing. A *planar embedding* is a representation of a planar graph in which the edges incident to a vertex are given in clockwise order with respect to a planar drawing. The embedding divides the plane into *faces.* The unbounded face is the *exterior face* or *outerface.* A cycle of length 3 is a *triangle.* A planar graph is *triangular* if every face is a triangle. A triangular planar graph has $3n-6$ edges and adding any other edge destroys planarity.

Let $G=(V,E)$ be a planar graph with n vertices. A cycle C of G divides the plane into its interior and exterior region. If C contains at least one vertex in its interior and at least one vertex in its exterior, it is called a *separating cycle.* A graph G is called k-connected, if deleting any $k-1$ vertices does not disconnect G. A 2-connected graph is also called *biconnected,* a 3-connected graphs is also called *triconnected.* A "disconnecting" set of k vertices is called a *separating k-set.* Separating 1-sets and 2-sets of vertices are called *cutvertices* and *separation pairs*, respectively. It is well-known that a planar graph is at most 5-connected and every triangular planar graph is at least triconnected.

A graph is said to *contain a star with central vertex w* at face F if all vertices v incident to F we have $v=w$ or $(v,w) \in E$; and there are at least four vertices other than w incident to F. Note that only if w belongs to F we can add an edge in F without producing a separating triangle. But even then F is not a triangle afterwards, and we cannot triangulate it without introducing a separating triangle. So any graph that contains a star cannot be made 4-connected.

3 Triangulating Embedded Planar Graphs

Assume that G is an embedded biconnected planar graph. We will show that unless G contains a star it can be triangulated without adding separating triangles. The idea is to take a vertex v with maximum degree and to add edges in an incident non-triangulated face. We will show later that while triangulating a face we do not create separating triangles and stars. For the algorithm we assume that a fixed embedding of G is given by the adjacency lists and that G does not contain a separating triangle.

TRIANGULATE
Input: An embedded planar biconnected graph G without separating triangle
Output: A triangulation of G without separating triangle if possible

```
(1) while there are nontriangular faces
(2) do
(3)     choose $v \in V$ such that $v$ has maximum degree among all vertices
(4)         which are part of a nontriangular face
(5)     let $F$ be a nontriangular face with vertices $v, u_1, u_2, ..., u_p, u_{p+1} = v$
(6)         (in clockwise or counterclockwise order so that $deg(u_1) \geq deg(u_p)$)
(7)     if $u_1$ and $u_p$ have no common neighbor but $v$
(8)         or if $p = 3$ and $u_1$ and $u_p$ have no common neighbor but $v$ and $u_2$
(9)     then add $(u_1, u_p)$
(10)    else let $w$ be a common neighbor of $u_1$ and $u_p$
(11)        determine minimal $j > 1$ such that $u_j$ is not adjacent to $w$
(12)        if such $j$ does not exist
(13)        then STOP, $G$ contains a star with central vertex $w$ at face $F$
(14)        if $u_j = v$
(15)        then add edges from $v$ to $u_2, \ldots, u_{p-1}$
(16)        else add edges from $u_j$ to $u_1, \ldots, u_{j-2}$
(17)            if $j = p - 1$ or $(u_j, u_p) \notin E$ then add $(v, u_j)$
(18)            else determine maximal $k < p$ such that $u_k$ is not adjacent to $u_j$
(19)                if $k = j$ then set $k = j + 1$.
(20)                add the edge $(u_k, u_1)$
(21)                add the edges from $v$ to $u_k, \ldots, u_{p-1}$
(22) od
```

In the remaining part of this section we will evaluate the running time of the algorithm, and then prove its correctness.

Lemma 3.1 *j can be found in $\mathcal{O}(j)$ time.*

Proof: Assume we have a procedure j-FIND(l) that returns the minimum index $j > l$ such that w is not adjacent to u_j (so in line (11) we make a call to

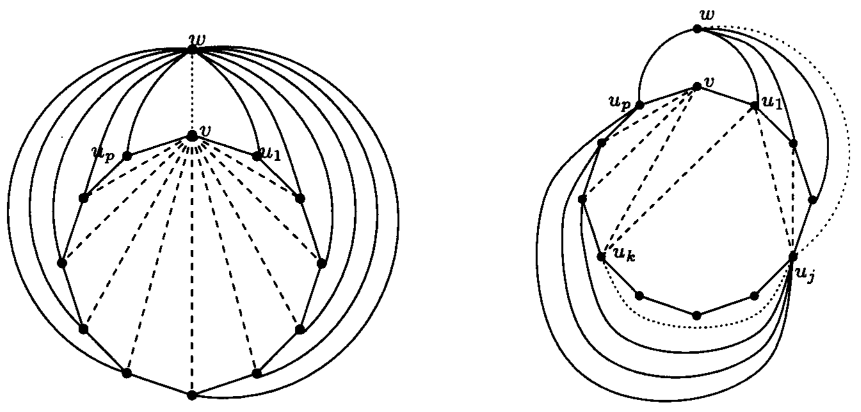

Figure 1: Two cases of the algorithm (line (15) resp. line (16),(18)-(21)). Nonexistent edges are dotted, added edges are dashed.

j-FIND(1)). Furthermore, we assume that when calling this procedure we know the edge (u_l, w) and its position in the adjacency list $N(w)$ of w.

If (w, u_{l+1}) is an edge, it must be the next element after (w, u_l) in the counterclockwise order of $N(w)$. Otherwise $\{w, u_l, u_{l+1}\}$ is a separating triangle, and we assumed G to be without separating triangles. So if this element is not (w, u_{l+1}), we can return the value $l + 1$ and are done. Otherwise, we return the value of j-FIND(l+1). Each call of j-FIND() needs only $\mathcal{O}(1)$ time, completing the proof. □

Testing whether (u_j, u_p) exists can be done by checking the adjacency list of u_p, which requires $\mathcal{O}(deg(u_p))$ time. This also gives us the starting point in the adjacency list of u_j to find the maximum index $k < p$ such that u_k is not adjacent to u_j. Similarly, u_k can be found in $\mathcal{O}(p - k)$ time.

Lemma 3.2 *Finding the next v can be done in overall linear time.*

Proof: Throughout the algorithm we will keep a doubly linked list L of vertices which are possible candidates for the next v. The entries of L are sorted by descending degree. We also keep an array A where $A[i]$ indicates the first vertex in L of degree i. We initialize L and A by performing bucket sort on the degree of the vertices which requires $\mathcal{O}(n)$ time. During the algorithm the degrees of vertices will change. We will show how to update L and A in $\mathcal{O}(1)$ time when increasing one degree by 1. Hence we need $\mathcal{O}(\#\{\text{added edges}\}) = \mathcal{O}(n)$ time.

Assume vertex x has its degree increased from i to $i+1$. The new position of x in L is before $A[i]$. We delete x from its old position and insert it before $A[i]$ (note that if $x = A[i]$ then L remains unchanged). If $A[i+1]$ is defined, it remains unchanged. Otherwise, set $A[i+1] = x$. If $A[i] \neq x$, $A[i]$ remains unchanged. Otherwise, let $A[i]$ be the successor of x in L if this successor has degree i and leave it undefined otherwise. Our next candidate v is the first element of L. When deleting v, we set $A[deg(v)]$ to be v's successor in L if its degree equals

$deg(v)$ and to be undefined otherwise. These operations each need $\mathcal{O}(1)$ time, so we need $\mathcal{O}(n)$ time overall. □

Lemma 3.3 *The above algorithm works in linear time and space.*

Proof: Let v and F be fixed, and consider the steps (7)-(21). We have to search for the vertex w adjacent to both u_1 and u_p and can do so in $deg(u_1) + deg(u_p) \leq 2 \cdot deg(u_1)$ time. So all steps need $\mathcal{O}(deg(u_1) + j + (p - k))$ time. We added $\Omega(j + (p - k))$ edges. By definition of v we know $deg(u_1) \leq deg(v)$. So we can consider $deg(u_1)$ as the minimum degree among the endpoints of the edge (v, u_1).

After these steps the edge (v, u_1) is contained in a triangular face which was not a triangular face before. This can happen at most twice to every edge. Also the degree of a vertex never decreases and therefore we have a running time of

$$\mathcal{O}\Big(\#\{\text{added edges}\} + \sum_{(v,w)\in E^*} \min\{deg(v), deg(w)\}\Big) = \mathcal{O}(n),$$

where E^* is the set of edges in the final graph and the last equality is due to Chiba and Nishizeki [2]. □

Now, we prove the correctness of the algorithm:

Lemma 3.4 *The graph stays simple.*

Proof: Notice that the edge (u_1, u_p) does not exist. Assume it did. Then v must have degree 2, otherwise we had a separating triangle. But u_1 has degree at least 3, which contradicts the choice of v. So we know that in line (9) of the algorithm we do not produce a double edge. In all other cases the introduced edges cannot have existed previously by planarity. □

Lemma 3.5 *No separating triangles are introduced.*

Proof: Separating triangles can be introduced only when we add an edge where the two endpoints have a common neighbor. We will show that in all cases this either does not happen, or that the new edge gives a triangle that is a face.

- Line (9): By definition u_1 and u_p have no common neighbor except those that form a face when adding the edge (u_1, u_p).
- Line (15): By definition v is the only vertex on F not adjacent to w. By planarity u_l for $2 < l < p - 1$ has no common neighbor with v. u_2 and u_{p-1} do have a common neighbor with v, but the introduced triangles are faces.
- Line (16): Remember that the edge (u_1, u_p) does not exist. So for $1 \leq l < j - 2$, the only possible common neighbor of u_j and u_l is w, but w is not adjacent to u_j. For $l = j - 2$, the vertices u_l and u_j have the common neighbor u_{j-1}, but $\{u_l, u_{j-1}, u_j\}$ forms a face afterwards.
- Line (17): Possible common neighbors of v and u_j are w, u_p and u_1 (if $j = 2$). But u_j is not adjacent to w. It is not adjacent to u_p if $j < p - 1$ and otherwise $\{v, u_p, u_{p-1}\}$ forms a face. If $j = 2$ then $\{v, u_1, u_j\}$ is a face.

- Line (20): Because of the edge (u_j, u_p) and the fact that $1 < j < k < p$, the only possible common neighbors of u_1 and u_k are u_j and u_p. But u_1 is not connected to u_p, and, by definition, u_k is not connected to u_j, unless $k = j + 1$. In the last case, $\{u_k, u_j, u_1\}$ forms a face afterwards.
- Line (21): Consider the face containing v that we get after all edges up to line (20) have been added. Rename it $x_0, x_1, \ldots, x_r$ such that $x_0 = u_k$, $x_{r-1} = v$ and $x_r = u_1$. Now x_1 and x_r have a common neighbor, u_j, and $r - 1$ is the smallest index j' such that $x_{j'}$ is not connected to this common neighbor. The argument now is a repetition of that for line (16) and (17).

$\square$

Lemma 3.6 *The added edges do not create a star if* $n \geq 6$.

Proof: Assume that our algorithm creates a star. Consider the step when we add for the first time a star, i.e. we have a face F_1 and a vertex w such that after the step we have a star with central vertex w at F_1. Therefore, all vertices on F_1 but one, say u, are either w or are adjacent to it and F_1 contains at least four vertices that are not w. Furthermore, the algorithm chose a face F that contains both u and w and adds the edge (u, w) in this step.

Let $u, x_1, \ldots, x_r$ be the vertices on F_1 in clockwise order (note that $w \neq x_1, x_r$). So u and w have the common neighbors x_1 and x_r. Remember that our algorithm never adds a separating triangle. So adding the edge (u, w) means that the resulting triangles $\{u, x_1, w\}$ and $\{u, x_r, w\}$ are faces afterwards. So the face F must have consisted of the four vertices u, x_1, w, x_r. In particular, $F \neq F_1$, since F_1 had at least four vertices that were not w.

Note that the three vertices x_1, u, x_r belonged to two faces, F and F_1, and therefore $deg(u) = 2$ and u has no other neighbor. Can x_r have a neighbor other than u, w, x_{r-1}? Since $(x_r, w), (x_r, u)$ both belong to F and $(x_r, u), (x_r, x_{r-1})$ both belong to F_1, such a neighbor would have to be between x_{r-1} and w in the adjacency list of x_r. But then $\{x_r, x_2, w\}$ is a separating triangle, a contradiction. Similarly no other x_i can have a neighbor other than $\{w, x_{i-1}, x_{i+1}\}$. Therefore, by biconnectivity, G consists only of the vertices $\{w, u, x_1, \ldots, x_r\}$.

So w is adjacent to all vertices but u and therefore $deg(w) = n - 2 \geq 4$. Every vertex $\neq w$ on F has at most three neighbors, therefore w was chosen to be v in the algorithm. But x_1 and x_r have no common neighbor (since $n \geq 6$ we have $r \geq 4$), and therefore the edge (x_1, x_r) will be added, and not edge (u, w). $\square$

For a graph to contain a star, it must have at least five vertices. So we have to include into our algorithm the special case of $n = 5$ which falls into a few easily handled cases. This concludes the correctness proof.

Theorem 3.7 *Let G be a planar graph with a fixed embedding. TRIANGULATE triangulates G without introducing separating triangles if and only if G does not contain a star. It does this in linear time and space.*

Proof: Nothing needs to be shown for biconnected graphs without separating triangles. So assume first that we have a biconnected graph G with separating triangles. Split G at its separating triangles (i.e., if $\{u, v, w\}$ forms a separating triangle we split G at the three edges $(u, v), (v, w)$ and (w, u) and add these edges to both components). We repeat this process until there are no separating triangles left and then apply TRIANGULATE to the subgraphs.

This can be achieved in linear time and space, using an initial list with pointers to the separating triangles in the graph (see also [14] for some details). This does not introduce a separating triangle iff none of the subgraphs contains a star. But this is the case iff G does not contain a star. Our algorithm never adds a star, so G is never made to contain a star throughout the algorithm.

If G is not biconnected, we add edges without adding separating triangles with the algorithm of Read [20] which we sketch here: Let v be a cutvertex and let u and w be two consecutive neighbors of v, belonging to different biconnected components. Add the edge (u, w) to G. Since u and w only share v among their neighbors, and $\{u, v, w\}$ is a face afterwards, this does not introduce a separating triangle. G now has one biconnected component less and is still planar. Repeating this procedure yields a biconnected planar graph in linear time. For this graph the claim has been shown. □

4 Embedding Graphs for Triangulation

In the next two sections we consider the problem of triangulating a planar graph G when no embedding is given in advance. If G is triconnected, the embedding is unique. Otherwise changing the embedding can vary the number of separating triangles. More precisely we have the following theorem.

Theorem 4.1 *Given a biconnected planar graph G, it is NP-complete to decide whether G can be embedded such that the number of separating triangles of G is at most k.*

Proof: The problem is in NP. Given a planar embedding of G we can count the number of separating triangles in the embedding in polynomial time (see [2]).

Let G be an arbitrary triangular planar graph. For every edge $(a, b) \in G$, we add a vertex x with edges to a and to b. Let G' be the resulting graph. Clearly G' is biconnected and planar. Let F and F' be the two faces incident to (a, b) in G. If we place x inside F then F is a separating triangle in G'. However, x must be embedded in either F or F', i.e. either F or F' must be a separating triangle in G'.

Let S be a minimum set of faces in G such that for every edge $(a, b) \in G$, at least one incident face belongs to S. S corresponds precisely to the minimum number of triangles in G' which are separating (place x in the face which belongs to S). The set S is a vertex cover in the dual graph G^* of G. Since G^* is a cubic triconnected planar graph and deciding whether there exists a vertex cover of size at most k is NP-hard for cubic triconnected planar graphs (cf. the following theorem), the problem of deciding whether a biconnected planar graph G can be embedded with at most k separating triangles is NP-complete. □

Theorem 4.2 *Vertex cover in cubic triconnected planar graphs is NP-hard.*

Proof: In the papers [7] and [8] it has already been shown that "vertex cover in planar graphs with maximum degree 3" is NP-hard. In the following, we show how we can make such a graph cubic and increase the connectivity until it is 3-connected while maintaining an equivalent vertex cover problem.

Operation 1 *Let v be a vertex of degree 1 with neighbor w. Delete v and replace it by a 4-cycle $\{v_1, v_2, v_3, v_4\}$ where v_2 is also connected with v_4. Add the edge (v_1, w).*

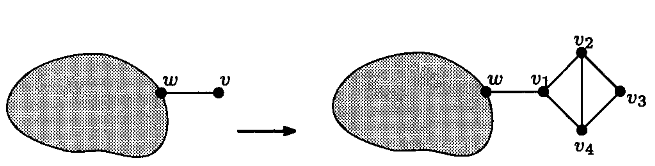

Figure 2: Removing a degree 1-vertex.

Lemma 4.3 *Let G have a vertex v of degree 1 and let G' be the graph resulting after applying Operation 1 to v. Then G has a vertex cover of size k iff G' has a vertex cover of size $k+2$.*

Proof: Let V^* be a vertex cover for G of size k. If $v \in V^*$ we set $V^{*'} = (V^* - \{v\}) \cup \{v_1, v_2, v_3\}$. Otherwise we must have $w \in V^*$ and set $V^{*'} = V^* \cup \{v_2, v_4\}$. The size of $V^{*'}$ is $k+2$ and it is a vertex cover for G'.

Conversely let $V^{*'}$ be a vertex cover of size k'. We consider the vertices v_1, v_2, v_3, v_4 which have been added. If those four vertices contain at most two vertices in $V^{*'}$ then these must be v_2 and v_4, and also w must be in $V^{*'}$. Consequently $V^* = V^{*'} - \{v_2, v_4\}$ is a vertex cover for G of size $k' - 2$. If those four vertices contain at least three vertices then let V^* be v plus the vertices of $V^{*'}$ not in $\{v_1, v_2, v_3, v_4\}$. Then V^* is a vertex cover for G and has at most $k' - 2$ vertices. □

Operation 2 *Let v be a vertex of degree 2 with neighbors w_1, w_2. Delete v and replace it by a 4-cycle $\{v_1, v_2, v_3, v_4\}$ where v_2 is also connected with v_4. Add the edges (v_1, w_1) and (v_3, w_2).*

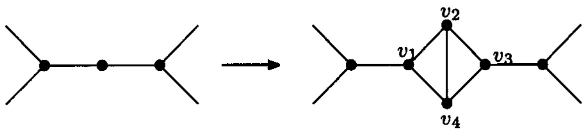

Figure 3: Removing a degree 2-vertex.

For Operation 2, an analogous lemma holds:

Lemma 4.4 *Let G have a vertex v of degree 2 and let G' be the graph resulting after applying Operation 2 to v. Then G has a vertex cover of size k iff G' has a vertex cover of size $k+2$.*

Now we show how to make the graph for the reduction triconnected.

Operation 3 *Let G be 3-regular. Replace every vertex by a cluster of 7 vertices and every edge by two edges, as shown in Figure 4.*

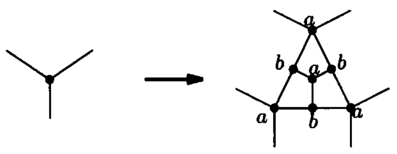

Figure 4: Increase the connectivity.

Lemma 4.5 *Let G be 3-regular with n vertices and let G' be the graph resulting after Operation 3. Then G has a vertex cover of size k iff G' has a vertex cover of size $3n+k$.*

Proof: Let V^* be a vertex cover for G of size k. We construct a vertex cover $V^{*'}$ of G' in the following way: If $v \in V^*$ we include into $V^{*'}$ the vertices marked "a" of the cluster replacing v. If $v \notin V^*$ we include the vertices marked "b" into $V^{*'}$. The size of $V^{*'}$ is $3n+k$ and this set is obviously a cover set for G'.

Conversely let $V^{*'}$ be a cover set of size k' for G'. We constuct a vertex cover V^* of G by considering each cluster that replaced $v \in G$. If $V^{*'}$ contains only three vertices out of the seven, they must be the vertices marked "b". In this case v will not be included into V^*. If $V^{*'}$ contains more than three vertices from the seven, include v into V^*. Let (u,v) be an edge of G. In one of the clusters of u and v in G' an "a"-vertex must have been included in $V^{*'}$. This cluster then contained at least four vertices of $V^{*'}$ and its vertex was in included in V^*. So the obtained set is a vertex cover, and its size is at most $k'-3n$. □

Lemma 4.6 *Let G be 3-regular and let G' be the graph resulting after Operation 3. If G was connected then G' is biconnected. If G was biconnected then G' is triconnected.*

Proof: Assume we had a path P in G between two vertices v and w. Let x be any vertex of the cluster replacing v, and let y be any vertex of the cluster replacing w. Path P now gives two disjoint paths connecting x and y in G'. This shows that G' is biconnected, provided that G was connected. For if we have two vertices within one cluster they are on a cycle. If we have two vertices in different clusters their corresponding vertices in G were connected by a path, and hence they are connected by two disjoint paths in G'.

Now let G be biconnected. Pick any two vertices x and y in G'. Assume first that they are in the same cluster, coming from the vertex v in G. There was a cycle in G containing v and it now gives two paths connecting x with y. A third disjoint path can be found within the cluster. Assume next that x and y are in different clusters. In G there were two disjoint paths connecting them. This two paths give four paths in G' which are disjoint, except at the clusters containing x and y. Using these four paths we can now find three disjoint paths between x and y. □

The resulting graph now has vertices of degree 4. In the following we will show how those vertices can be removed while maintaining an equivalent vertex cover problem.

Operation 4 *Let G have a vertex v of degree 4. Delete v and replace it by a cluster of ten vertices as shown in Figure 5.*

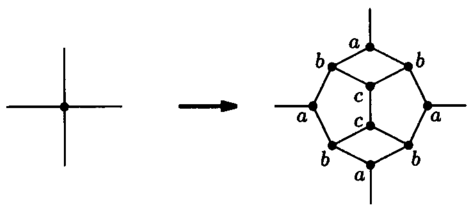

Figure 5: Removing vertices of degree 4.

Lemma 4.7 *Let G have a vertex v of degree 4 and let G' be the resulting graph after applying Operation 4 to v. Then G has a vertex cover of size k iff G' has a vertex cover of size $k + 5$.*

Proof: Let V^* be a vertex cover for G of size k. We construct a vertex cover $V^{*'}$ of G' of size $k + 5$ in the following way: If $v \in V^*$ we include the vertices marked "a" and both vertices marked "c" into $V^{*'}$. If $v \notin V^*$ we include the vertices marked "b" and one of the vertices marked "c" into $V^{*'}$. Obviously the size of $V^{*'}$ is $k + 5$ and it is easy to see that this set is a cover set for G'.

Conversely let $V^{*'}$ be a cover set of size k' for G'. We consider the cluster that replaced v. The 8-cycle formed by the "a"- and "b"-vertices must contain at least four vertices of $V^{*'}$. Moreover, at least one of the vertices marked "c" must be in the cover. So at least five vertices of the cluster must be in $V^{*'}$. One sees easily that the only way to cover all edges with five vertices is to take the "b"-vertices and one "c"-vertex. In this case all neighbors of the "a"-vertices must have been in $V^{*'}$ and v will not be included in the cover set V^* for G. If V^* contains more than five vertices of the cluster we include v in V^*. Now V^* contains at most $k' - 5$ vertices and obviously is a vertex cover for G. □

To conclude the proof of the theorem we give the whole construction of the reduction: We start with the graph by Garey and Johnson [7] with maximum degree 3. We transform it to a cubic planar graph by applying Operations 1 and 2. If it is not triconnected we increase the connectivity by one with Operation 3. Then we make the graph cubic again with Operation 4. If G is still not triconnected we apply Operations 3 and 4 once more. After that a cubic planar triconnected graph is obtained. We have reduced the problem "Find a vertex cover of prescribed size for a planar graph with maximum degree 3" to "Solve the same problem for a cubic planar triconnected graph". The first problem has been shown to be reducible to the 3-satisfiability problem [7], and hence the latter problem is NP-hard as well. □

5 An Approximation

In this section, we present a linear time algorithm to construct an embedding of a planar graph G with at most twice the optimal number of separating triangles. After the embedding is computed, we can use the algorithm of Section 3 to get a triangulation of G without new separating triangles.

We use the *SPQR-tree*, a data structure that represents the decomposition of a biconnected graph into its triconnected components [3]. The *triconnected components* of a biconnected graph G are defined as follows. If G is triconnected itself is the unique triconnected component. Otherwise let $\{u, v\}$ be a separation pair of G. We partition G into two connected subgraphs G_1 and G_2 which have only the vertices u and v in common. We continue the decomposition process recursively on $G_1' = G_1 + (u, v)$ and $G_2' = G_2 + (u, v)$ until no further decomposition is possible. The added edges are called *virtual edges*. The resulting graphs are each either triconnected simple graphs with at least four vertices, or sets of three multiple edges, or triangles. The triconnected components of G are obtained from such graphs by merging the triple bonds into maximal sets of multiple edges (*bonds*), and the triangles into maximal simple cycles (*polygons*).

The SPQR-tree T is defined as follows: for every triconnected component we create a node in T. Namely, for every polygon an S-node, for every bond a P-node, and for every triconnected graph an R-node. Moreover, we add a Q-node for every edge. Note that every edge, whether virtual or in G, belongs to exactly

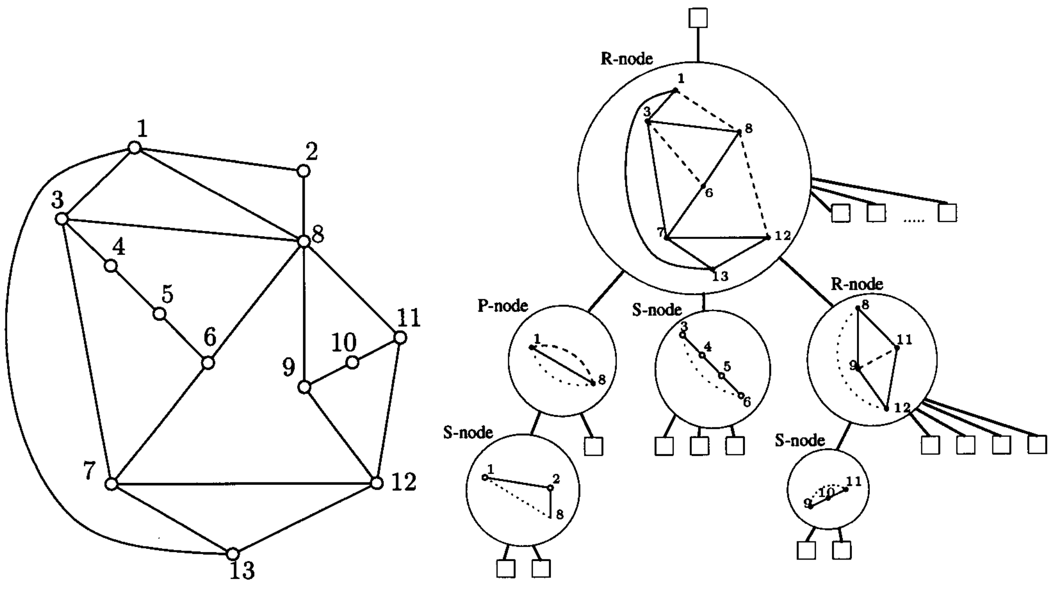

Figure 6: Example of a biconnected graph and its SPQR-tree (from [3]). Q-nodes are represented by boxes. Virtual edges are dashed in the parent-node and dotted in the child-node. Deleting dotted edges gives *skeleton*(b) for every node b.

two nodes in T. Two nodes in T are now adjacent if and only if they have either a non-virtual edge or a virtual edge added in the same step in common.

Let T be the SPQR-tree of a given biconnected planar graph G, rooted at an arbitrary Q-node. We visit the nodes of T in a bottom-up order and handle the corresponding components in this order. Assume we visit node b in T. Let b' be the parent-node of b. Denote by *skeleton*(b) the associated subgraph of G, but leaving out the virtual edge that defines the arc from b to b'. (Notice that this differs from the definition in [3], but it makes our description easier.) The two common vertices of *skeleton*(b) and *skeleton*(b'), i.e. the endpoints of the common edge, are called the *poles* of b. Note that the poles are always on the outerface of *skeleton*(b). *pertinent*(b) is defined by taking *skeleton*(b) and replacing all virtual edges e_i by *pertinent*(b_i) of the corresponding child-node b_i.

Note that for *pertinent*(b) no embedding is fixed. We will find an embedding for it and call it *pertinent'*(b). We will explain the algorithm with respect to the type of b in T. We need the following notation: Whenever we have calculated *pertinent'*(b_i) for some node b_i we have two distinguished paths between its poles u_i, v_i, namely, the two paths on the outerface of *pertinent'*(b_i). We denote these paths by $P_{i,1}$ and $P_{i,2}$ and assume that $P_{i,1}$ has not more edges than $P_{i,2}$, i.e., if we denote by $|P|$ the *length* of a path P, measured by the number of edges, then $|P_{i,1}| \leq |P_{i,2}|$.

5.1 b is a P-node

Denote by u and v the two poles of *skeleton*(b), and the set of multiple edges between u and v by $e_1, \dots, e_K$. If e_i is virtual then assume the corresponding child-node to be b_i. We want to arrange the edges in an order $e_{i_1}, \dots, e_{i_K}$ from left to right such that when replacing all virtual edges e_{i_k} with *pertinent*$'(b_i)$ we get no more than the necessary number of separating triangles. So if the edge (u, v) exists and we have a path $P_{j,1}$ of length 2, we want to place it next to (u, v).

Assume $(u, v) \in E$ and in particular $(u, v) = e_1$. Let $j_1, \dots, j_L$ be the indices for which $|P_{j_l,1}| = 2$. If $L < 2$ we set $i_1 = 1$ and $i_2 = j_1$ (if $L = 1$) and if $L \geq 2$ we set $i_1 = j_1$, $i_2 = 1$ and $i_3 = j_2$. In this case we rename the paths on the outerface of b_{j_1} and b_{j_2} in such a way that $|P_{j_1,2}| = |P_{j_2,1}| = 2$. All other i_k are assigned arbitrarily.

If (u, v) was not an edge we set $i_k = k$ for all k. We replace e_{i_k} by *pertinent*$'(b_{i_k})$ such that $P_{i_k,1}$ is placed left from $P_{i_k,2}$. Thereby we avoid introducing separating triangles at b_{j_1} and b_{j_2}. See Figure 7.

5.2 b is an S-node

If b is an S-node with poles u and v, then *skeleton*(b) is a path on the edges $e_1, \dots, e_K$. We replace any virtual e_i by *pertinent*$'(b_i)$, where b_i is its corresponding child-node, such that all paths $P_{i,1}$ are placed on one side of the path.

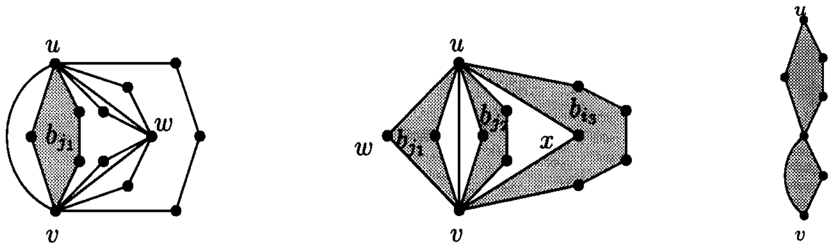

Figure 7: How to handle a P-node (2 cases) and an S-node.

5.3 b is an R-node

If b is an R-node, then *skeleton*(b) plus the virtual edge between its poles u and v is triconnected and it has a unique embedding. We fix the embedding of *skeleton*(b) accordingly. Notice that this is the only possible embedding of *skeleton*(b) with both u and v on the outerface. Let $e_1, \dots, e_K$ be the virtual edges of *skeleton*(b) and assume the corresponding child-nodes to be $b_1, \dots, b_K$. For all e_i with $|P_{i,1}| > 1$ we replace e_i by *pertinent*$'(b_i)$ immediately. Call the resulting graph $G' = (V', E')$. If there are virtual edges left in G' we compute a "dual" graph H of G' in the following way: For every triangular face F in G' we add a vertex v_F in H. An edge is added between v_F and $v_{F'}$ in H if and only if F and F' share a virtual edge e_i in G'. Note that for a virtual edge e_i in G' we have $|P_{i,1}| = 1$, i.e. $P_{i,1}$ is the edge e_i. Merging *pertinent*$'(b_i)$ such that $P_{i,1}$ belongs to F makes F' a separating triangle, and vice versa. To decide whether $P_{i,1}$ should be embedded in F or F' we compute a vertex cover S in H. At least one endpoint of $(v_F, v_{F'})$, say v_F, will be part of S, and this will become the separating face.

We replace every virtual edge e_i remaining in G' by *pertinent*$'(b_i)$. Let e_i belong to the faces F and F'. Assume both of F and F' are not in H. If one of them is the outerface then we place $P_{i,1}$ in the outerface, otherwise we place $P_{i,1}$ arbitrarily. If one of F and F', say F, is not in H then we place $P_{i,1}$ in F'. If both are in H then e_i induces an edge between them in H, and one of its endpoints, say F, must be in S. We then place $P_{i,1}$ in F'. See Figure 8 for an example.

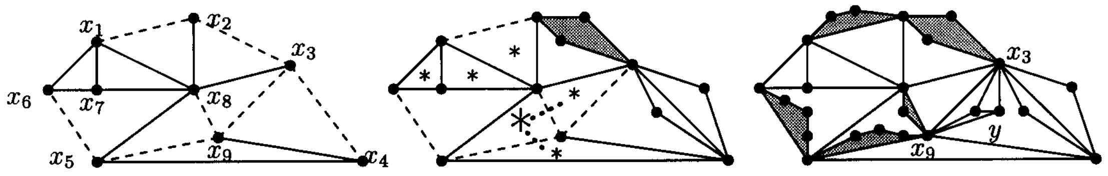

Figure 8: The embedding algorithm when b is an R-node

For the following proofs assume that when dealing with a node b we denote by P the shorter of the paths between the poles of b on the outerface of *pertinent*$'(b)$. To prove that this algorithm gives indeed a good embedding we show the invariant that for any b this path P is as short as possible. Here for something "to be unavoidable" means that we can avoid it only by increasing the number of separating triangles.

Lemma 5.1 *Let b be a P-node with poles u and v. If $(u, v) \in$ pertinent(b) then $|P| = 1$ unless unavoidable.*

Proof: If $(u, v) \in E$ it is placed at the outerface, unless there were two child-nodes b_{j_1} and b_{j_2} of b which both had $|P_{j_i,1}| = 2$. But in this case placing (u, v) on the outerface would give one more separating triangle, so $|P| > 1$ was unavoidable. □

Lemma 5.2 *Let b be an S-node or an R-node with poles u and v. If there exists a path of length 2 on the outerface of some embedding of pertinent(b) then $|P| = 2$ unless unavoidable.*

Proof: Note first that the edge (u, v) does not exist in *skeleton*(b). For assume it did. Then the triconnected component that defined b would have a double edge, and therefore b would be a P-node.

We know that there exists an embedding of *pertinent*(b) with a path of length 2 on the outerface. Then we already must have a path $P' = \{u, w, v\}$ of length 2 on the outerface of some embedding of *skeleton*(b), since replacing virtual edges never shortens a path. But for both an S-node and an R-node the embedding of *skeleton*(b) is unique, at least if we demand that both u and v be on the outerface. So we know that P' is on the outerface of *skeleton*(b), and we only have to show that it stays there in *pertinent*$'(b)$ unless unavoidable. If either of the edges of P' is non-virtual it stays on the outerface of *pertinent*$'(b)$.

So we have to deal only with the virtual edges on path P'. Consider the edge (u, w), and assume it is virtual and belongs to the child-node b_i. Then b_i is a P-node. We may assume that (u, w) is on the outerface of *pertinent*$'(b_i)$ since otherwise the extension of the path on the outerface of *pertinent*$'(b)$ was unavoidable (Lemma 5.1). For an S-node if (u, w) is on the outerface of *pertinent*$'(b_i)$ it is also on the outerface of *pertinent*$'(b)$. Moreover, the same is true for the edge (w, v) and both edges are placed on one side, so one side of *pertinent*$'(b)$ is exactly the path, unless it was unavoidable.

For an R-node note that the outerface of *skeleton*(b) is never a triangle, since the edge (u, v) does not exist. So we know that there is no vertex corresponding to the outerface in H. By the algorithm *pertinent*$'(b_i)$ is placed such that (u, w) is on the outerface, unless the face in G' on the other side of (u, w) is a triangle. In this case placing (u, w) on the outerface of *pertinent*$'(b)$ would create a new separating triangle. In Figure 8 this would happen if $u = x_1$ and $w = x_2$. So (u, w) is placed on the outerface of *pertinent*$'(b)$ unless unavoidable. The argument for (w, v) is the same, so the path remains on the outerface unless unavoidable. □

Lemma 5.3 *Handling P-nodes and S-nodes does not increase the number of separating triangles by more than the necessary number.*

Proof: Let us first deal with the easy case of an S-node b. Since *skeleton*(b) is a simple path, any triangle in *pertinent*(b) is also a triangle in some *pertinent*(b_i), so we do not create new triangles at all.

Now for P-nodes. Assume that we did produce a new separating triangle which has to consist of u and v and a third vertex, w. The path $P' = \{u, w, v\}$ belongs to some child-node b_i, which is an S-node or an R-node. P' was placed on the outerface of *pertinent*(b_i) unless unavoidable (Lemma 5.2). So if P' is not on the outerface then the new triangle can be avoided only at the expense of creating another separating triangle in b_i. See e.g. the vertices $\{u, v, w\}$ in Figure 7(a).

If P' was on the outerface of *pertinent*$'(b_i)$, but not placed next to (u, v), then either $P' = P_{i,2}$ and $P_{i,1}$ was placed next to (u, v). See $\{u, v, w\}$ in Figure 7(b). Or there were two other indices j_1 and j_2 with $|P_{j_l,1}| = 2$. See $\{u, v, x\}$ in Figure 7(b). Either way, placing P' next to (u, v) means that some other path of length 2 will be taken away from (u, v), so we get another separating triangle in exchange. □

Lemma 5.4 *Assume we use a vertex cover heuristic that is within a factor c of optimality. Then the number of separating triangles produced when handling an R-node b is within a factor c of optimality.*

Proof: Assume we produced a separating triangle $\{u_i, v_i, w_i\}$. There are two cases: The first case is that not all of $\{u_i, v_i, w_i\}$ are in *skeleton*(b). Since the separating triangle did not exist in any subgraph, it must be that two of the vertices, say u_i and v_i, belong to *skeleton*(b). In Figure 8 this happens with $u_i = x_3, v_i = x_9$ and $w_i = y$. So (u_i, v_i) is a virtual edge in *skeleton*(b) that belongs to some P-node b_i. Furthermore, w_i must be on the outerface of *pertinent*$'(b_i)$ (otherwise the triangle was separating in b_i). Also, there must be at least one more vertex in *pertinent*(b_i) (otherwise the triangle is not separating afterwards). So when handling b the path $P' = \{u_i, w_i, v_i\}$ was not placed next to the edge (u_i, v_i). Since (u_i, v_i) is on the outerface there is only one possible explanation why this happened. Namely, P' belonged to some subgraph b_j of b_i and both $P_{j,1}$ and $P_{j,2}$ had three vertices. In this case one of those paths has to form a separating triangle with (u, v) in *pertinent*$'(b)$, so a separating triangle could not be avoided.

The second case is that all of $\{u_i, v_i, w_i\}$ belong to *skeleton*(b), where they form a triangle. This triangle is either separating in *skeleton*(b) already (in which case it cannot be avoided since the embedding is unique), or it forms a face F of *skeleton*(b). Since the triangle is separating afterwards, at least one of the edges of it must be virtual. Assume it is (u, v) and it belongs to the child-node b_i. Now if (u, v) is not on the outerface of *pertinent*$'(b_i)$ then by Lemma 5.1 the separating triangle $\{u_i, v_i, w_i\}$ can only be avoided by increasing the number of separating triangles in *pertinent*(b_i). So the overall number of separating triangles cannot be reduced (see e.g. the vertices $u_i = x_3, v_i = x_4, w_i = x_9$).

So we may assume that for all of (u_i, v_i), (v_i, w_i) and (u_i, w_i) the edge is either non-virtual or it is on the outerface of *pertinent*$'(\ldots)$ of the corresponding child-node of b. In this case we have $v_F \in H$. In fact, we must have $v_F \in S$ since otherwise the separating triangle would have been avoided. So out of this last case we get at most $|S|$ separating triangles and at least as many separating triangles as in a minimal vertex cover of H. So the factor between the total number of produced separating triangles and the minimal number of separating triangles is at least as good as the size of S in relation to the minimal vertex cover. $\square$

Theorem 5.5 *There is a linear time and space algorithm for computing an embedding of G which has at most twice the minimal number of separating triangles.*

Proof: For the computation of the vertex cover we use a standard linear time algorithm, achieving a vertex cover of at most twice the optimal size. We compute an arbitrary maximal (i.e., non-extendable) matching M in H and for every edge $e \in M$ we add both endpoints to S. This is a vertex cover since for any maximal matching M all edges have at least one endpoint that belongs to an edge in M.

Since in every vertex cover at least one endpoint of every edge in M must be part of the cover it follows directly that S is at most twice times optimal. With the above lemmas it therefore follows directly that the computed embedding has at most twice the minimal number of separating triangles.

Next we show the linear time complexity. Di Battista and Tamassia [3] showed that the SPQR-tree T can be computed in linear time and that the sum of the vertices and edges of *skeleton*(b)'s over all $b \in T$ is $\mathcal{O}(n)$. Let b be an R-node, and let $G' = (V', E')$ be the corresponding graph. Computing the dual graph H of G' and its vertex cover requires $\mathcal{O}(|V'|)$ time which might be too big. But in reality we need not replace the suitable virtual edges e_i by b_i to be able to calculate H. We only need to know whether the length of $P_{i,1}$ is 1 or not. So if we store this information with e_i we can calculate H and S in $\mathcal{O}(|\mathit{skeleton}(b)|)$ time.

For any node b we need to know only whether the length of $P_{i,1}$ is 1, 2, or more for each child-node b_i to calculate *pertinent*$'(b)$ in $\mathcal{O}(|\mathit{skeleton}(b)|)$ time. After this is done we can check in constant time whether the shorther path on the outerface of *pertinent*$'(b)$ has length 1,2, or more. So handling a node b can be done in $\mathcal{O}(|\mathit{skeleton}(b)|)$ time which yields a total running time of $\mathcal{O}(n)$. □

Theorem 5.6 *Let G be planar and biconnected. Then we can test in linear time whether G can be made 4-connected while maintaining planarity.*

Proof: We first test in linear time whether G can be embedded without any separating triangles by applying the above algorithm. We claim that if that is the case the resulting embedding does not contain a star, unless unavoidable.

Assume we did get a star with central vertex w and the vertices $x_1, \dots, x_r$ on the face F. Let G' be the graph induced by $w, x_1, \dots, x_r$ and note that there is only one embedding of G' without separating triangles. In this embedding all faces but F are a triangle. Since F is also a face in G and since G was embedded without separating triangles we must have $G = G'$. But then G contains in every embedding either a star or a separating triangle. So the star was unavoidable.

So after G was embedded without separating triangles we consider a vertex of maximum degree. If its degree is $n - 1$ then G contains a star and cannot be made 4-connected. Otherwise G contains neither a star nor a separating triangle, and we can make it 4-connected with the algorithm of Section 3. □

6 Further Remarks and Open Questions

In this paper we considered the problem of triangulating a planar graph without introducing separating triangles. We showed that if the embedded planar graph has no separating triangles and does not contain a star the resulting triangulation does not contain a separating triangle, i.e. the graph is 4-connected. We also show how to check in linear time whether a non-embedded graph can be made 4-connected while maintaining planarity. These results have important applications in the area of visibility representations of planar graphs.

In order to apply visibility representations, rectangular duals, and straight-line drawings, the graph has to be 4-connected and triangular [14, 17, 9]. In the general augmentation context the question arises how to find a mimimum set of edges whose addition makes a planar graph 4-connected while retaining planarity. To our knowledge this problem is open.

On the other hand, we are able to test 4-connectivity of a planar graph in linear time, using the recent work of Eppstein [4]: He solves the subgraph isomorphism problem in planar graphs in linear time for any pattern of constant size. If there are k occurrences he lists them in time $\mathcal{O}(n+k)$. We use this result as follows: let G be planar and triconnected, hence the embedding of G is unique. Construct a new graph H as follows: Every vertex (resp. face) in G is represented by a vertex-node (resp. face-node) in H. We add an edge between a vertex-node and a face-node iff the corresponding vertex and face in G are incident. Note that H is planar, biconnected, and bipartite.

Assume we have a vertex $v \in G$, and $(v,u),(v,w)$ is a pair of edges consecutive in the adjacency list of v. Then this pair gives rise to a 6-cycle in H as follows: Let (v,u) belong to the faces F_1, F_2, and let (v,w) belong to the faces F_2, F_3. Since $deg(v) \geq 3$ by triconnectivity we have $F_1 \neq F_3$ and hence get the 6-cycle $C = v - v_{F_1} - u - v_{F_2} - w - v_{F_3} - v$ in H. Note that at v we get $deg(v)$ many such cycles. We call these 6-cycles the *edge-cycles.* Also every face F of length 3 gives rise to one additional 6-cycle in H, as shown in Figure 9.

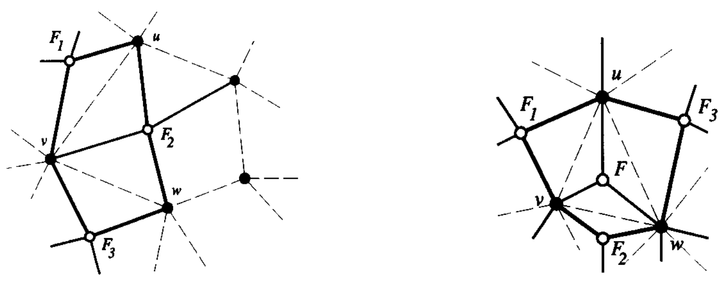

Figure 9: An edge cycle and a face cycle in H, respectively.

A 4-connected planar graph has no other 6-cycle in H. To be precise, we have the following lemma:

Lemma 6.1 *Let G be a triconnected planar graph. with a minimum degree of at least 3. Let k be the number of faces of length 3 in G. Then G is 4-connected if and only if there are exactly $2m+k$ 6-cycles in H.*

Proof: First we show the "only if" part. Assume G is 4-connected and let $C = v - v_{F_1} - u - v_{F_2} - w - v_{F_3} - v$ be a 6-cycle. The vertices of G must be either all inside or all outside this cycle, for otherwise the triplet $\{v, u, w\}$ would be a separation set. Assume w.l.o.g. that they are all outside the cycle. So on the inside of C we have only edges in G. However, since C uses three different faces there must be at least two edges on the inside of C, and those can only be edges of the set $\{(u, v), (v, w), (w, u)\}$. If all three of these edges are in G then C corresponds to a face-cyle, otherwise it corresponds to an edge-cycle. So G contains no other 6-cycle than the edge- and the face-cycles. By the above there are $2m$ edge-cycles and k face-cycles in H.

Conversely, let G be not 4-connected and let $\{u, v, w\}$ be a separation triplet. By planarity there must be three faces F_1, F_2, F_3 with u, v in F_1, v, w in F_2 and w, u in F_3, and we have the 6-cycle $C = v - v_{F_1} - u - v_{F_2} - w - v_{F_3} - v$. Since $\{u, v, w\}$ is a separation pair we have two vertices in two different components of $G - \{u, v, w\}$. But then one of them must be on the inside of C and the other one on the outside. None of the edge- and face-cycles contains vertices on both the inside and the outside. So there must be more than $2m + k$ 6-cycles in H.
□

Using the algorithm of Eppstein, this lemma yields the following result.

Theorem 6.2 *Testing 4-connectivity of a planar graph can be done in linear time and space.*

Triangulating planar graphs without introducing separating quadrangles is a hard problem. Even triangulating a graph consisting of a cycle of length 5 introduces a separating quadrangle. The question arises as to whether there is a characterization of planar graphs that can be augmented to a 5-connected triangular planar graph. Also the time complexity of this augmentation algorithm is an interesting problem for further research. Another natural question is to apply Eppstein's result to test 5-connectivity of general planar graphs in linear time.

However, the authors are not aware of any practical applications of this 5-connectivity triangulation problem. This problem would thus appear to be more of theoretical interest.

References

[1] Bhasker, J., and S. Sahni, A linear algorithm to check for the existence of a rectangular dual of a planar triangulated graph, *Networks* 7 (1987), pp. 307–317.

[2] Chiba, N., and T. Nishizeki, Arboricity and subgraph listing algorithms, *SIAM J. Comput.* 14 (1985), pp. 210–223.

[3] Di Battista, G., and R. Tamassia, Incremental planarity testing, in: *Proc. 30th Annual IEEE Symp. on Foundations of Comp. Science*, 1989, pp. 436–441.

[4] Eppstein, D., Subgraph Isomorphism in Planar Graphs and Related Problems, *Proc. 6th Annual ACM-SIAM Symp. on Discrete Algorithms* 1995, pp. 632–640.

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

[6] Frank, H. and W. Chou, Connectivity considerations in the design of survivable networks, *IEEE Trans. on Circuit Theory*, CT-17 (1970), pp. 486–490.

[7] Garey, M.R., D.S. Johnson, The rectilinear Steiner tree problem is NP-complete, *SIAM J. Appl. Math.* **32** (1977), pp. 826-834.

[8] Garey, M.R., D.S. Johnson and L. Stockmeyer, Some simplified NP-complete graph problems, *Theoret. Comp. Science* 1 (1976), pp. 237–267.

[9] He, X., *Grid Embedding of Internally Triangulated Plane Graphs Without Non-empty Triangles*, Tech. Report TR-95-06, Dept. of Computer Science, SUNY at Buffalo, 1995.

[10] Hsu, T.-S., On four-connecting a triconnected graph, in: *Proc. 33st Annual IEEE Symp. on Found. of Comp. Science*, 1992, pp. 70–79.

[11] Hsu, T.-S., and V. Ramachandran, A linear time algorithm for triconnectivity augmentation, *Proc. 32th Annual IEEE Symp. on Found. of Comp. Sc.*, 1991, pp. 548–559.

[12] Hsu, T.-S., 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.

[13] Kanevsky, A., R. Tamassia, G. Di Battista and J. Chen, On-line maintenance of the four-connected components of a graph, In: *Proc. 32th Annual IEEE Symp. on Foundations of Comp. Science*, 1991, pp. 793–801.

[14] Kant, G., A more compact visibility representation, in: *Proc. 19th Intern. Workshop on Graph-Theoretic Concepts in Comp. Sc. (WG'93)*, Lecture Notes in Comp. Science 790, Springer-Verlag 1994, pp. 411–424.

[15] Kant, G., *Algorithms for Drawing Planar Graphs*, PhD thesis, Utrecht University, Dept. of Computer Science, 1993.

[16] Kant, G., The planar triconnectivity augmentation problem, submitted to *Theoretical Computer Science*, 1993.

[17] Kant, G., and X. He, Two algorithms for finding rectangular duals of planar graphs, in: *Proc. 19th Intern. Workshop on Graph-Theoretic Concepts in Comp. Sc. (WG'93)*, Lecture Notes in Comp. Science 790, Springer-Verlag 1994, pp. 396–410.

[18] Koźmiński, K., and E. Kinnen, Rectangular dual of planar graphs, *Networks 5* (1985), pp. 145–157.

[19] Lengauer, Th., *Combinatorial Algorithms for Integrated Circuit Layout*, Wiley, 1990.

[20] 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.

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

[22] Steiglitz, K., P. Weiner and D.J. Kleitman, The design of minimum-cost survivable networks, *IEEE Trans. on Circuit Theory*, CT-16 (1969), pp. 455-560.