Planar Graph Augmentation Problems

Goos Kant en Hans L. Bodlaender

RUU-CS-91-25
July 1991

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

Planar Graph Augmentation Problems

Goos Kant en Hans L. Bodlaender

Technical Report RUU-CS-91-25
July 1991

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

Planar Graph Augmentation Problems*

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

Abstract

In this paper we investigate the problem of adding a minimum number of edges to a planar graph in such a way that the resulting graph is biconnected and still planar. It is shown that this problem is NP-complete. We present two approximation algorithms for this planar biconnectivity augmentation problem, running in $O(n \log n)$ time and $O(n^2 \log n)$ time, and that have performance ratio 2 and 3/2, respectively. An $O(n)$ approximation algorithm with performance ratio 5/4 is presented to make a biconnected planar graph triconnected by adding edges without losing planarity.

1 Introduction

Many problems concerning the planarity of graphs arise from the wish to draw the graph in an elegant way. Many planar graph drawing algorithms are known, but several of them work only on special classes of planar graphs. For example, the graph drawing algorithm of Tutte [19] to draw a graph with convex faces requires as input a triconnected graph, and the graph drawing algorithm of Woods [20] requires as input a biconnected graph. If the graph is not biconnected, several additional (dummy) edges are added, e.g., using an algorithm of Read [16]. However, there exist instances for which Read's algorithm adds $O(n)$ edges, even when only one edge would be sufficient to make the graph biconnected. (In this paper, n denotes the number of vertices and m denotes the number of edges.) When looking for elegant drawings of graphs, it is useful to preserve the structure of the original drawing as much as possible. For this reason we are looking for the minimum number of edges, that when added to the input graph, yields a biconnected or triconnected planar graph. To be precise, we are looking for the following optimization problem and its decision variant:

*This work was supported by the ESPRIT II Basic Research Actions Program of the EC under contract No. 3075 (project ALCOM).

ISSN: 0924–3275

[PLANAR BICONNECTIVITY AUGMENTATION] (PBA)
Instance: Connected, planar graph $G = (V, E)$.
Question: Find a planar, biconnected graph $H = (V, F)$ with $E \subseteq F$, and $|F - E|$ as small as possible.

Eswaran & Tarjan [2] studied this augmentation problem without the requirement of planarity. They proved that this problem is NP-complete for weighted graphs and linear solvable for unweighted graphs (see also Rosenthal & Goldner [15]). In [3], Frederickson & Ja'Ja give approximation algorithms for the augmentation problem on weighted graphs, working within two times optimal in $O(n^2)$ time. This result is improved by Khuller & Thurimella [11], who obtained the same performance by a simpler algorithm, running in $O(m + n \log n)$ time. [2] and [3] include augmentation algorithms to meet edge-connectivity constraints as well, which formed the base for further research. Recently, Naor et al. [13] describe an $O(\delta^2 nm + nF(n))$ algorithm to find the smallest set of edges to increase the edge-connectivity of G by δ, where $F(n)$ is the time to perform one maximum flow on G. Naor's algorithm finds an optimal sequence of δ augmentations, where in each augmentation-step the edge-connectivity is increased by one, by adding a minimum number of edges. Also recently, Hsu & Ramachandran [8] described a linear algorithm to augment a graph by a minimum number of edges to admit triconnectivity, without the requirement of planarity. However, these algorithms cannot easily be modificated to meet planarity requirements, but some of their techniques will be useful in our approach.

In this paper we prove that the Planar Biconnectivity Augmentation problem is NP-complete, even for the unweighted case. Therefore, we present two approximation algorithms, a fast and relatively easy one, running in $O(n \log n)$ time and has performance ratio 2, and a more complicated and slower one, running in $O(n^2 \log n)$, which has approximation ratio 1.5. Hereto we make use of matching techniques and incremental planarity testing (see [1]), which finds a nice application in this context. The same algorithms can be used to augment a planar graph such that the graph is bridge-connected and planar. In some special cases there exist optimal, polynomial time algorithms. Especially, when the inputgraph is outerplanar, linear algorithms can be obtained such that the augmented graph is bridge-connected, biconnected or triconnected and still planar, by adding a minimum number of edges [9].

We also consider the Planar Triconnectivity Augmentation problem. This is the problem to find the minimum number of edges, which when added to a planar graph yields a triconnected planar graph. It is unknown yet whether the Planar Triconnectivity Augmentation problem is NP-complete. In this paper we describe an $O(n^3)$ approximation algorithm for biconnected graphs, which gives solutions within 5/4 times optimal.

This paper is organized as follows. In section 2 some definitions and preliminary results are given and we prove that deciding whether a planar graph can be made biconnected by adding $\leq K$ edges is NP-complete. In section 3 we present an

$O(n \log n)$ algorithm to make a planar graph biconnected, which works within 2 times optimal. In section 4 we present an $O(n^2 \log n)$ algorithm to make a planar graph biconnected, which works within 3/2 times optimal. In section 5 we describe the 1-2-matching problem, and present an approximation algorithm, which we will use in section 6 to augment a biconnected planar graph such that the resulted graph is triconnected and planar, which works 5/4 from optimal. In section 7 some concluding remarks are given.

2 Preliminaries

Let $G = (V, E)$ be an undirected graph. Assume G has at least two biconnected components, also called *blocks*. A block of G is called *pendant* if it contains exactly one cutvertex. Let p be the number of pendant blocks of G. For each $v \in V$, let $d(v)$ denote the number of connected components of $G - \{v\}$, the graph obtained by removing v from G. Each of these components is called a v-block. Let $d = \max\{d(v) | v \in V\}$, and let $p(v)$ denote the number of pendants connected at v. Let q be the number of blocks in G, not connected with the remaining part of the graph.

Now consider the following operation: take two vertices v_1, v_2 (not cutvertices) in two pendant blocks V_1, V_2 which lie in different v-blocks B_1, B_2 and connect them. This may destroy planarity. However, the new formed graph G' has one v-block less, and the number of pendant blocks is decreased by two, unless the new pendant containing v_1 and v_2 is an entire v-block, in which case the number of pendant blocks is decreased by one. This observation forms the basis for the following theorem.

Theorem 2.1 ([2]) *$max\{d - 1, q + \lceil \frac{p}{2} \rceil\}$ edges are necessary and sufficient to make G biconnected.*

The proof of theorem 2.1 leads to an algorithm for finding a minimum augmenting set of edges to biconnect a graph, which runs $O(m + n)$ time [15] (without the requirement of planarity).

When we require that the augmented graph is also planar, then it is not always possible to connect pairs of pendant blocks. For example, in Figure 2, edges e_1, e_2, e_3 and e_4 each form a pendant block. e_1 and e_2 can be connected without losing planarity, but e_3 and e_4 cannot. Essentially, making a graph biconnected with the minimum number of edges corresponds to connecting as much as possible pairs of pendant blocks with each other without destroying planarity. The number of added edges equals the number of edges, that connect pairs of pendant blocks (hereafter called matching edges), plus the number of pendant blocks, not connected by a matching edge. Minimizing the number of extra edges is hard, as stated in the following theorem:

Theorem 2.2 *The following problem is NP-complete:*

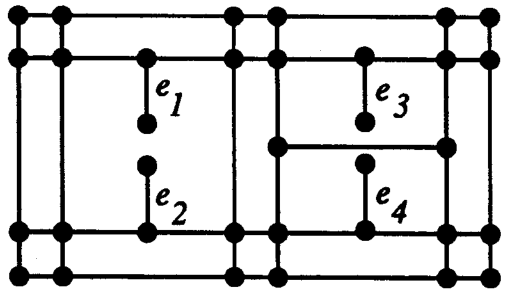


Figure 1: e_1 and e_2 can be matched with each other, but e_3 and e_4 can not.

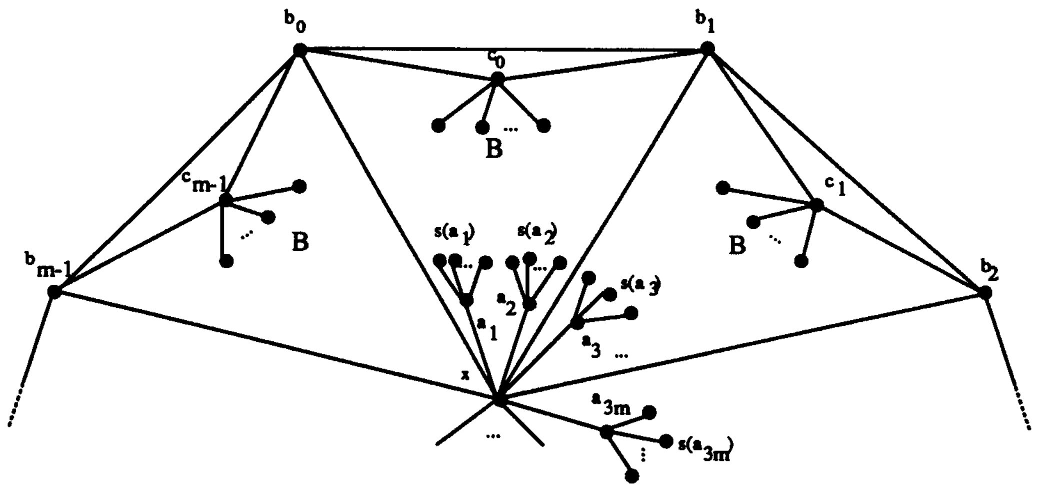


Figure 2: Construction of the graph for the NP-completeness proof.

[PLANAR BICONNECTIVITY AUGMENTATION] (PBA)
Instance: Connected, planar graph $G = (V, E)$, and an integer K.
Question: Find a planar, biconnected graph $H = (V, F)$ with $E \subseteq F$, and $|F - E| \leq K$.

Proof: Clearly the problem is in NP: guess $L \leq K$ edges to be added to G, and check in polynomial time whether the resulting graph is biconnected and planar.

To prove the NP-hardness, we show that 3-PARTITION (which is well-known to be NP-complete in the strong sense) is reducible to the planar biconnectivity augmentation problem. Let an instance of 3-PARTITION be given, i.e., a set A of $3m$ elements $a_1, \ldots, a_{3m}$, a bound $B \in \mathbb{Z}^+$ and a size $s(a_i) \in \mathbb{Z}^+$ for each $a_i \in A$ such that $B/4 < s(a_i) < B/2$ and $\sum_{a_i \in A} s(a_i) = mB$. We construct a planar graph as follows:

Introduce a vertex s and for each $i, 0 \leq i < m$, introduce vertices b_i, c_i, and the edges $(b_i, b_{i+1}), (b_i, x)$, (b_i, c_i) and (b_{i+1}, c_i) (additions modulo m). Introduce for each vertex c_i B additional edges to B new vertices. Introduce $3m$ new vertices $a_1, \ldots, a_{3m}$, connected at x and each vertex a_i has $s(a_i)$ additional edges to $s(a_i)$ new vertices (see Figure 2). Let G be the resulting graph.

Clearly G has $2mB$ pendant blocks, so at least mB edges are needed to make G biconnected. G can be made biconnected without destroying planarity with exactly mB edges, if and only if it is possible to have a matching edge from each pendant of cutvertex a_i to an unique pendant of a cutvertex c_j. This can be done, if and only if for each c_j, the B pendants are matched with pendants of some vertices $a_{i_1}, a_{i_2}, a_{i_3}$, i.e., if and only if there exists a partition of A into m disjoint sets $A_1, \ldots, A_m$ such that for all $i, 1 \leq i \leq m, \sum_{a \in A_i} s(a) = B$. As G can be constructed in time, polynomial in m and B, there is a polynomial time transformation from the strongly NP-complete 3-partition problem to the planar biconnectivity augmentation problem, hence the latter is NP-complete. □

In a special case, however, the problem becomes efficiently solvable:

Theorem 2.3 *If all cutvertices are part of one triconnected component, then we can find in $O(n^{2.5})$ time the minimum number of edges, which added to the graph gives a biconnected planar graph.*

Proof: Let G' be the triconnected component that contains all cutvertices of the inputgraph G. We construct a new graph H by representing every vertex $v \in G$ by $p(v)$ vertices in H. We add an edge (v_1, v_2) in H iff v_1 and v_2 share a common face in the unique embedding of G', and v_1 and v_2 do not represent the same vertex in G. The planar biconnectivity augmentation problem is now equal to finding a maximum cardinality matching M in H. Let $P = \sum_{v \in G} p(v)$, then it follows that the number of required edges is equal to $P - |M|$. Crossings between added edges in one face can be removed easily. Constructing a maximum cardinality matching can be done in $O(\sqrt{|V_H|}.|E_H|)$ time [12], but since $|E_H|$ can be $O(|V_G|^2)$, the total running time of the algorithm is $O(|V_G|^{2.5}) = O(n^{2.5})$. □

3 Approximation within 2 times optimal

Since the planar biconnectivity augmentation problem is NP-complete, we restrict our attention to approximation algorithms. In this section we present an $O(n \log n)$ algorithm, whose number of added edges is within 2 times optimal. Therefore, we first recognize the cutvertices of the graph by constructing the *block-cutpoint graph* $bc(G)$ of G as follows (cf. Harary [7]): every block and cutvertex of G is represented by a vertex of $bc(G)$ and two vertices v_1, v_2 of $bc(G)$ are adjacent if and only if the corresponding cutvertex of v_1 in G is contained in the corresponding block of v_2

in G or vice versa. $bc(G)$ can be constructed in linear time, and it can easily be shown that $bc(G)$ is always a forest; it will be known as the bc-tree of G when G is connected. Let p and q be the number of leaves and isolated vertices of $bc(G)$, respectively. If G is not connected, then we apply the following algorithm (see also [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 $1 \leq i \leq t$.
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]. Hence we may assume further that G is connected and let T be its bc-tree. Let one arbitrary blockvertex b be the root of T and let for each block B_i in G, b_i denote the corresponding blockvertex in T. Assume we have pointers from the children to their parent in T and assume that the children of every vertex are stored in a doubly linked list. Finally, we want to test whether adding an edge (v_1, v_2) to G, denoted by PLANAR(v_1, v_2) preserves the planarity. Hereto we construct the PQRS-datastructure of Di Battista & Tamassia [1] to store the planar graph G such that we can test in $O(n \log n)$ time amortized whether adding an edge (v_1, v_2) to G preserves the planarity. The PQRS-datastructure can be built in $O(n \log n)$ time. Let an *outside* vertex of a block be an arbitrary vertex on the outerface of that block. When an edge is added between two blocks, these blocks and the blocks in between have to be coalesced into one block. For this we introduce an union-find structure (see Tarjan [17]), which means that finding father(v_c) for a cutvertex v_c implies a find-call on the underlying union-find structure in the following algorithm:

```
PBA_2OPT
  while $T$ is not a single vertex $b$ do
    let $c$ be a cutvertex with only leaves as sons in $T$;
    if $c$ has $\geq 1$ sons $b_1, b_2, \ldots, b_k$ in $T$ then
      add edges between outside vertices of $B_i$ and $B_{i+1}$, $1 \leq i < k$;
    $v$ := outside vertex of $B_1$; $c_2 := c_1 := c$;
    while PLANAR$(v, c_1)$ and father$(c_1) \neq b$ do
      $c_2 := c_1$;
      $c_1$ :=father(father$(c_2)$); {take next cutvertex towards $b$}
    od
    if PLANAR$(v, c_1)$ then $c_2 := c_1$;
    $b_x$ := father$(c_2)$;
    add an edge between $v$ and an outside vertex of $B_x$, neighbour of $c_2$;
    for all cutvertices $w$ on path between $v$ and $b_x$ do
      if $deg_T(w) > 2$ then father$(w) := b_x$ and update degrees
    for all blockvertices $b_y$ between $b_1$ and $b_x$ do union($b_x, b_y$);
od
```

Theorem 3.1 *This algorithm can be implemented to run in $O(n \log n)$ time*

Proof: We first construct the block-cutpoint graph and connect it to a tree, requiring $O(n)$ time. Building the PQRS-datastructure requires $O(\log n)$ time amortized per edge, so total $O(n \log n)$ time, since $m = O(n)$ for G. In every step we take a cutvertex c in T. We first count the total number of iterations in these two while-loops and the for-loop. For this, notice that every time that PLANAR(v, c_1) is true, then the number of blocks is decreased by one, hence this will occur at most n times. Since also PLANAR(v, c_1) is false at most n times, this leads to at most $2n$ iterations of the while- and for-loops.

The PLANAR(v, c_1) test requires $O(\log n)$ time amortized each [1]. Finding the father of a cutvertex requires $O(\alpha(m,n)) = O(\log n)$ time amortized each [17], with $\alpha(m,n)$ the functional inverse of Ackermann's function, by using the union-find structure, including path compression and union by rank. Unioning two blocks into one also requires $O(\alpha(m,n))$ time [17]. All other statements in the algorithm can be implied in $O(1)$ time, which implies an $O(n \log n)$ time algorithm. □

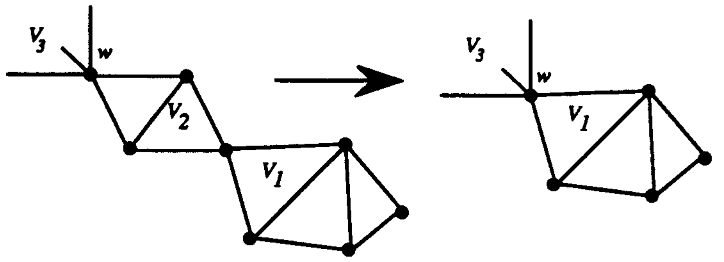
Theorem 3.2 *This algorithm works within 2 times optimal.*

Proof: By theorem 2.1, the number of required edges is at least $\lceil \frac{p}{2} \rceil$. In PBA_2OPT, every pendant gets one additional edge. Furthermore, one non-pendant block B_i gets an additional edge, if and only if PLANAR(v, father(b_i)) is false for all cutvertices v, descendants of b_i in T. This means that no descendant pendant of B_i can be matched outside B_i. But now also in the optimal solution B_i must get an additional edge to preserve the biconnectivity and planarity. Say this occurs r times, then now $p + r$ edges are added, but it easily follows that also in the optimal solution $\lceil \frac{p}{2} \rceil + \lceil \frac{r}{2} \rceil$ edges are required, to preserve the planarity. □

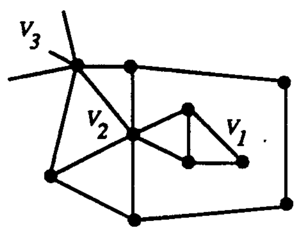
4 Approximation within 3/2 times optimal

In section 3 a relative easy algorithm is described, working within two times optimal. Our approach is now to lower the constant 2 to a significantly lower constant. Therefore, in this section we present a more complicated algorithm, for which we prove that it works within 1.5 times optimal, which is tight as well.

When G has a pendant V_1, that is connected to exactly one other block V_2, and V_2 is connected to exactly one other block V_3, then we call $V_1, V_2, \cdots$ a *chain of blocks*. The first step in PBA_1.5OPT is to reduce chains of blocks. When we add an edge from V_1 to a block outside the chain, then all blocks $V_2, V_3, \cdots$ in the



(a) We can delete block V_2 for simplicity.



(b) V_1 lies inside V_2

Figure 3: Reduce Chains.

chain are in the same block as V_1. Hence we can discard the blocks $V_2, V_3, \cdots$, for simplicity (see Figure 3(a)). On the other hand, if V_1 lies *inside* V_2 then V_1 can only be connected with V_2 and we can union them into one pendant immediately (see Figure 3(b)). Inside means that no other cutvertex of V_2 shares a common face with the cutvertex of V_1. We can check this by testing whether the graph plus the edge between the two cutvertices is not planar, by using the function PLANAR of section 3. These observations can be described formally by the following algorithm:

```
REDUCE_CHAINS
    while not all pendants visited do
        let $V_1$ be an unvisited pendant; $i := 1$;
        let $v_1$ be the cutvertex between $V_1$ and $V_{i+1}$;
        while $d(v_i) = 2$ and $V_{i+1}$ contains at most two cutvertices $v_1, v_{i+1}$ do
            if PLANAR($v_1, v_{i+1}$) do
                delete $V_{i+1}$ and connect $V_1$ at $v_{i+1}$;
            else
                merge $V_1$ and $V_{i+1}$ into one pendant $V_{i+1}$ by adding an edge between
                two outside vertices of $V_1$ and $V_{i+1}$;
                $V_1 := V_{i+1}; v_1 := v_{i+1}$
            $i := i + 1$
        od
    od
```

When two pendants V_1 and V_2 in a face F are matched by an edge, then all pendants left from this edge in F cannot be matched with pendants right from this edge in F without destroying the planarity (see e.g. Figure 4). However, when such a crossing between added edges $(v_1, v_2), (v_3, v_4)$ occurs in a face, then we can always remove it: change (v_1, v_2), (v_3, v_4) into (v_1, v_3), (v_2, v_4) or into (v_1, v_4), (v_2, v_3). So, we allow this type of crossings. To model the situation and to facilitate planarity tests, we introduce *face vertices*.

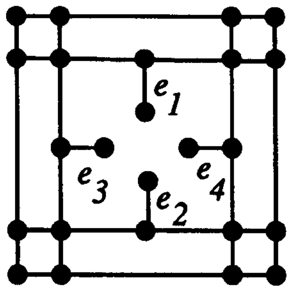


Figure 4: If we match e_1 and e_2 with each other, then e_3 and e_4 can not be matched. If we match e_1 with e_3, then e_2 can be matched with e_4.

Instead of adding an edge between two outside vertices v_1 and v_2 of pendants V_1 and V_2, respectively, we add an edge from v_1 and v_2 to the face vertex of F. This means that two pendants V_1 and V_2 can be matched with each other if PLANAR(v_1, F) and PLANAR(F, v_2) for a certain face vertex F holds, or if PLANAR(v_1, v_2) holds. Initially there are no face vertices. We now give the main step of PBA_1.5OPT. Recall that $p(v)$ denotes the number of pendants connected at v.

```
MATCH_PENDANTS
    **while** there is a matching between pendants possible **do**
        choose $v_1$, $v_2$, such that PLANAR$(v_1, v_2)$ or there exists a face $F$ with
        (PLANAR$(v_1, F)$ and PLANAR$(F, v_2)$) with $p(v_1)$
        maximal, and for this choice of $v_1$, $p(v_2)$ maximal;
        **if** there is no $F$ with (PLANAR$(v_1, F)$ and PLANAR$(F, v_2)$) **then**
            introduce a new face vertex $F$;
        add an edge between a vertex from a $v_1$-pendant and from a vertex
        from a $v_2$-pendant to $F$;
        decrease $p(v_1)$ and $p(v_2)$ by 1
    **od**
```

If there are several cutvertices v_2 with the required property in MATCH_PENDANTS, then we take this vertex v_2, for which the coalesced pendant is as large as possible, hence for which v_1 and v_2 have the highest common ancestor in $bc(G)$. Otherwise, suppose we have a planar graph for which the bc-tree forms a binary tree with 2^k leaves (see section 3 for the definition of a bc-tree). Since every pendant can be matched with another pendant and all cutvertices have $p(v) = 1$, we can match in every step two arbitrary pendants with each other. This leads to a worst-case solution of $2^k - 1$ edges, while if we match in every step two pendants with highest common ancestor with each other, we obtain the optimal solution of 2^{k-1} edges (see Figure 5).

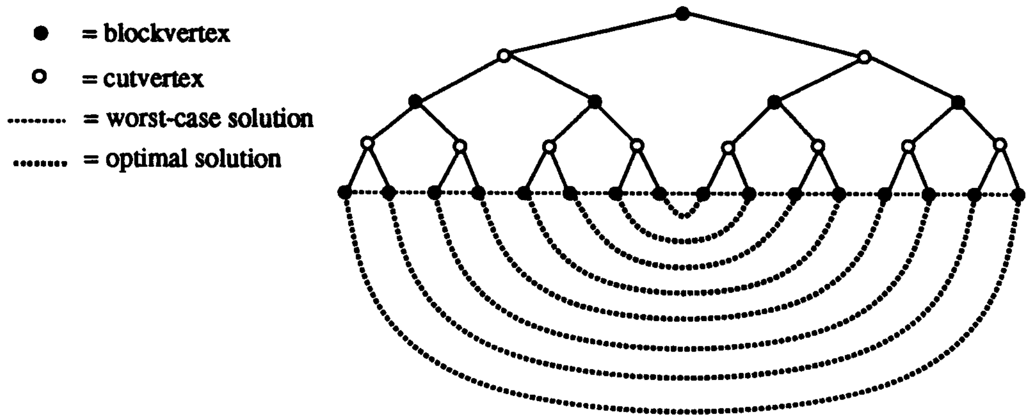


Figure 5: Optimal and worst-case solution when matching cutvertices, not with highest common ancestor.

Note that by matching two pendants in one face, the embedding of G is also restricted in a certain way. This may mean that several other pendants of the two corresponding cutvertices can only be matched with each other. Hence, in general, the $p(v)$ pendants of cutvertex v will be matched with pendants of less than $p(v)$ other cutvertices.

After completing MATCH_PENDANTS, we change the matching edges $(v_1, F), (F, v_2)$ of a matching between the blocks V_1 and V_2 into one edge (v_1, v_2). We delete the face vertices and remove crossings between the additional edges as described above.

At this point in the algorithm, there may still exist some labels $p(v) > 0$ in the graph, which correspond with pendants which cannot be matched anymore with another pendant. For each of these pendants, we have to add an additional edge, that only resolves the biconnectivity of that pendant. In the remainder of this section, we will call this type of edges *extra edges*. Hence, if M is the sum of the unmatched labels then M extra edges are required. We distinguish two types of extra edges:

cheap edges = extra edges, which are also required in the optimal solution.

expensive edges = extra edges, which are not required in the optimal solution.

If $k \leq p(v)$ pendants, connected at cutvertex v, are matched with other pendants in a common face F or can only be embedded in a face F without destroying planarity, then we say that label k *belongs to* face F. To count the number of expensive edges, the following lemma is useful.

Lemma 4.1 *Let the labels* $B_1, \ldots, B_k$ belong to a face F, *with* $\sum_{i=1}^{k} B_i = B$. *If* $max_i\{B_i\} < \lceil \frac{B}{2} \rceil$ *then all pendants can be matched by* $\lfloor \frac{B}{2} \rfloor$ *matching edges, and if* B

is odd one extra edge, otherwise $B-$ $max_i\{B_i\}$ matching edges and $2\cdot max_i\{B_i\} - B$ extra edges are required.

Proof: If $B_1 = \max_i\{B_i\} \geq \lceil\frac{B}{2}\rceil$ then edges go from label B_1 to all other labels and $B_1 - (B - B_1)$ pendants remain unmatched. If $B_1 < \lceil\frac{B}{2}\rceil$ then it is not possible that ≥ 2 pendants remain unmatched. For suppose that $k \geq 2$ pendants in face F remain unmatched, then they must necessarily be connected with the same cutvertex v_1. Inspect the last matching between two pendants in F, connected with cutvertices v_2 and v_3 ($v_2 \neq v_1$ and $v_3 \neq v_1$). But at that moment $p(v_2) = p(v_3) = 1$ and $p(v_1) = k > 1$ holds, hence a matching between a v_1-pendant and a v_2-pendant (or v_3-pendant) was made by the algorithm. Hence all pendants can be matched with each other by $\lceil\frac{B}{2}\rceil$ matching edges. □

Since we are counting the expensive edges in worst-case situation and in the case $\max_i\{B_i\} < \lceil\frac{B}{2}\rceil$ no extra edges are required, we assume in our analysis w.l.o.g. that $\max_i\{B_i\} \geq \lceil\frac{B}{2}\rceil$ holds.

Lemma 4.2 $|$*expensive edges*$| \leq 2|$*matching edges*$|$.

Proof: Every matching edge has to *pay* for two expensive edges and initially no matching edge pays. Consider an optimal solution and our approximate solution of PBA_1.5OPT. Let $p(v) = A$ be a biggest label in MATCH_PENDANTS, matched with another label B (or some labels with total sum B), whereas in the optimal solution A is matched with labels $B_1, \ldots, B_k$. It follows by MATCH_PENDANTS that $\max_i\{B_i\} \leq B$. For suppose that $B_1 = \max_i\{B_i\} > B$, and A is not matched with B_1, then B_1 is matched before A, say with label A'. Then $B_1 > A$ or $A' > A$ holds and, hence, A was not the biggest label matched with another label, which yields a contradiction.

Notice that by the approximate matching $B' \leq B$ pendants remain unmatched, which are matched with B in the optimal solution. This leads to B' expensive edges, to be paid by the B matching edges (between labels B and A). In the optimal solution all A pendants are matched, so assume w.l.o.g. that $\sum_{i=1}^{k} B_i \geq A$. If in MATCH_PENDANTS the B_i-pendants are matched in a common face F, then we change the matching edges in F such that they appear between two B_i-pendants or between two other pendants. Since $\max_i\{B_i\} \leq B$ this means that at least $\sum_{i=1}^{k} B_i - B$ matching edges between B_i-pendants are possible, hence at most $2B - \sum_{i=1}^{k} B_i$ expensive edges. This leads to a total number of $B' + (A - B) + 2B - \sum_{i=1}^{k} B_i \leq 2B$ expensive edges and at least B matching edges. So assume further that labels $B_1, \ldots, B_p$ ($p \leq k$) are matched in other faces and labels $B_{p+1}, \ldots, B_k$ are matched in F.

First suppose that labels $B_1, \ldots, B_p$ are matched with labels $C_1, \ldots, C_p$, with $C_i > B_i$ ($1 \leq i \leq p$). Thus B_i matching edges can be made and $C_i - B_i$ pendants remain unmatched for $1 \leq i \leq p$. Since the B matching edges have to pay for the

B' expensive edges, also the B_i matching edges have to pay for $B_i' \leq B_i$ expensive edges of pendants, which remained unmatched now, but are matched with B in the optimal solution. Hence we assign to each B_i a number of B_i expensive edges and we still have to pay for the following number of expensive edges: $B' + (A - B) + 2 \cdot \max_i\{B_i\} - (B_{p+1} + \ldots + B_k) - (B_1 + \ldots + B_p) \leq A + 2 \cdot \max_i\{B_i\} - \sum_{i=1}^{k} B_i \leq 2B$. These can be assigned to the B matching edges.

We now may further assume that all B_i-pendants ($1 \leq i \leq p$) are matched with C_i-pendants with $C_i \leq B_i$ ($1 \leq i \leq p$). This means that only C_i matching edges between the B_i- and C_i-pendants can occur, yielding $B_i - C_i$ expensive edges. Since C_i is matched with B_i, C_i' ($C_i' \leq C_i$) other pendants remain unmatched, which are matched with label C_i in an optimal solution. Assume further w.l.o.g. that by the approximation algorithm first B_1 is matched with C_1, then B_2 with C_2, ..., then B_p with C_p. Then $B_1 \geq C_1 \geq B_2 \geq \ldots \geq C_{p-1} \geq B_p \geq C_p \geq \max\{B_{p+1}, \ldots, B_k\}$ holds. This leads to

$$\begin{array}{rl} & B + B_{p+1} + \ldots + B_k - \max_{p+1 \leq i \leq k}\{B_i\} + C_1 + \ldots + C_p \\ \geq & \max_{p+1 \leq i \leq k}\{B_i\} + B_1 + \ldots + B_p \end{array}$$

matching edges and

$$\begin{array}{rl} & B' + (A - B) + 2.\max_{p+1 \leq i \leq k}\{B_i\} - (B_{p+1} + \ldots + B_k) \\ & \quad +(B_1 - C_1) + \ldots + (B_p - C_p) + C_1' + \ldots + C_p' \\ \leq & A - \sum_{i=1}^{k} B_i + 2 \cdot (B_1 + \ldots + B_p) + 2 \cdot \max_{p+1 \leq i \leq k}\{B_i\} \\ \leq & 2 \cdot (\max_{p+1 \leq i \leq k}\{B_i\} + B_1 + \ldots + B_p) \end{array}$$

expensive edges.

By this it follows that although the matching of A and B may not lead to an optimal matching, all involved expensive edges can be paid by the involved matching edges, and some expensive edges are assigned at some matching edges, which are treated later. We can further ignore all those treated matching and expensive edges: delete these edges from the graph G, obtaining a reduced graph G'. Apply the same argument to G': find in G' the biggest label A, matched with another label B, whereas in the optimal solution A was matched with the labels $B_1, \ldots, B_k$. Suppose that some $B_i > B$ and B_i is now matched with A'. Then $B_i > A$ or $A' > A$ holds, which means that we have already treated the labels B_i and A', so they are already removed from the graph. Hence $\max_i\{B_i\} \leq B$ holds, and we can apply our argument to G'.

Since each time at least one edge is deleted from the graph, this certainly stops and every matching edge pays for at most two expensive edges, which proves the lemma. □

Our approximate solution consists of three types of edges: the matching edges (M_m), the cheap edges (M_{c1}) and the expensive edges (M_e). The optimal solution consists of matching edges (M_{opt}) and cheap edges (M_{c2}). Note that $M_e \leq 2M_m$ by lemma 4.2 and $M_{c2} \geq M_{c1}$ by definition.

Theorem 4.3 PBA_1.5OPT *gives solutions that use as most 3/2 times the optimal number of edges.*

Proof: Let p be the total number of pendants when starting PBA_1.5OPT. Then $p = 2 \cdot M_{opt} + M_{c2} = 2 \cdot M_m + M_e + M_{c1}$. Hence $M_m \leq M_{opt}$ and $M_{opt} = (2 \cdot M_m + M_e + M_{c1} - M_{c2})/2 \geq M_m + \frac{1}{2}M_e - \frac{1}{2}M_{c2}$. The approximate solution is $M_m + M_{c1} + M_e \leq M_m + \frac{1}{2}M_m + \frac{3}{4}M_e + \frac{3}{2}M_{c2} - \frac{1}{2}M_{c2} \leq \frac{3}{2}(M_{opt} + M_{c2})$. Hence PBA_1.5OPT works within 1.5 times the optimal solution. $\square$

In REDUCE_CHAINS we use the *block-cutpoint graph* $bc(G)$, to find the blocks and cutvertices. In the internal while-loop every time a block is visited, a block is deleted or two blocks are coalesced into one, i.e., the number of blocks decreases by one in each pass. Furthermore, every pendant and block will be visited at most once, hence the test PLANAR(v, c_1) will be executed at most n times, requiring $O(\log n)$ time amortized each. All other statements of the algorithm requires only $O(1)$ time, hence the algorithm REDUCE_CHAINS requires $O(n \log n)$ time.

Next, we construct for each cutvertex v a binary balanced tree $CV(v)$, containing those cutvertices w, for which in the original graph G, PLANAR(v, w) holds. Furthermore we introduce for each cutvertex v another binary balanced tree $FV(v)$. $FV(v)$ contains these face vertices F, for which PLANAR(F, v) holds during the augmentation of G. So initially $FV(v)$ is empty for all cutvertices v of G. With bucketsort we sort the labels $p(v)$ in non-increasing order. In every step we take the biggest label $p(v_1)$ and search for the biggest label $p(v_2)$. Checking whether pendants v_1, v_2 can be matched is done as follows:

1. Check if $v_1 \in CV(v_2)$. If not then PLANAR(v_1, v_2) did not hold initially, so find another pair.
2. Check if there is some F, with $F \in FV(v_1)$ and $F \in FV(v_2)$.
3. If not, then check whether PLANAR(v_1, v_2) holds.

If there is some face vertex F with $F \in FV(v_1)$ and $F \in FV(v_2)$, then this means that there is a face vertex F, such that PLANAR(v_1, F) and PLANAR(F, v_2) holds. If there is no such F, but PLANAR(v_1, v_2) holds, then a new face vertex F must be created. After adding the edges (v_1, F) and (F, v_2), the trees FV are updated as follows: if F is new, then F is added to $FV(w)$ for every cutvertex w with $v_1, v_2 \in CV(w)$. If F is not new, then F is deleted from every $FV(w)$ for every cutvertex w with $F \in FV(w)$ and $v_1 \notin CV(w)$ or $v_2 \notin CV(w)$. Updating the trees FV requires $O(n \log n)$ time, but notice that this will happen at most $O(n)$ times, since at most $O(n)$ edges will be added.

Using the algorithm of [1], it is possible to carry out all PLANAR(v_1, v_2) tests in $O(\log n)$ time per call, amortized. Analysis of the time complexity of our algorithm leads to the following result:

Theorem 4.4 *There exists an approximation algorithm for the Planar Biconnectivity Augmentation Problem with performance ratio 1.5 and time complexity* $O(n^2 \log n)$.

5 1-2-Matching

5.1 The algorithm

In this section we consider a problem on triconnected planar graphs, and give an approximation algorithm for this problem. The algorithm will be used as a 'tool' in section 6. Consider triconnected planar graphs $G' = (V, E)$, in which every edge e has a set of one or more characteristics associated with it, denoted by $C(e)$, where each characteristic is one of the following: $0 \sim 0, 1 \sim 0, 1 \sim 1, 2 \sim 0, 2 \sim 1$ or $2 \sim 2$. If $|C(e)| \geq 2$ then for all $i \sim j \in C(e)$, $i + j$ is even or for all $i \sim j \in C(e)$, $i + j$ is odd. A 1-2-matching is a 'matching', that is obtained as follows:

1. We fix one characteristic $i \sim j \in C(e)$ for each edge e, and we assign i to one of the adjacent faces, and j to the other adjacent face.
2. In a face, it is possible to match a 1 with a 1 or with a 2, a 2 with a 2 or with two 1's, or three 2's with each other. (Every number may be matched at most once.)

In our application, every edge (a, b) represents a connected surbgraph V' of G, connected with a and b with $G - V'$. A characteristic $i \sim j$ of (a, b) represents the number of augmentation edges at both sides of V', which has to go from V' to $G - V'$ to make V' triconnected. If two numbers between two edges are matched, then one or more *matching* edges are added between the subgraphs in G that are represented by the edges in G'. In some cases, some extra edges are needed. These edges are added in the following way: if a 1 is matched with a 1, then one matching edge is taken; if a 2 is matched with a 2, then two matching edges are taken between the two subgraphs; if a 1 is matched with a 2, then one matching edge is taken, and one extra edge, which goes from the component with a 2; if a 2 is matched with two ones, then two matching edges are taken, each between the component with number 2, and to one of the components with number 1; if three 2's are matched with each other, then three matching edge are taken, between each of the three pairs of subgraphs one. Finally, for each number i that is not yet matched, i extra edges are taken, going to the corresponding subgraphs. The problem is to fix the characteristics, and to find an assignment of numbers to faces, and a matching, such that the total number of matching edges plus extra edges is as small as possible. We call this problem the 1-2-matching problem. An interesting, but still open question is whether this problem can be solved in polynomial time. In Figure 6 an inputgraph and a corresponding minimum 1-2-matching are given.

In this section we present an approximation algorithm APPROX_1-2-M, that yields a solution that uses at most 5/4 times the optimal number of edges.

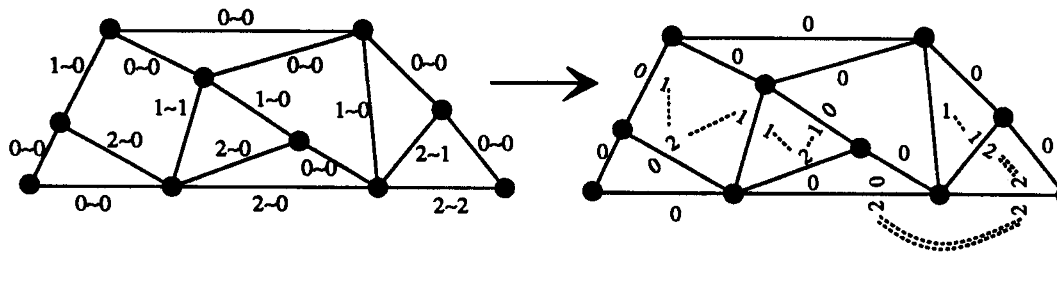


Figure 6: Example of an inputgraph and a 1-2-matching.

Definition 5.1 *An edge e is a 2-edge if there is a characteristic $2 \sim i \in C(e)$, otherwise it is a 1-edge.*

Note that if we can match a 2-edge with two 1-edges, then instead of it we could also match the two 1-edges with each other. Therefore, we first attempt to match a maximum number of 2-edges with each other, and then we try to match the remaining 2-edges with the present 1-edges and a maximum number of 1-edges with each other. This forms the basis for our approximation algorithm APPROX_1-2-M, which consists of constructing two maximum matchings, called the 2-matching and the 1-matching, on two to be constructed matching graphs, H_2 and H_1. H_2 is defined as follows:

If for a 2-edge e, $2 \sim 2 \in C(e)$, then e is represented by two vertices v_1 and v_2 in H_2, otherwise e is represented by one vertex in H_2. If two 2-edges e_i and e_j share a common face in G', then an edge between the two corresponding vertices v_i and v_j in H_2 is added. Notice that a triconnected planar graph has an unique embedding.

Every vertex v_i in H_2 receives an $a(v_i)$-value as follows: if the corresponding 2-edge e_i in G' has $|C(e)| \geq 2$ or if e_i shares a common face with a 1-edge then $a(v_i) = 0$, otherwise $a(v_i) = 1$. We are now looking for a subgraph $H_2' = (V, E')$ of H_2 such that we first maximize the number of vertices v_i with $a(v_i) \leq deg_{H_2'}(v_i) \leq 1$ and then for this number of vertices, also maximize $|E'|$ (this is called the *degree constrained subgraph problem*, e.g. see [16]). In other words, if $a(v_i) = 1$, then v_i must be matched if possible, i.e., if there are no 1-edges to match v_i with and the corresponding 2-edge e_i contains no other characteristics.

For every unmatched vertex v in H_2 we test if there is a path $v = v_1, \ldots, v_k$ (k **odd**) in H_2 such that $(v_{2i}, v_{2i+1}) \in E'$, and v_k has two neighbours w_1, w_2, with $(w_1, w_2) \in E'$. If so, then we change this matching as follows: delete matching edges (v_{2i}, v_{2i+1}) and (w_1, w_2), add the matching edges (v_{2i-1}, v_{2i}), $1 \leq i \leq \lfloor \frac{k}{2} \rfloor$ and match the three vertices w_1, w_2 and v_k with each other. Notice that no two unmatched vertices can have a path to a common pair w_1, w_2, otherwise the matching E' was not maximal. The resulting matching is called a 2-matching.

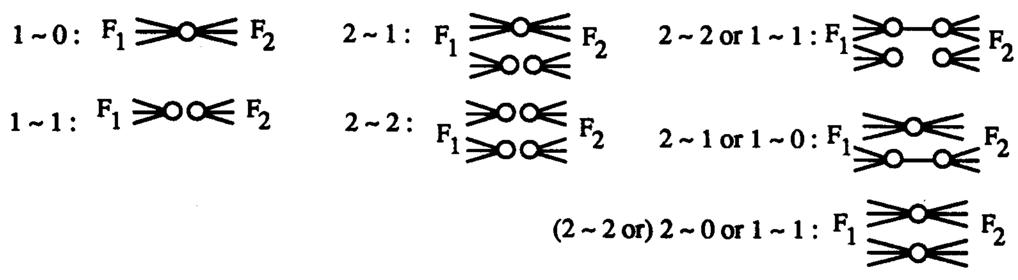


Figure 7: Matching components for the characteristics.

We change in the characteristics the 2 into 0 of those 2-edges in G', whose corresponding vertices are matched in H_2'. Edges in G', that have a characteristic $0 \sim 0$ are ignored, but for the remaining 2- and 1-edges, we construct a second matching graph H_1. The edges e in G' are represented in H_1 according to $C(e)$ by one of the matching components (assume e belongs to faces F_1 and F_2), as shown in Figure 7. This means that if $C(e) = \{1 \sim 1\}$ then e is represented by two vertices v_1 and v_2 in H_1. There is an edge from v_1 (v_2) to those vertices in H_1, whose corresponding edge in G' belongs to face F_1 (F_2). All other characteristics are dealt with in a similar way. If e has except characteristic $2 \sim 2$ also characteristic $2 \sim 0$, then we can ignore characteristic $2 \sim 2$ for simplicity.

The matching components are chosen in such a way that there is a directed correspondence between the 1-2-matching in G' (except when these involve $2 \sim 0$ characteristics) and (ordinary) matchings in H_1. If e has characteristics $2 \sim 2, 1 \sim 1$ and $2 \sim 0$ associated with it, then we ignore the characteristic $2 \sim 2$, because if we can match a $2 \sim 2$ characteristic with two times two 1-edges, then instead of it we can match two 1-edges with each other and match the other two 1-edges with a $2 \sim 0$ characteristic. It easily follows that a $1 \sim 1$ or $2 \sim 0$ characteristic correspond exactly with two $1 \sim 0$ characteristics.

Note that for characteristic $2 \sim 0$ no corresponding matching component in H_1 is constructed. Consider an edge with characteristic $2 \sim 0$, that is adjacent to faces F_1 and F_2. The problem here is that both matching edges must belong to the same face. We are not able to represent this by a matching component. We deal with the situation in the following way. First note that the faces F_1 and F_2 can not contain other matched or unmatched 2-edges, because then the 2-matching would not have been maximal. If $k \geq 2$ 1-edges belong to face F_1 (or F_2), then we give every edge between the corresponding vertices in H_1 weight $1 + \frac{1}{k}$. If there is only one 1-edge in F_1 (or F_2), then we introduce a new vertex in H_1, with an edge of weight $\frac{3}{4}$ to the corresponding vertex. We now apply an algorithm to find a maximum weighted matching in H_1 [6]. The weights $1+\frac{1}{k}$ makes that these edges are preferred in the maximum weighted matching. (The meaning of the values of these weights will become clear in the analysis.) The matching edges in H_1 correspond in

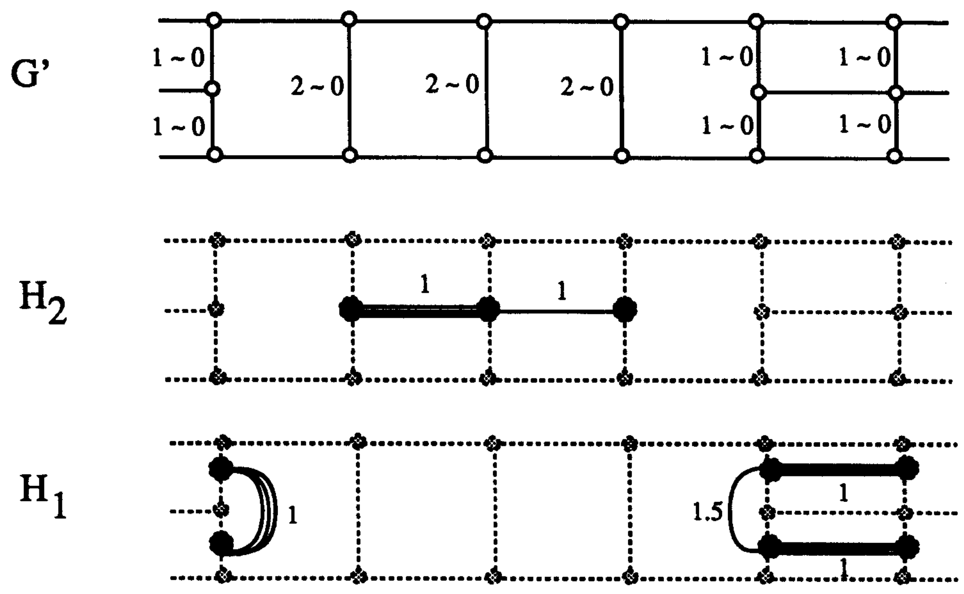


Figure 8: An inputgraph G', and the two matching graph H_2 and H_1. Here 7 edges are required, in the optimal solution 6.

a straightforward way to matchings made between numbers in G'.

The final steps of the algorithm are the following. For each unmatched edge e in G', with $2 \sim 0 \in C(e)$, we remove — if possible — one matched $(1+\frac{1}{k})$-edge in one of the two shared faces, and connect the two endpoints to two new vertices, representing e. Such an $(1+\frac{1}{k})$-edge is then *marked.* In every face at most one $(1+\frac{1}{k})$-edge is marked. If there is no such matched $(1+\frac{1}{k})$-edge in H_1, then if the edge of weight $\frac{3}{4}$ is matched in H_1 (this occurs when F_1 or F_2 has only one 1-edge), then e is matched with the I-edge, otherwise the 2-edge e remains unmatched, and we call the corresponding faces *unmatched.* Call the resulting matching in H_1 the 1-matching. The required matching in G' is obtained by taking together the 2-matching and the 1-matching. We call the resulting algorithm APPROX_1-2-M.

5.2 Analysis of APPROX_1-2-M

Due to the lack of a $2 \sim 0$ matching component it easily follows that APPROX_1-2-M does not always compute solutions, which leads to a minimum number of matching and extra edges (see figure 8 for an example). But since the other matching components correspond exactly with the characteristics, there is always a matching in H_2 and a matching in H_1 that corresponds with an optimal solution. In this section we count the number of matching and extra edges in APPROX_1-2-M and the number of matching and extra edges in an optimal solution of the 1-2-matching. We show that the number of (matching plus extra) edges computed by APPROX_1-2-M is at most 5/4 times the optimal number of (matching plus extra) edges.

The non-optimal solution arises from the edges with only characteristic $2 \sim 0$ in G', and therefore we assume w.l.o.g. that all 2-edges in G' are edges with characteristic $2 \sim 0$. We will call unmatched 2-edges in the 2-matching u2-edges. These u2-edges, which remain unmatched in the 1-matching, are called u1-edges. To count the number of involved matching and extra edges for the u2-edges and the u1-edges in APPROX_1-2-M and in the optimal solution, we inspect all the possibilities of paths P in H_2 and H_1 of alternating matched and unmatched edges, with the different weights. We test whether changing the matched and unmatched edges leads to a better solution. If so, then we assign the extra edges to the matched edges of P and count the edges in the new solution. Since the optimal solution also corresponds with a matching in H_2 and H_1, this technique ensures us that we reach the optimum. The different cases will be proved by lemma's. We prove for these cases that the following property *Prop* holds:

Prop: The number of involved matching and extra edges, implied by a path P of alternatingly matched and unmatched edges in H_2 or H_1 is $\leq 5/4$ times the number of involved matching and extra edges in the optimal solution.

Lemma 5.1 *If an u1-edge is matched with a 2-edge in the optimal solution, then* Prop *holds.*

Proof: Let e_1 be an u1-edge, which is matched with a 2-edge e_2 in the optimal solution. Then there must be a 2-edge e_k, which is matched in H_2 by APPROX_1-2-M, but e_k is not matched with a 2-edge in the optimal solution, otherwise our H_2-matching was not maximum. If $a(e_k) = 0$, then two extra edges are required for e_k in the optimal solution, so assume w.l.o.g. that $a(e_k) = 1$, hence $a(e_1) = 1$. Hence if S_1 is the set of u1-edges $e_i, a(e_i) = 1$, matched with a 2-edge in the optimal solution, and S_2 is the set of 2-edges $e_j, a(e_j) = 1$, matched in H_2 but not in the optimal solution, then $|S_1| \leq |S_2|$. Otherwise again our H_2-matching was not maximum.

So for each 2-edge $e_i \in S_1$ we can assign an unique 2-edge $e_j \in S_2$. Inspect such a pair e_i, e_j, with $a(e_i) = a(e_j) = 1$. Assume e_j is matched with 1-edges e_{j_1} and e_{j_2} in the optimal solution. In our 1-matching, e_{j_1} and e_{j_2} are matched with each other, or with two other vertices. Since e_{j_1} and e_{j_2} are matched with e_j in the optimal solution, we assign the half of the matching edges of e_{j_1} and e_{j_2} to this matching.

Inspect in H_2 all $|S_1|$ paths P between the pairs $e_i \in S_1$ and $e_j \in S_2$, which can have several matching edges in common. Such a path P is of even length k, and implies k matching edges. Furthermore e_1 implies two extra edges, leading to $k + 2 + \frac{1}{2} + \frac{1}{2} = k + 3$ edges. In the optimal solution $k + 2$ edges are required (k edges on P and two for e_k). Since $k \geq 2$, *Prop* holds for every pair e_i, e_j. □

2-edges, which are matched with a 1-edge in APPROX_1-2-M, but with a 2-edge in the optimal solution are also covered by Lemma 5.1, because this matching edge

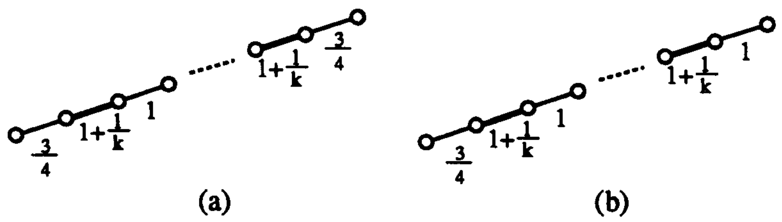


Figure 9: The two different paths, ending in an unmatched edge of weight $\frac{3}{4}$.

between the 1-edge and the 2-edge implies one more matching edge in APPROX_1-2-M and a half more assigned edge in the optimal solution in Lemma 5.1, hence then also *Prop* holds.

Hence we may now further assume that all u1-edges are matched with 1-edges in the optimal solution. In the analysis it is already sufficient to inspect only paths P in the 1-matching. This means that though the 2-matching may not be equal to the matchings between the 2-edges in the optimal solution, these edges will not imply a difference in the number of involved matching and extra edges. Therefore we assume that our 2-matching is equal to the matching between the 2-edges in the optimal solution and we ignore these matched 2-edges and matching edges. The analysis is now completed by the following two lemma's:

Lemma 5.2 *If an u1-edge is matched with one 1-edge in the optimal solution, then* Prop *holds.*

Proof: If in APPROX_1-2-M the 2-edge is not matched with the 1-edge, then there is an unmatched edge of weight $\frac{3}{4}$ in H_1, which should be matched. Inspect a path P in H_1, ending in an unmatched $\frac{3}{4}$-edge, containing no unmatched $(1+\frac{1}{k})$-edges (because we inpect this case in lemma 5.3). In worst-case the length of P, denoted by p, is odd, because then both begin- and endpoint are unmatched by APPROX_1-2-M, while they are matched in an optimal solution, so assume p is odd.

If the last edge e is of weight $\frac{3}{4}$, then P must contain matched $(1+\frac{1}{k})$-edges with *extra* weight $\geq 1/2$ (see Figure 9(a)). In worst-case these $(1+\frac{1}{k})$-edges are unmarked, and the corresponding 2-edge of e is not matched. This implies two extra edges for both unmatched 2-edges of the $\frac{3}{4}$-edges and $\frac{p-1}{2}$ matching edges on P. In the optimal solution the matched and unmatched edges are changed, yielding $2+\frac{p-3}{2}+2$ edges, hence *Prop* holds, since $p \geq 3$.

If e is of weight 1 (see Figure 9(b)), then $p \geq 5$ holds, and by a similar counting argument it follows that the number of involved edges is now at most $\frac{p+5}{p+3}$ times optimal, hence then also *Prop* holds. □

Lemma 5.3 *If an u1-edge is matched with two 1-edges in the optimal solution, then* Prop *holds.*

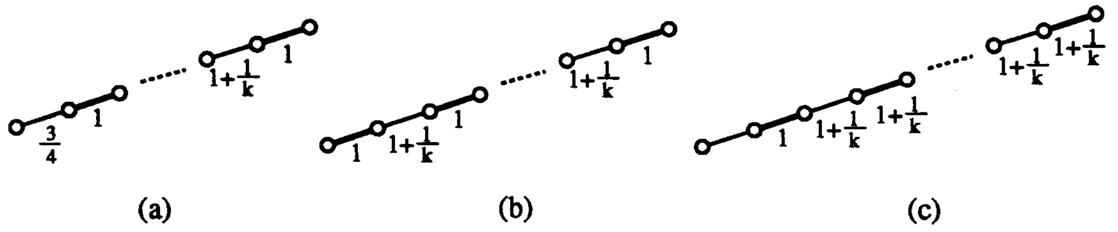


Figure 10: The three different paths, in which the $(1+\frac{1}{k})$-edges are not matched.

Proof: The 2-edge is not matched in the 1-matching, hence the corresponding $(1+\frac{1}{k})$-edges are not matched. There are three possibilities (or combinations of it) in which the $(1+\frac{1}{k})$-edges are not matched, as given in Figure 10.

1. p is of length $2h, h \geq 2$, and contains h matched edges of weight 1, one unmatched $\frac{3}{4}$-edge and $h-1$ unmatched edges of weight $1+\frac{1}{k_i}$, with $\sum_{i=1}^{h-1}(1+\frac{1}{k_i}) \leq h-1+\frac{1}{4}$.
2. p is of length $2h+1, h \geq 1$, and contains $h+1$ matched edges of weight 1 and h unmatched edges of weight $1+\frac{1}{k_i}$, with $\sum_{i=1}^{h-1} 1+\frac{1}{k_i} \leq h+1$.
3. p is of length $2h+1, h \geq 1$, and contains h_1 matched $(1+\frac{1}{k_i})$-edges and h_2 unmatched $(1+\frac{1}{k_j})$-edges, with $\sum_{i=1}^{h_1} 1+\frac{1}{k_i} \geq \sum_{i=1}^{h_2} 1+\frac{1}{k_j}$.

In the unmatched faces of APPROX_1-2-M two extra edges are required for the 2-edge, while in the optimal solution this 2-edge is matched with two 1-edges in worst-case. We assign the cost of these extra edges to the k matched 1-edges of this face, hence every matching edge with one endpoint in an unmatched face has to *pay* for $\frac{2}{k}$ extra edges. Similar in the optimal solution one matched $(1+\frac{1}{k})$-edge is changed into two edges to the 2-edge. We assign this matching edge to the k 1-edges in this face.

In case 1 $\sum_{i=1}^{h-1}\frac{1}{k_i} \leq 1\frac{1}{4}$ holds. There are already $\frac{2}{k_i}$ extra edges assigned to the $2(h-1)$ endpoints. Also for the u1-edge corresponding to the $\frac{3}{4}$-edge two extra edges are required, leading to a total of $h-1+2+\sum_{i=1}^{h-1}\frac{4}{k_i}$ edges. In the optimal solution the $(1+\frac{1}{k_i})$-edges are matched, and $\frac{2}{k_i}$ edges are assigned to each matching edge. Since for the first $\frac{3}{4}$-edge two edges are required and for the last vertex one extra edge is required, it follows that *Prop* holds, because $h \geq 2$.

Furthermore all other matching edges, not on this path P, but having an endpoint on these unmatched faces, have to pay for $\frac{2}{k_i}$ extra edges. In the optimal solution $\frac{1}{k_i}$ edges are assigned to these matching edges. Let K be the size of this set, then $K \geq \frac{1}{2}\sum_{i=1}^{h-1}(k_i-2)$. If $k_i \leq 5$, then $h \leq 2$, and it easily follows by counting all edges in APPROX_1-2-M and in the optimal solution, that always a performance guarantee of at least 5/4 holds, hence we assume that $k_i \geq 6$ holds.

The cost for these matching edges is now $\sum_{i=1}^{h-1} \frac{2}{k_i}(k_i - 2) + K$ and in the optimal solution $\sum_{i=1}^{h-1} \frac{1}{k_i}(k_i-2)+K$, hence this is now $(\sum_{i=1}^{h-1} \frac{2}{k_i}(k_i-2)+K)/(\sum_{i=1}^{h-1} \frac{1}{k_i}(k_i-2)$ $+K) = 1 + (\sum_{i=1}^{h-1} \frac{1}{k_i}(k_i - 2))/(\sum_{i=1}^{h-1} \frac{1}{k_i}(k_i - 2) + K)$. But since $k_i \geq 6$ was assumed and $K \geq \frac{1}{2}\sum_{i=1}^{h-1}(k_i - 2)$ holds, it follows that

$$1 + \frac{\sum_{i=1}^{h-1} \frac{1}{k_i}(k_i-2)}{\sum_{i=1}^{h-1} \frac{1}{k_i}(k_i-2)+\sum_{i=1}^{h-1} \frac{1}{2}(k_i-2)} \leq 1 + \frac{\frac{1}{6}\sum_{i=1}^{h-1}(k_i-2)}{\frac{1}{6}\sum_{i=1}^{h-1}(k_i-2)+\frac{1}{2}\sum_{i=1}^{h-1}(k_i-2)} = 1 + \frac{1/6}{1/6+1/2} = 5/4,$$

hence here also the number of involved matching and extra edges in APPROX_1-2-M is at most 5/4 times the number of involved matching and extra edges in the optimal solution.

Proving case 2 goes analog to the proof of case 1.

In case 3 path P contains matched $(1+\frac{1}{k})$-edges, which are unmarked, and P contains unmatched $(1 + \frac{1}{k})$-edges, belonging to unmatched faces in worst-case. In the optimal solution the matched and unmatched $(1 + \frac{1}{k})$-edges are changed. If K such matched $(1 + \frac{1}{k})$-edges can be changed with K unmatched $(1 + \frac{1}{k})$-edges, then this leads to K less unmatched faces. The K matched $(1 + \frac{1}{k})$-edges come from $K' \leq K$ matched faces. In each of these faces at least one $(1 + \frac{1}{k})$-edge must remain matched (otherwise here extra edges are required). Hence in the optimal solution the 2-edges in $K + K'$ faces are matched, each with two 1-edges, thus $2(K + K')$ matching edges. In APPROX_1-2-M the 2-edges in the K' faces are matched, each with two 1-edges. Furthermore in these K' faces there are $K \geq K'$ unmarked matched $(1 + \frac{1}{k})$-edges, and there are K unmatched faces, requiring two extra edges each. Hence this leads to a total of $2K + K' + 2K'$ edges in APPROX_1-2-M, thus *Prop* holds, since $K' \leq K$. □

Theorem 5.4 APPROX_1-2-M *works within 5/4 times optimal in $O(n^3)$ time.*

Proof: It is shown in Lemma 5.1, 5.2 and 5.3, that in the three different possibilities for the extra edges of ul-edges in APPROX_1-2-M, the number of involved matching and extra edges in APPROX_1-2-M is at most 5/4 times the number of involved matching and extra edges in the optimal solution. In APPROX_1-2-M we may count edges twice in the paths P, but then we have counted them also twice in the optimal solution. Hence APPROX_1-2-M works within 5/4 times optimal.

For the time complexity we note that the the matching graphs can be constructed in $O(n^2)$ time. We can rewrite the degree constrained subgraph problem to finding a maximum cardinality matching [16], which can be solved in $O(\sqrt{|V_{H_2}|} \cdot |E_{H_2}|)$ time [12]. The 2-matching is equal to finding a maximum weighted matching, which can be implemented to run in $O(|V_{H_1}| \cdot |E_{H_1}| + |V_{H_1}|^2 \cdot \log |V_{H_1}|)$ time [6]. Since $|V_{H_1}| = O(|V'_G|), |V_{H_2}| = O(|V'_G|), |E_{H_1}| = O(|V'_G|^2)$ and $|E_{H_2}| = O(|V'_G|^2)$ worst-case, this yields an $O(n^3)$ running time for the algorithm APPROX_1-2-M. □

6 Planar Triconnectivity Augmentation

In this section we deal with the question how to augment a biconnected graph such that the augmented graph is triconnected and still planar. Triconnected planar graphs have nice characteristics, e.g., they have only one embedding in the plane and they can be drawn with convex faces [19]. Drawing biconnected graphs with as much as possible convex faces is rather difficult, as stated in the following theorem.

Theorem 6.1 *Deciding whether a biconnected planar graph can be drawn with $\geq K$ convex faces is NP-complete.*

Proof: (i) The problem is in NP: assume G has $F \geq K$ faces and pick $L \leq F - K$ faces, triangulate these faces by adding additional edges and then check in polynomial time if the resulting graph admits the constraints of theorem 5.1 of [18], which is a characterization of the planar graphs, that can be drawn with convex faces. If this is the case then G can be drawn with $\geq F - L$ convex faces.

(ii) We use a transformation from the vertex cover problem on triconnected planar graphs (which is easily shown to be NP-complete, using the reduction in [5]). Let a triconnected graph $G = (V_G, E_G)$ be given. Inspect the dual graph H of G.[1] Note that H is also triconnected and hence has exactly one planar embedding, which can be drawn with convex faces [18]. We construct a new graph H' by changing every edge (a, b) in H by $(a, ab_1), (ab_1, b), (a, ab_2)$ and (ab_2, b) in H', by introducing two new vertices ab_1 and ab_2. We now claim that H' has an embedding with $\geq K$ convex faces, if and only if G has a vertex cover of size $\leq K$. Suppose that (a, b) in H belongs to the faces F_1 and F_2, then F_1 or F_2 cannot be drawn convex in H'. So we have to find in H the smallest possible set S of faces such that for each edge (a, b) in H, belonging to the faces F_1 and F_2, F_1 or F_2 belongs to S. Then those faces are drawn non-convex in H'.. Notice that the number of faces in H is equal to the number of vertices in G and every set S of faces in H corresponds to a vertex cover of G. Hence there is a vertex cover in G of size $\leq K$ if and only if there is set S of faces of size $\leq K$ in H if and only if there is a set of non-convex faces of size $\leq K$ in H'. The construction is easily computable in polynomial time, so the problem whether there exist a drawing such that the biconnected planar graph can be drawn with $\leq K$ non-convex faces is NP-complete. □

For this and theoretical reasons we consider the Planar Triconnectivity Augmentation Problem for biconnected planar graphs. In this section we present an approximation algorithm, in which we will use the approximation algorithm APPROX_1-2-M in section 5 for the 1-2-matching problem. In the remainder of this section, suppose that G is a biconnected, planar graph. We may assume that G contains at least two triconnected components (or, shortly *tricomps*). We consider the

[1]The vertices of the dual graph H of a planar graph G are the faces of G, and there is an edge between two vertices F_1 and F_2 in H if and only if the faces F_1 and F_2 have a common edge in G.

problem to add a (close to) minimum number of edges to G, such that the resulting graph is planar and triconnected.

A pair of tricomps that have an added edge between them in the resulting graph is said to be *matched* by a *matching edge*. We identify a maximal number of the added edges as matching edges, such that no tricomp is adjacent to more than one matching edge. All other added edges are called *extra edges*. Note that we need exactly one extra edge per component that is not matched by a matching edge. Every tricomp V_1 is connected at two cutvertices, say a and b, with the remaining part of the graph, $G - V_1$, and belongs to two faces of $G - V_1$. It is possible that other tricomps are also connected at both a and b with the remaining part of the graph. We call this situation a *parallel chain* of tricomps. Another possibility is that a second tricomp V_2 is connected at b and c with $G - V_2$, a third tricomp V_3 at c and d, etc. We call this a *serial chain* of tricomps. Notice that a serial chain can be part of a parallel chain, which again can be part of a serial chain, etc.

To abstract the tricomps in a correct way, we introduce *typical characteristics*. We represent the original graph G, and also partly augmented graphs of G later by a graph G' that is obtained by taking an edge for every tricomp. Each edge e in G' gets associated with it a set of typical characteristics, denoted by $C(e)$.

Initially every tricomp V' in G is represented by an edge e in G' between the corresponding two cutvertices a and b, with the characteristic $1 \sim 0$ associated with it, $C(e) = C((a, b)) = \{1 \sim 0\}$ (see Figure 11). $1 \sim 0$ means that this tricomp V' (possibly one vertex v with $deg(v) = 2$) must get one additional outgoing edge to become triconnected. All other added edges e in G are represented by an edge in G' with $C(e) = \{0 \sim 0\}$. In general, a characteristic $i \sim j$ in the set of characteristics means that i augmentation edges must go from the component to one face, and j edges must go to the other face.

Let a *2-subgraph* in this reduced graph G' be a serial or parallel chain or a tricomp. For a 2-subgraph V' we want to add matching edges between the edges with characteristics, representing the number of outgoing edges at both sides of the corresponding tricomps. V' must also be triconnected with $G - V'$ and, hence, must also get typical characteristics associated with it. We try to add as few as possible (outgoing) edges between tricomps in V' such that V' is triconnected except some unmatched tricomps, which must get outgoing edges to $G - V'$. Notice that V' may contain several typical characteristics. Instead of characteristic $2 \sim 1$ one could have $1 \sim 0$ (see Figure 11), and similarly instead of $2 \sim 2$ one could have $2 \sim 0, 1 \sim 1$, or both.

The following lemma is crucial for our algorithm.

Lemma 6.2 *Every typical characteristic $i \sim j$ for a 2-subgraph V' can be changed such that $i \leq 2$ and $j \leq 2$, without increasing the number of extra edges.*

Proof: Suppose there is an optimal solution in which parallel chain V_1 has $k \geq 3$ outgoing edges to another parallel chain W_1 (a similar argument can be

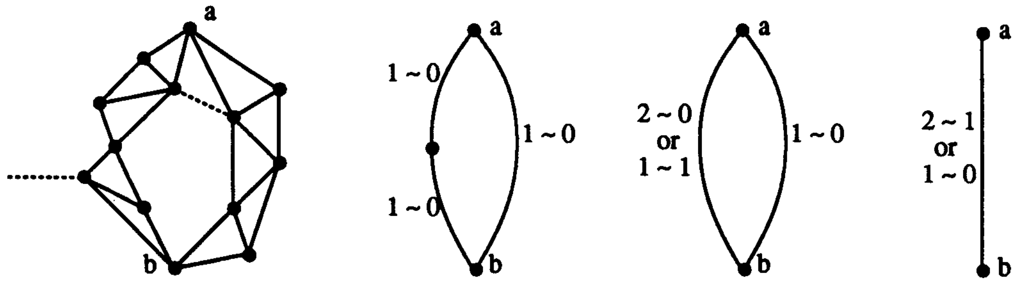


Figure 11: Example of typical characteristics for serial and parallel chains.

applied when V_1 has $k \geq 3$ edges to several parallel chains $W_1, \ldots, W_{k'}$, $k' \leq k$). Assume first that k is odd. We will show that instead of adding k edges between V_1 and W_1, we can connect $k-1$ tricomps of V_1 with each other by $\frac{k-1}{2}$ edges, the same in W_1 and one edge from V_1 to W_1 in such a way that the resulting graph is triconnected again.

Let $T_1, \ldots, T_k$ be the tricomps of V_1 having an extra edge to a tricomp of W_1, hence they share one face, F. Let $T_1, \ldots, T_k$ further belong to faces $F_1, \ldots, F_l$, $l \leq k$, in which tricomps $T_{i_j}, \ldots, T_{i_{j+1}-1}$ belong to face F_j, $1 \leq j \leq l$, with $T_{i_1} = T_1$ and $T_{i_{l+1}-1} = T_k$. hence tricomps $T_{i_j}, \ldots, T_{i_{j+1}-1}$ share two common faces, F and F_j, and hence they form a serial chain of tricomps. We can add a matching edge between T_{i_j} and $T_{i_{j+1}-1}$, between $T_{i_{j+1}}$ and $T_{i_{j+1}-2}, \ldots$, until one or two tricomps are left unmatched. After applying this for all j, $i \leq j \leq l$ we renumber the unmatched tricomps to $T_1, \ldots, T_p$, with $p \leq k$ and p odd. For every pair of consecutive tricomps T_i, T_{i+1}, $1 \leq i < p-1$, we add an edge between T_{i+1} and T_{i+2}, if T_i and T_{i+1} share the same face F_j. We again renumber the unmatched tricomps to $T_1, \ldots, T_{p'}$, with $p' \leq p$ odd. We now add edges between T_{2i} and T_{2i+1}, for $1 \leq i \leq \frac{p'-1}{2}$. T_1 remains unmatched. We apply the similar argument to W_1, leaving there one tricomp T_1' unmatched, and we add an edge from T_1 in V_1 to T_1' in W_1.

Every serial chain $T_{i_j}, \ldots, T_{i_{j+1}-1}$ has one outgoing edge to another serial chain $T_{i_{j'}}, \ldots, T_{i_{j'+1}-1}$, $j' \neq j$, but since they belong to different faces F_j and $F_{j'}$, they are now triconnected. This holds for every serial chain of V_1, and similar for the tricomps of W_1. Adding an edge between the two unmatched tricomps leads to a triconnected graph.

A similar argument can be applied when k is even. □

The main idea of the algorithm is the following: we take either a serial chain, a parallel chain, or a tricomp in the reduced graph G', compute an (approximation of) the optimal way of adding augmenting edges, and reduce the chain or tricomp to a single edge, with the typical characteristic associated with it. In case of parallel or serial chains, we can find the optimal way of adding edges, and is described below. In case of tricomps, we use the approximation algorithm for the 1-2-matching problem as described in section 5. The problem, to find a minimum number of edges to add to this tricomp of G' such that this entire part becomes triconnected, corresponds exactly to the 1-2-matching problem. Note however, that in case the tricomp is not the only tricomp in G', then this tricomp must made triconnected to the remainder of the graph, so some augmenting edges should go outside the tricomp. However, this extra requirement can be taken care of by a small modification of algorithm APPROX_1-2-M. We omit the details. Below, the main loop of the algorithm is described more formally.

```
TRICONNECT
  $G' := G$;
  replace every tricomp $V'$ by an edge $e$ between the cutvertices,
  with $C(e) := \{1 \sim 0\}$; for the other edges $e \in G, C(e) := \{0 \sim 0\}$;
  done := false;
  **while** not done **do**
    **if** $G'$ contains a serial chain $V'$ **then** SERIAL($V'$) **else**
      **if** $G'$ contains a parallel chain $V'$ **then** PARALLEL($V'$) **else**
        do APPROX_1-2-M($V'$), with $V'$ a tricomp of $G'$;
    **if** $G' = V'$ **then** done := true **else**
      replace $V'$ by an edge $(a, b)$ with $C((a, b))$ := the set of typical
      characteristics of $V'$
  **od**
  choose one typical characteristic for $G'$ and from this,
  the unique typical characteristics follow for all tricomps in $G$;
  add corresponding edges between tricomps.
```

We remark, that an invariant of our algorithm is, that for every edge e with $i \sim j \in C(e)$, and $i' \sim j' \in C(e)$, then either both $i + j$ and $i' + j'$ are even or both are odd. Next, we describe the algorithms SERIAL(V') and PARALLEL(V'), which compute the typical characteristics for V', when V' is a serial or parallel chain. Every edge $e_i = (a, b)$ in V' corresponds with a subgraph V_i in G, connected with a and b with $G - V_i$. These subgraphs V_i may contain several tricomps and must get outgoing edges to $G - V_i$ to become triconnected. Therefore we will talk about the subgraph V_i in G that is represented by an edge e_i in G'.

SERIAL(V') replaces a path of vertices $v_1, \ldots, v_k$ of degree 2 in G' by one edge (a, b) with the set of typical characteristics associated with it. If two consecutive edges $(v_{i-1}, v_i), (v_i, v_{i+1})$ both have characteristics $0 \sim 0$, then $deg(v_i) = 2$ in G and, hence, v_i must get an ougoing edge. To accomplish this, we put $C((v_i, v_{i+1})) = \{1 \sim 0\}$. If there remain some edges (a, b) with $C((a, b)) = \{0 \sim 0\}$ after this operation, then we contract these edges (a, b), i.e., we identify a and b and delete the edge (a, b). Note that this does not affect the triconnectivity of any vertex. Now there are no edges with characteristic $0 \sim 0$, hence we may assume that every subgraph must get at least one outgoing edge.

Next we assign to every edge $e \in V'$ the typical characteristic $i \sim j \in C(e)$ for which $i + j$ is maximal. If $C(e) = \{1 \sim 1, 2 \sim 0\}$, then we take an arbitrary one. To ensure that we obtain all possible typical characteristics for V', we apply the algorithm twice, with one characteristic $2 \sim 1$ changed into $1 \sim 0$ for one edge $e \in V'$, if $C(e) = \{2 \sim 1, 1 \sim 0\}$ (or $2 \sim 0$ changed into $1 \sim 1$ or $2 \sim 2$ changed into $1 \sim 1$, if there are edges in V', containing both characteristics). We say that when we *assign* an edge $e \in V'$ to the leftside (rightside), then we assign i to the leftside and j to the other side of V' such that $i \sim j$ is associated with e, with $i \geq j$. The left- and rightside of V' are chosen arbitrary at the beginning. *Left* and *right elements* are vertices of the subgraph corresponding with $e \in V'$, which must get one left or right outgoing edge by the corresponding characteristic to become triconnected. The idea is to flip the edges such that there are alternatingly left and right outgoing edges from the corresponding subgraphs. This can be described formally as follows:

```
SERIAL($V'$)
    let $e_1, \ldots, e_k$ be the edges of $V'$;
    $i := 1; j := 2$;
    Assign $e_i$ to the leftside;
    **while** $j \leq k$ **do**
        assign $e_j$ to the rightside;
        match left elements of $V_j$ with left elements of $V_i, \ldots, V_{j-1}$;
        let $p_1$ be the number of remaining left elements of $V_i, \ldots, V_j$;
        $p_2 := 0; j_2 := j$;
        **while** $p_2 < p_1$ **do**
            $j_2 := j_2 + 1$;
            assign $e_{j_2}$ to the leftside;
            $p_2 := p_2 +$ left element of $V_{j_2}$;
        **od**;
        match $p_1$ left elements with the $p_2$ left elements;
        $i := j; j := j_2$;
        swap(left, right);
    **od**;
    match left elements of $V_j$ with left elements of $V_i, \ldots, V_{j-1}$;
```

Finally, if there remain i elements at one side and j at the other side, then this gives a $i \sim j$ characteristic. If $j = 0$ then by deleting one matching edge we obtain a $i \sim 2$ characteristic. By applying the algorithm twice with a small modification (as explained above), we can get instead of a $1 \sim 1$ characteristic a $2 \sim 0$ characteristic. Let $C(V')$ be the complete set of *all* characteristics of $e \in V'$, and let $S(V')$ be the sum of the elements of the *assigned* characteristics of $e \in V'$, then the following rules are obtained for the typical characteristic set $C((a, b))$ of a serial chain V':

If $S(V') = 1$ then $C((a, b)) := \{1 \sim 0\}$, if $S(V') = 2$ then if $1 \sim 0 \in C(V')$ then $C((a, b)) := \{1 \sim 1, 2 \sim 0\}$, else $C((a, b)) := C(V')$, if $S(V') \geq 3$, then

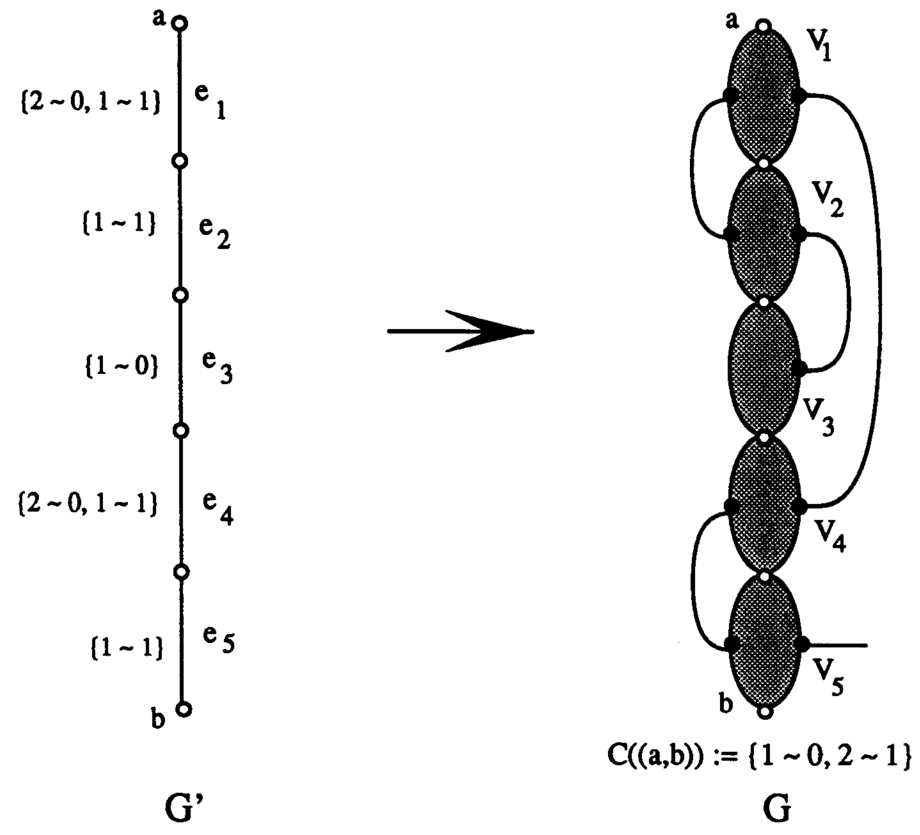


Figure 12: Example of algorithm SERIAL(V'). Elements are given by black dots.

if $S(V')$ is odd, then $C((a,b)) := \{2 \sim 1, 1 \sim 0\}$, else if $1 \sim i \in C(V')$ then $C((a,b)) := \{2 \sim 2, 2 \sim 0, 1 \sim 1\}$, else $C((a,b)) := \{2 \sim 2, 2 \sim 0\}$.

Theorem 6.3 SERIAL*(V') computes the set of typical characteristics in linear time for a serial chain V'.*

Proof: We only have to show that the matching edges in SERIAL(V') makes V' triconnected, except these tricomps of V', which must get outgoing edges to $G - V'$. For this we note that every time we start the first while-loop e_i is assigned to the left and e_j is assigned to the right, with $j \geq i$. Then we add left matching edges between $V_i, \ldots, V_{j-1}$ and $V_j, \ldots, V_{j_2}, j_2 \geq j$. There goes a right matching edge between V_j and V_p, with $p > j_2$. This means that if we delete three cutvertices in $V_i, \ldots, V_{j_2}$, then we can always still reach the remaining part of the graph via V_i, V_j or V_{j_2}. This holds for all i, j, j_2, hence the augmented graph of V' is triconnected. □

In figure 12 an example of applying the algorithm SERIAL(V') is given.

Let now V' be a parallel chain, which essentially is a set of parallel edges $e_1, \ldots, e_k$, with characteristics associated to them. We may permute these edges in any order and we have to find an order such that adding a minimum number of matching edges between the corresponding subgraph in G makes V' triconnected. If $e_1, \ldots, e_k$ is the optimal order then we have to add augmenting edges between the corresponding subgraphs V_i and V_{i+1}, $1 \leq i < k$. If both V_1 and V_k have outgoing edges, then for exactly one j, $1 \leq j < k$, no augmenting edges to V_{j+1} are required. This place is called a *gap* and if V_1 (V_k) has no outgoing edges to a component outside the chain, then the gap is said to be at V_1 (V_k).

We now put every subgraph $V_1, \cdots, V_k$ in one of the sets K_1, K_2, K_3, K_4, and K_5, such that V_i is placed in K_j, if the edge e_i representing V_i fulfills $1 \sim 1 \in C(e_i)$ ($j = 1$), $2 \sim 2 \in C(e_i)$ ($j = 2$), $2 \sim 1 \in C(e_i)$ ($j = 3$), $1 \sim 0 \in C(e_i)$ ($j = 4$), $2 \sim 0 \in C(e_i)$ ($j = 5$), and every subgraph is placed in the set K_j with smallest possible index j. The algorithm PARALLEL is based on the following observations:

If at some place in the optimal order π one matching edge e is added between two consecutive tricomps $V_{\pi(i)}$ and $V_{\pi(i+1)}$, then we can delete e and add the set K_1 of $1 \sim 1$ tricomps between $V_{\pi(i)}$ and $V_{\pi(i+1)}$, with matching edges between every pair of consecutive tricomps. Similar can be done for K_2, if there are added two matching edges between $V_{\pi(i)}$ and $V_{\pi(i+1)}$. If $|K_4| + |K_5| > 2$ then extra edges are required, because otherwise there occurs more than one gap. Therefore, as long as $|K_4| > 2$ holds, we can add an extra edge between two $1 \sim 0$ edges, and treat this united tricomps as a $1 \sim 1$ tricomp, hence adding it to K_1. We can do similarly for K_5, adding the united tricomp to K_2, so we assume w.l.o.g. that $|K_4|, |K_5| \leq 2$. If $|K_4| + |K_5| > 2$ then we union a $1 \sim 0$ tricomp and a $2 \sim 0$ tricomp by an extra edge to a $2 \sim 1$ tricomp, thereby increasing the set K_3. If $|K_3|$ is even then we can union it optimally by matching edges to one $2 \sim 2$ or $1 \sim 1$ tricomp (adding it to K_1 or K_2), otherwise we can union it optimally to one $2 \sim 1$ tricomp. If $|K_4| + |K_5| = 2$ then one gap occurs inside the optimal order; if $|K_4| = 1$ or $|K_5| = 1$ then the gap occurs at V_1 or V_k, (i.e., the resulting characteristic will be of the form $0 \sim i$, for $i \in \{1, 2\}$.)

Except for this we also inspect the edges e with $|C(e)| \geq 2$ to get the complete set of typical characteristics without increasing the extra edges. This plus a tedious case analysis of the different sizes of the sets K_i leads to the algorithm PARALLEL(V'), for which the following theorem holds.

Theorem 6.4 PARALLEL(V') *computes the optimal order of the tricomps and the set of typical characteristics for a parallel chain V' in linear time.*

The total time of the algorithm is dominated by the time for the 1-2-matching problem, as the latter costs us $O(n^3)$ time, and executings of SERIAL and PARALLEL both cost linear time. The performance ratio is also dominated by the 1-2-matching problem, as we solve the problem for parallel and serial chains optimally, and have a performance ratio 5/4 for the 1-2-matching problem.

Theorem 6.5 *There exists an approximation algorithm for Planar Triconnectivity Augmentation for biconnected planar graphs that has performance ratio 5/4 and uses $O(n^3)$ time.*

7 Final Remarks

In this paper we considered the problem of adding a minimum number of edges to a planar graph, such that the augmented graph is biconnected or triconnected and still planar. For both problems approximation algorithms are given, but the approximation algorithm for admitting triconnectivity requires the graph to be biconnected. For this, we can apply the algorithm APPROX_1-2-M, but as already noticed by Naor et al. [13], it is not true that *every* optimal set of edges which increases the connectivity by one can be extended to an optimal solution to increase the connectivity by $k > 1$. Hence the performance ratios of 3/2 and 5/4 do not guarantee a performance ratio of 15/8 to make an arbitrary planar graph triconnected without losing planarity. This interesting open problem becomes efficiently solvable, when the inputgraph G is outerplanar. In this case, an elegant linear algorithm can be obtained to augment G by a minimum number of edges such that the resulted graph is planar and triconnected ([9]).

When the number of added edges is not that important, then it becomes interesting to search for simple, linear algorithms to make a graph biconnected or triconnected and planar, working within a constant from optimal. When the number of added edges is irrelevant, we can simple triangulate the planar graph in linear time (see e.g. [10]), and the triangulated planar graph has $3n - 6$ edges.

Instead of considering the planarity constraint for the augmentation problems, we can ask similar questions in which the augmented graph must satisfy some other specified properties, e.g., belonging to a class of outerplanar graphs, perfect graphs or partial k-trees. These open problems are an interesting field for further study and research, to come to a general technique for augmentation algorithms.

Acknowledgements

The authors wish to thank Marinus Veldhorst for some useful suggestions for approximating the 1-2-Matching.

References

[1] Di Battista, G., and R. Tamassia, Incremental planarity testing, *Proc. 30th Annual IEEE Symp. on Found. on Comp. Science*, North Carolina, 1989, pp. 436–441.

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

[3] Frederickson, G.N., and J. Ja'Ja, 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] Garey, M.R., D.S. Johnson and L. Stockmeyer, Some simplified NP-complete graph problems, *Theoret. Comput. Science* 1 (1976), pp. 237–267.

[6] Gabow, H.N., Data structures for weighted matching and nearest common ancestors with linking, in: *Proc. 1st Annual ACM-SIAM Symp. on Discrete Algorithms*, San Fransisco, 1990, pp. 434–443.

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

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

[9] Kant, G., *Optimal linear planar augmentation algorithms for outerplanar graphs*, Techn. Rep., Dept. of Computer Science, Utrecht University, 1991 (to appear).

[10] Kant, G., *On drawing planar graphs with wide angles*, in preparation.

[11] Khuller, S., and R. Thurimella, *Faster approximation algorithms for graph augmentation*, The University of Maryland at College Park, manuscript, 1991.

[12] Micali, S., and V.V. Vazirani, An $O(\sqrt{V} \cdot E)$ algorithm for finding maximum matching in general graphs, in: *Proc. 21st Annual IEEE Symp. Foundations of Computer Science*, Syracuse, 1980, pp. 17–27.

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

[14] 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.

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

[16] Shiloach, Y., Another look at the degree constrained subgraph problem, *Inf. Proc. Lett.* 12 (1981), pp. 89–92.

[17] Tarjan, R.E., A class of algorithms which require nonlinear time to maintain disjoint sets, *J. Comp. Syst. Sci.* 18 (1979), pp. 110-127.

[18] Thomassen, C., Planarity and duality of finite and infinite planar graphs, *J. Combin. Theory, Series B* 29 (1980), pp. 244-271.

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

[20] Woods, D., *Drawing Planar Graphs*, Ph.D. Dissertation, Computer Science Dept., Stanford University, CA, Tech. Rep. STAN-CS-82-943, 1982.