ANGEWANDTE MATHEMATIK UND INFORMATIK
UNIVERSITÄT ZU KÖLN

Report No. 92.107

A fast linear time embedding algorithm based on the Hopcroft-Tarjan planarity test

by

Petra Mutzel

1992

Institut für Informatik
UNIVERSITÄT ZU KÖLN
Pohligstraße 1
D-5000 Köln 51

A fast O(n) Embedding Algorithm based on the Hopcroft-Tarjan Planarity Test

Petra Mutzel

Institut für Informatik, Universität zu Köln

Abstract - The embedding problem for a planar undirected graph $G = (V, E)$ consists of constructing adjacency lists $A(v)$ for each node $v \in V$, in which all the neighbors of v appear in clockwise order with respect to a planar drawing of G. Such a set of adjacency lists is called a (combinatorial) embedding of G. Chiba presented a linear time algorithm based on the 'vertex-addition' planarity testing algorithm of Lempel, Even and Cederbaum using a PQ-tree. It is very complicated to implement this data structure. He also pointed out that it is fairly complicated to modify the linear 'path-addition' planarity testing algorithm of Hopcroft and Tarjan, such that it produces an embedding. We present a straightforward extension of the Hopcroft and Tarjan planarity testing algorithm which is easy to implement. Our method runs in linear time and performs very efficiently in practice.

1. Introduction

A graph G is planar if it can be drawn on a plane with no two edges crossing each other except at their vertices. Such a drawing is called a planar embedding of G. There are essentially two different linear time algorithms for the problem of testing planarity of a graph. Lempel, Even and Cederbaum [LEC] presented the 'vertex-addition' based algorithm using PQ-trees, a very complicated data structure. It was proved to have a linear time implementation in 1976 partly by Even and Tarjan [ET] and partly by Booth and Lueker [BL], by introducing an st-numbering of a graph. The first linear time algorithm was given by Hopcroft and Tarjan [HT] in 1974 using depth-first-search trees. This algorithm, which is 'path-addition' based, is generally considered to be more complicated than the PQ-tree approach.

Many applications require not only testing planarity but also embedding a planar graph in the plane. The embedding problem for a planar undirected graph $G = (V, E)$ consists of constructing adjacency lists $A(v)$ for each node $v \in V$, such that all the neighbors of v appear in clockwise order with respect to a planar drawing of G. Such a set of adjacency lists is called a (combinatorial) embedding of G.

In 1985, Chiba and Nishizeki [CN] presented an algorithm for constructing a combinatorial embedding based on the PQ-tree approach. They wrote: "Hopcroft and Tarjan mentioned that an embedding algorithm can be constructed by modifying their testing algorithm. However, the modification looks to be fairly complicated; in particular, it is quite difficult to implement a part of the algorithm for embedding an intractable path called 'special path'. " In fact, the presence of this 'special path' makes the embedding problem more difficult. Nevertheless, it is possible to give a straightforward solution for the embedding problem based on the Hopcroft and Tarjan planarity test. Our method runs in linear time and space and performs very efficiently in practice.

In section 2 we explain the main idea of the Hopcroft and Tarjan planarity testing algorithm. Also we define some notation and give some basic observations for the combinatorial embedding. The main idea of the embedding algorithm is presented in section 3.1 together with the necessary data structures and the most important procedures. In section 3.2 a proof of the correctness of the embedding algorithm is given. Computational results are presented in section 4.

2. Preliminaries

2.1 Planarity test

In the sequel we will consider a depth first search tree of the graph $G = (V, T, B)$, where V is the set of DFS numbers of the vertices, T is the set of *tree edges* and B the set of *back edges* [M]. G is assumed to be biconnected.

The idea is the following: Suppose we identify a cycle, in the sequel called *spine cycle*, starting in the root (node 1) of the DFS-tree consisting of tree edges followed by one back edge leading to node 1 again. Such a back edge must exist, because of the biconnectedness of G.

Consider the graph in Figure 1. Here C would be the cycle $1 \rightarrow_T 2 \rightarrow_T 3 \rightarrow_T 4 \rightarrow_T 5 \rightarrow_T 6 \rightarrow_B 1$. Our aim is to get a plane embedding of G. We can avoid one intersection by embedding the edge $5 \rightarrow 11$ outside the cycle C. But then also edge $11 \rightarrow 2$ has to be embedded outside C. We notice that there are some dependencies between edges with respect to the cycle C. After removing the cycle C from G, the graph $G \setminus C$ falls apart in several 'dependency components', called the *segments* with respect to cycle C.

In a first step (A) the planarity of every segment $S(e_i)$ together with the cycle C is tested, before in a second step (B) the embeddings of the segments are merged together to get one planar embedding of $C \cup S(e_1), \ldots \cup S(e_k)$, if possible. In order to solve (A) we determine a new cycle in $C \cup S(e_i)$, remove this one and repeat the above steps. So, this idea leads us to a recursive algorithm. To describe this idea more detailed, a few definitions will be necessary.

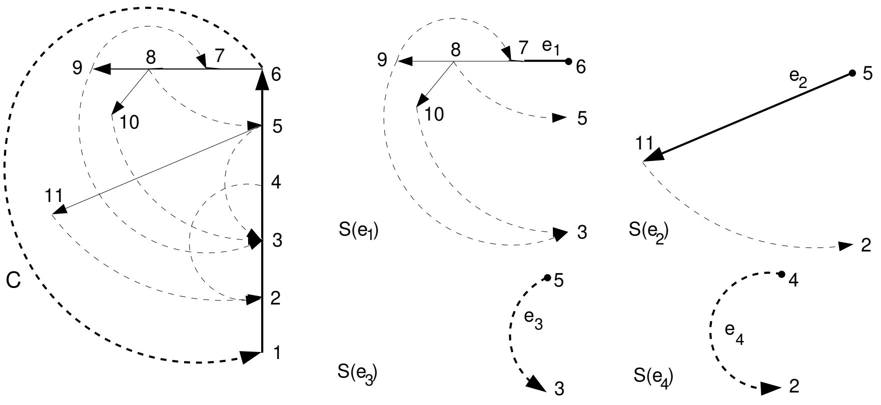


Figure 1. Depth First Search Tree G - C has 4 components =: 'segments'

Let $E = B \cup T$ and $e = (x, y) \in E$ emanating from the spine cycle C. With e we associate a *segment* $S(e)$, a *daughter cycle* $C(e)$, which is the spine cycle of the next recursion step, the set of *attachments* $A(e)$ and the set $low(e)$ with respect to C as follows:

If $e \in B$ then $S(e)$ and $C(e)$ are defined to be the union of e and the tree path from y to x. The *set of attachments* $A(e)$ is the set $\{x, y\}$ and $low(e) = \{y\}$. If $e \in T$, then $S(e)$ is the subgraph of G that consists of e, the subtree T_e of T rooted at y, and all the back edges that emanate from vertices V_e of T_e. $low(e)$ is defined to be the lowest endpoint of all back edges starting in V_e. The *set of attachments* $A(e)$ is the set of endpoints of all back edges in $S(e)$ which end in tree ancestors of x. The daughter cycle $C(e)$ is the tree path from $low(e)$ to one node $w \in V_e$ with $(w, low(e)) \in B$ together with the back edge $(w, low(e))$. Notice that there may be several choices of this cycle at this point. In (2.1) an exact choice of the correct cycle will be given, but it is still not unique. This fact will cause the major trouble in the embedding algorithm.

The *daughter cycle* $C(e)$ is divided into its *stem* $low(e) = w_0 \rightarrow_T \cdots w_r$, the *initial edge* $e = (w_r, w_{r+1})$, its *spine path* $w_{r+1} \rightarrow_T \cdots w_k$ and its only back edge (w_k, w_0) (see Figure 2). The spine path of the root cycle starting in node 1 consists of the path $1, \ldots, w_k$ where $(w_k, 1) \in B$ (In this case the stem is empty). The initial edge e of a segment $S(e)$ starting at the spine path of C is called *daughter edge* of the cycle C.

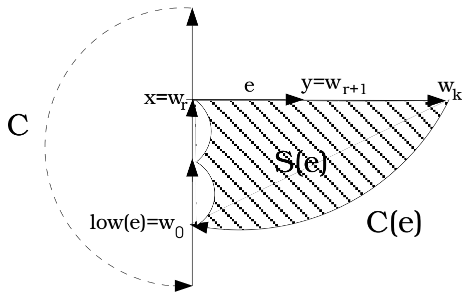


Figure 2

Let us consider Figure 1 again. Here the daughter cycle $C(e_1)$ with $e_1 = (6,7)$ of C could be the stem $3 \rightarrow_T 4 \rightarrow_T 5 \rightarrow_T 6$ together with either the spine path $7 \rightarrow_T 8 \rightarrow_T 9$ and back edge $(9,3)$ or the one with spine path $7 \rightarrow_T 8 \rightarrow_T 10$ and back edge $(10,3)$. The set of attachments of $S(e_i)$ are $A(e_i) = \{3,5,6\}$ and $low(e) = \{3\}$.

A segment $S(e)$ is called *strongly planar* if there exists a planar embedding of $S(e)$, in which the stem of the daughter cycle $C(e)$ borders the outerface. The correctness of the following lemma is obvious if you consider Figure 2 (see [M]).

Lemma 1

Let e be an edge emanating from a spine cycle C. Then $C + S(e)$ is planar if and only if $S(e)$ is strongly planar.

Task (A) is solved if we will have an algorithm which tests strong planarity. Now, suppose that all of the k segments $S(e_i)$ are strongly planar. Under what conditions can they be combined to one planar embedding of $C \cup S(e_1) \cup \ldots S(e_k)$?

Let e and e' emanate from a spine cycle C. Then the segments $S(e)$ and $S(e')$ *interlace* if either there exist vertices $x, y, z, u \in C$ with $x < y < z < u$ and $x, z \in A(e)$ and $y, u \in A(e')$ or $A(e)$ and $A(e')$ have 3 points together (see Figure 3).

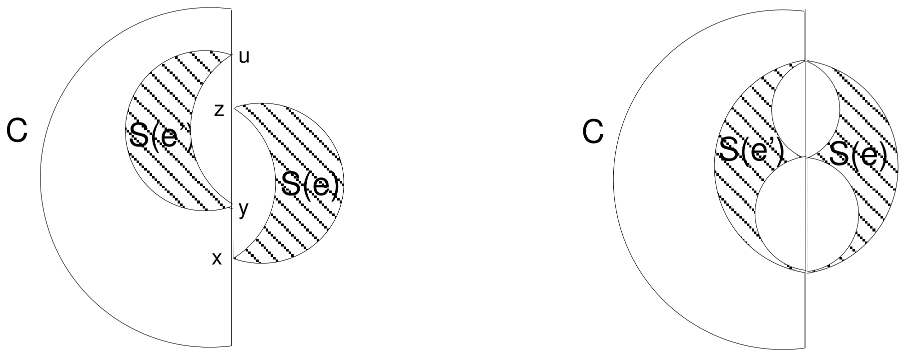


Figure 3. Interlacing segments S(e) and S(e')

It is intuitively obvious and follows from the Jordan Curve Theorem that interlacing segments cannot be embedded on the same side of the cycle C. It is easy to construct the *interlacing graph* $IG(C)$ with respect to C: The vertices of this graph correspond to the segments $S(e_i)$, where the edges e_i, for $i = 1, \ldots, m$ are all the edges leaving the spine path of cycle C. Two nodes will be connected by an edge if and only if the segments corresponding to these nodes interlace. The following lemma answers the above question.

Lemma 2

Let e be a tree edge, $C(e) = w_0 \rightarrow_T w_1 \rightarrow_T \ldots \rightarrow_T w_k \rightarrow_B w_0$ and $e = (w_r, w_{r+1})$. Let $e_1, \ldots, e_m$ be all the daughter edges of $C(e)$ leaving the cycle in nodes w_j, $r < j \leq k$. Then

$S(e)$ is planar if and only if $S(e_i)$ is strongly planar for all $i = 1, \ldots, m$ and $IG(C(e))$ is bipartite. More specifically, there exists a partition L, R of the set $S\{(e_1), \ldots, S(e_m)\}$ such that no pair of segments of L or R interlaces.

Moreover, segment $S(e)$ is strongly planar if and only if $S(e)$ is planar and for every component D of $IG(C(e))$ either $w_1, \ldots, w_{r-1} \cap \bigcup_{S(e) \in D \cap L} A(e) = \emptyset$ or $w_1, \ldots, w_{r-1} \cap \bigcup_{S(e) \in D \cap R} A(e) = \emptyset$ is satisfied.

Proof: see [M].

Lemma 2 is intuitively obvious, although the proof of it is complicated. The condition in the last part of the lemma tests exactly the fact if the stem of the spine cycle $C(e)$ borders the outerface. So, for example, for the segment $S(e)$ in Figure 4 the condition is not satisfied, which is equivalent to the fact that $S(e)$ is not strongly planar. This lemma already suggests the algorithm: In order to test strong planarity of a segment $S(e)$, just test strong planarity for all the segments $S(e_i)$, $i = 1, \ldots, m$, construct the interlacing graph $IG(C(e))$ and test the additional condition. For the efficiency of the algorithm it is important to sort the adjacency lists of the nodes initially. We define the following weights $c((v, w))$ for all edges $e = (v, w)$. Note that each node is identified with its DFS-number.

$$(2.1) \qquad c((v,w)) := \begin{cases} 2 * w, & \text{if } (v,w) \in B \\ 2 * \mathit{low1}(w), & \text{if } (v,w) \in T \text{ and } \mathit{low2}(w) \geq v \\ 2 * \mathit{low1}(w) + 1, & \text{if } (v,w) \in T \text{ and } \mathit{low2}(w) < v \end{cases}$$

where $\mathit{low1}(w) := \min(\{v\} \cup \{z \mid w \rightarrow_T^* x \rightarrow_B z\})$ and $\mathit{low2}(w) := \min(\{v\} \cup \{z \mid w \rightarrow_T^* x \rightarrow_B z \text{ with } z \neq \mathit{low1}(w)\})$ ($w \rightarrow_T^* x$ denotes the tree path from node w to node x). Then we sort the adjacency list of each node in increasing order by these weights. Thus edge $e = (w_j, z)$ is considered before edge $e' = (w_j, z')$ if either $low(e) < low(e')$ or if $low(e) = low(e')$ together with $|A(e)| = 2$ and $|A(e')| \geq 3$ is satisfied.

In the sequel we assume that the adjacency lists are sorted as described above. The spine cycle of a segment $S(e) = S((w_r, w_{r+1}))$ is now easy to construct. Start with node w_{r+1} and call the first node in its adjacency list w_{r+2}. In the adjacency list of w_{r+2} we find at the first position w_{r+3}, and so on, until we reach a back edge (w_k, w_0). This back edge closes the cycle $C(e)$. Note that $w_0 = low(e)$ and the spine path of the segment $S(e)$ is $w_{r+1} \rightarrow_T \cdots w_k$. The planarity testing algorithm can be described as follows.

Procedure stronglyplanar(e);

{ Tests if the segment $S(e)$, $e = (w_r, w_{r+1})$ is strongly planar. In this case the output is the ordered list of attachments of $S(e)$ }

```
(1) Construct the spine path of the segment $S(e)$ in the following way: Start in node
    $w_{r+1}$ and continue taking the first neighbor on every adjacency list until a back edge
    $(w_k, w_0)$ is encountered. The spine path is now $w_{r+1}, w_{r+2}, \dots, w_k$.
(2) For every edge $e_i$ emanating from the spine path of $C(e)$ at $w_k, \dots, w_{r+1}$ do
(3)     Call stronglyplanar($e_i$);
(4)     If $S(e_i)$ is not strongly planar, then STOP { $G$ is not planar };
(5)     Add $S(e_i)$ to the interlacing graph $IG$
(6)     If the interlacing graph is not bipartite, then STOP
(7) { $C(e) + S(e_1) + \dots + S(e_m)$ is planar }
    If $C(e) + S(e_1) + \dots + S(e_m)$ is not strongly planar, i. e. the additional condition of
    Lemma 2 is not satisfied, then STOP { $G$ is not planar }
```

This is just a basic version of the planarity testing algorithm of Hopcroft and Tarjan. There are a few questions left open: The interlacing graph may have a quadratic number of edges. In order to implement the algorithm in linear time and space it is not necessary to store the interlacing graph specifically. Moreover, we have not yet described how to test bipartitness in constant time. But it is not necessary to know such details in order to understand the embedding algorithm. For interested readers [M] and [T] is recommended.

It is important though to understand the output of the planarity testing algorithm. The procedure *stronglyplanar*(e) constructs the components of the interlacing graph $IG(C(e))$. Every component D consists of the two interlacing sides $L(D)$ and $R(D)$. Let $e_1, \dots, e_m$ be all the daughter edges emanating from the cycle $C(e)$. The function *alpha* is defined by $alpha : \{S(e_1), \dots, S(e_m)\} \to \{LS, RS\}$ with the condition that the function values of all pairs of interlacing segments are different.

So, for every segment $S(e_i)$ we get an entry $alpha(S(e_i))$ which puts the interlacing segments on different sides of the cycle $C(e)$ in the way that

$$(2.2)\quad \begin{array}{lll} alpha(S(e_i)) = RS \text{ implies that} & A(e_i) \in \{w_0\} \cup \{w_r, w_{r+1}, \dots, w_k\} \text{ and} \\ alpha(S(e_i)) = LS \text{ that} & A(e_i) \in \{w_0, w_1, \dots, w_{r-1}, w_r, \dots w_k\}. \end{array}$$

Consequently, in most of the cases except a special one (see Observation 3), every segment with the entry $alpha(S(e_i)) = RS$ can be embedded on the outer side of the cycle $C(e)$. (It is no problem to embed them inside the cycle $C(e)$. But segments with the entry $alpha(S(e_i)) = LS$ cannot be embedded on the outer side of the cycle in general, like $S(e_i)$ in Figure 4).

2.2 Preliminaries for the embedding

Let $C(e)$ be a spine cycle, $e = (v_0, v_1)$, $e_i = (x, y)$ an edge emanating from the spine path of $C(e)$ and z be an attachment of the segment $S(e_i)$. z is called *normal* if $z > low(e)$ and *special* if $z = low(e)$. z is called *e-normal* if $z > low(e)$ and $z < v_0$. A daughter cycle $C(e_i)$ is called *normal* if either $alpha(S(e_i)) = LS$ is *true* or the conditions $alpha(S(e_i)) = RS$ and $low(e_i) > low(e)$ are satisfied. We will see later that the embedding of a *normal* cycle is easy and that problems occur if the cycle is *special*. In the sequel three observations are shown in order to get a feeling about the embedding problem and the involved notion.

Observation 1

Let $C(e)$ be a normal cycle and e_i an edge emanating out of the spine path of $C(e)$. If there is an e-normal attachment in $A(e_i)$, then the segment $S(e_i)$ must be embedded into the cycle $C(e)$ in all planar embeddings of the graph.

Proof: The edge $e = (v_0, v_1)$ is emanating out of the spine path of C. Without loss of generality, we assume that $C(e)$ is embedded in the inner side of the cycle C. Suppose $S(e_i)$ is embedded on the outer side of the cycle $C(e)$ (see Figure 4).

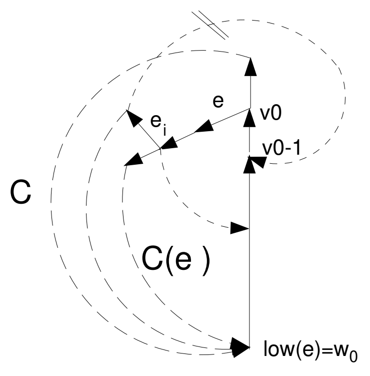


Figure 4

The starting point of the edge e_i is inside C. To reach a point between $(low(e) + 1)$ and $(v_0 - 1)$ with a path starting in e_i, either the cycle $C(e)$ must be passed or the cycle C. This is a contradiction to the planarity of the graph.

$\diamond$

The *daughters* (v_i, x_i) of a cycle $C(e)$ are all the edges e_i emanating from the spine path of $C(e)$ and all the nodes v_i are called *sprouts*. The corresponding cycles $C(e_i)$ and $C(e)$ are called *daughter cycle* and *mother cycle*. The edge e_i is called *left* (*right*) if it emanates to the left (right) side of the mother cycle (left/right with respect to the view from the starting point to the endpoint of the spine path). This fact is also denoted by the expression $side(e_i) =$ *left* or *right*, respectively. It is obvious that Observation 2 holds.

Observation 2:

Let $C(e)$ be a mother cycle and $side(e) = left$ (right), then for all daughters e_i we have: If $alpha(S(e_i)) = LS$ then the $side(e_i) = left$ (right) again, but if $alpha(e_i) = RS$ then e_i has to be embedded to the other side, i. e. $side(e_i) = right$ (left).

An *exceptional cycle* $C(e)$ is a cycle for which $alpha(e) = RS$ and $low(e) = low(e_0)$, where $C(e_0)$ is the mother cycle of $C(e)$. This means the segment $S(e)$ with $alpha(S(e)) = RS$ has a special attachment.

Consider the exceptional cycle $C(e)$ in Figure 5. Let $e = (w_r, w_{r+1}) = (v_0, v_1)$ and $\tilde{e} = (w_k, w_0)$ the only back edge in $C(e)$.

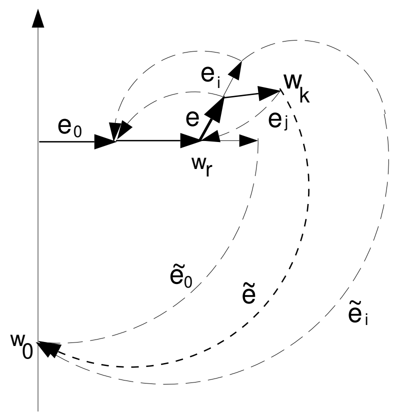


Figure 5

Since $alpha(S(e)) = RS$, we must have $low2(v_1) < v_0$. Every daughter edge e_i of $C(e)$ with $alpha(S(e_i)) = LS$ has to be embedded on the outside of the cycle $C(e)$, because only edges e_j with $low(e_j) \in \{w_0\} \cup \{w_r, \ldots, w_k\}$ can be embedded into $C(e)$. In this case Observation 1 does not hold any more.

Observation 3:

Let $C(e)$ be an exceptional cycle and e_i an edge emanating out of the spine path of $C(e)$. If $alpha(S(e_i)) = RS$, then $S(e_i)$ must be embedded into the cycle $C(e)$ and if $alpha(S(e_i)) = LS$, then $S(e_i)$ must be embedded on the outside of $C(e)$.

If $C(e)$ is an exceptional cycle, we will have additional problems with embedding it into its mother cycle. A cycle is called *orientated left* if it is directed against the clock, otherwise it is called *orientated right*.

3. The embedding algorithm

3.1 The idea

In a first phase we will run the algorithm *stronglyplanar*. The output is, in the case that the graph is planar, an entry $alpha(S(e_i))$ for every segment $S(e_i)$, which determines on which side of its mother cycle the segment has to be embedded (see Observation 2). Then we start the second phase, in which a planar map has to be determined. Therefore it is necessary to find an algorithm which embeds the daughter segments $S(e_i)$ inside their mother cycle $C(e)$. This problem can be reduced to the problem of embedding the daughter cycles $C(e_i)$ inside the cycle $C(e)$. The correct embedding of the segments $S(e_i)$ is guaranteed by recursion.

During the embedding of these cycles we construct a clockwise ordered neighbor list, consisting of a list '*ancestor*', in which the unique tree ancestor of every node is recorded, a list '*successor*', in which an entry of the first discovered successor of every node is entered and a list of the remaining neighbors for every node. It is trivial to fill up the lists '*ancestor*' and '*successor*'. The hard task is to construct the clockwise ordered lists of the remaining neighbors. Obviously, in every iteration an entry in one of those remaining neighbor lists is necessary only for the initial edge and the back edge of the actual cycle.

The order of embedding the cycles is very important. It is exactly the same order as in the procedure *stronglyplanar*. We assume again that the adjacency list of every node v is sorted as in (2.1). This ordering leads to the following rules for embedding the cycles.

Rules for embedding the cycles

Let $C(e)$ be the cycle $w_0 \rightarrow_T w_1 \rightarrow_T \cdots w_k \rightarrow_B w_0$ and $e = (w_r, w_{r+1})$. Let $e_p = (w_p, x_p)$ and $e_q = (w_q, x_q)$ with $p, q \in \{r+1, \cdots, k\}$.

The daughter cycle $C(e_p)$ is embedded before the daughter cycle $C(e_q)$ if $w_p > w_q$.
If $w_p = w_q$, then for $i, j \in \{p, q\}$ $C(e_i)$ is embedded before $C(e_j)$ if $low1(x_i) < low1(x_j)$. If these two values are equal, then e_i will be the first if $low2(x_i) \geq w_i$. Moreover, we have $low1(x_p) \geq low1(w_{r+1})$ and, because of the biconnectedness of G, the inequality $low1(x_p) < w_p$ holds.

We can make the following observations in the case $w_p = w_q$ and $low1(x_i) = low1(x_j)$. If both values $low2(x_p)$ and $low2(x_q)$ are greater than or equal to w_p, then it does not matter which cycle will be embedded first. If both values are smaller than $w_p = w_q$, then they interlace. The planarity test took both segments on different sides of the mother cycle. So, it does again not matter which segment will be embedded first. Thus the ordering of the edges, which is not uniquely determined, will not lead to any intersections.

In the sequel the general ideas are given for embedding the daughter cycles in its mother cycle. Consider the edge $e_i = (w_i, x_i)$.

(A) The embedding of the first edge $\mathbf{e_i}$ of the spine path of $\mathbf{C(e_i)}$

Claim: It is correct to embed e_i directly next to the edge (w_{i-1}, w_i), where w_{i-1} is the tree ancestor w_{i-1} of node w_i in the cycle $C(e)$ (see Figure 6a). This corresponds to the insertion of node x_i directly next to node w_{i-1} into the neighborlist of node w_i. Recall that each node has exactly one tree ancestor, therefore it is sufficient to have one array for all nodes to store their T-ancestors.

(B) The embedding of the back edge $\mathbf{\tilde{e}_i = (x_k, z)}$ of $\mathbf{C(e_i)}$, if $\mathbf{C(e_i)}$ is not an exceptional cycle

Claim: The correct way to embed $\tilde{e}_i$ is next to the edge (w_l, w_{l+1}), where $w_l = z$ and w_{l+1} denotes the successor of z in the cycle $C(e)$ (see Figure 6b, the case where z is special). This corresponds to the insertion of node x_k directly next to node w_{l+1} in the neighborlist of z. Note that those successors are not necessarily identical with the ones in the list *successor*. (In Figure 6b edge $\tilde{e}_j$ has to be embedded directly next to edge (w_r, w_{r+1}), but the entry in the list *successor* is y). Therefore, we have to update this particular successor in every iteration step.

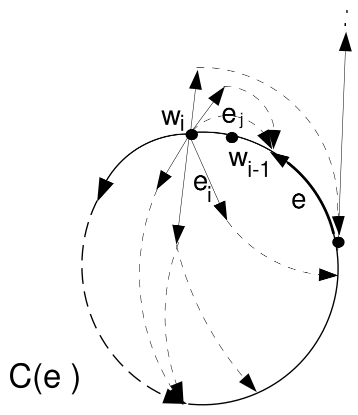


Figure 6a

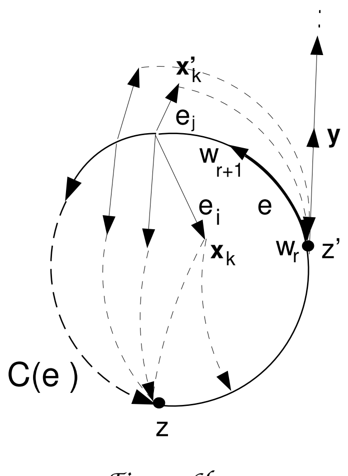


Figure 6b

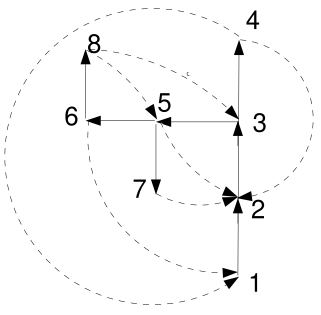
(C) The embedding of the back edge, if $\mathbf{C(e_i)}$ is an exceptional cycle

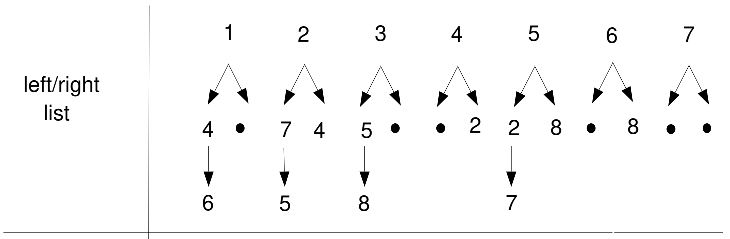
Here only the node $w_0 := low1(x_i) = low1(w_{r+1})$ is of interest. The idea is the following: Given a mother cycle and any special daughter cycle, then there are exactly two possibilities to embed the latter one correctly, depending on the relative position of $C(e_i)$ to $C(e)$ (inside or outside $C(e)$). This corresponds to the fact that there are exactly two possible positions in the neighborlist of w_0, where x_k may be inserted. These two positions must be updated before the embedding of the next cycle.

More detailed: before starting the embedding of the daughters of $C(e)$, the only back edge in the cycle $C(e)$ is already added to the adjacency list of w_0. After the corresponding insertion in the neighborlist of w_0, the positions next to the entry of edge $\tilde{e}$ are marked. So, the first daughter cycle $C(e_i)$ with minimum attachment w_0 will be embedded directly next to this edge, say inside $C(e)$. After the corresponding insertion into the neighborlist, it is necessary to update the marked position inside $C(e)$ in order to include the back edge (w'_k, w_0) of the next daughter cycle of $C(e)$ correctly. Hence, it is necessary to find an algorithm, which updates the correct positions after the embedding of a back edge (see Figure 5).

3.2 The data structure

To realize the above ideas, it is necessary to construct the above described data structure, the *neighbor record*. For every node we have a record field, *nghbrec[v]*, which contains all necessary data of the already embedded neighbors of v, among them the most important two lists *predlist* and *succlist* which will be constructed during the run of the algorithm. $Grandcycle(v)$ denotes the first cycle containing v. All the neighbors of v which emanate from or into the left side of the $grandcycle(v)$ are stored in the list *nghbrec[v].predlist*. The neighbors of v emanating from or into the right side of the *grandcycle(v)* are stored in *nghbrec[v].succlist*. Each of the neighborlists *nghbrec[v].predlist* and *nghbrec[v].succlist* has two pointers *frmark* and *bhmark* to mark those edges, after which the next neighbor of v can be included. The ancestor of every node is stored in the record field *nghbrec[v].ancestor*. Recall that every node has exactly one tree ancestor except the first one (node 1). Its tree ancestor is determined to 0, which corresponds to the supposition that the first cycle C is a loop at node 1 and the edge $(1, 2)$ is the only daughter edge of C. The successor of v in *grandcycle(v)* is stored in the record field *nghbrec[v].successor*. In the following we will shorten the notation informally to $predlist(v)$, $succlist(v)$, $ancestor(v)$ and $successor(v)$.





	1	2	3	4	5	6	7
ancestor	0	1	2	3	3	5	5
successor	2	3	4	1	6	1	2

Example 1. The neighbor record of the plane graph

In the above example the described data structure becomes clear.

Whenever a new daughter cycle will be considered, an entry is done in the list *sname* of the *sprout*, i. e. the head of the initial edge. This entry contains the information to which side of its mother cycle the path will be embedded, that is either on the left or on the right side, which is important when (a few iterations later) a cycle with its back edge ending at the sprout will be embedded. In this case we need the above information in order to decide into which neighborlist (*predlist* or *succlist*) the entry has to be inserted. Because of the partition of the neighborlist of v in *predlist*(v) and *succlist*(v) we do not have to construct a clockwise ordering anymore. The new task is to construct the neighborlists *predlist*(v) and *succlist*(v) in which the remaining neighbors occur.

As soon as all cycles are embedded, the new ordered adjacency list can be constructed easily. Just concatenate the lists in the following way:
$ancestor(v) \cup predlist(v) \cup successor(v) \cup (succlist(v))^{-1}$.

The following procedures are crucial for the embedding algorithm. They include an edge e at the correct position. The variable *vlist* carries information in which part of the list the edge has to be included, i. e. either *predlist* or *succlist*. Recall that there is no difference between including the edge $e = (w, v)$ and inserting the node w into the list of v.

Procedure putinfront(e,vlist)

This procedure includes the edge e at the beginning of the neighborlist given by *vlist*. The pointers *frmark* and *bhmark* are not changed.

Procedure append(ẽ,zlist)

The edge $\tilde{e}$ will be included directly behind the pointer *bhmark* in the list *zlist*. Afterwards, *bhmark* points to the new edge $\tilde{e}$ in the list given by *zlist*.

Lemma 3

In the algorithm the procedure append includes the edge e to the end of the list given by zlist.

Proof: We have to show that the pointer *bhmark* points to the last element of the list given by *zlist*, whenever the procedure *append* is called in the algorithm. This is shown in section 3.4.

$\diamond$

In the following two procedures, let $\tilde{e} = (w_j, z)$ be the only back edge of the daughter cycle $C(e)$, which has to be embedded. The variable *direction* $\in \{$*before*, *behind*$\}$ tells us to which of the two marked positions, *frmark* and *bhmark*, the edge $\tilde{e}$ has to be included. The correct part of the neighborlist, *predlist*(z) or *succlist*(z), is given by the variable *zlist*.

Procedure rangout(ẽ, direction, zlist);

```
(1) If ($direction$ = $before$) then
(2)        Append edge $\tilde{e}$ behind the field marked with $frmark$ in the neighbor-
           list of z given by $zlist$);
(3)        Do not change $frmark$;
(4) If ($direction$ = $behind$) then
(5)        Append the edge $\tilde{e}$ behind the field marked with $bhmark$ in the list
           given by $zlist$;
(6)        Let $bhmark$ point to the actual included field, edge $\tilde{e}$ is marked.
```

Procedure rangin(ẽ, direction, zlist);

This is the same as Procedure *rangout* with the only changes in lines (3) and (6):

```
(3')       Set $frmark$ to the new field;
(6')       Do not change $bhmark$.
```

These two procedures have different effects. In procedure *rangout* the cycles are embedded to the outer side corresponding to the edge $\tilde{e}$, whereas in procedure *rangin* the daughter cycles are embedded to the inner side corresponding to the edge $e' = (z, w_{l+1})$. In this case the edge e' stays marked the whole time. Now, it is easy to describe the algorithm.

3.3 The Embedding Algorithm

Let $e = (v_0, v_1)$ be an edge emanating from C. Embed the cycle $C(e)$ correctly into his mother cycle. Then embed recursively all daughter cycles $C(e_i)$, $i = 1, \cdots, m$, of $C(e)$.

(P) Procedure planarembedding(e, side)

```
(0) Walk along the spine of $C(e)$ while determining the ancestors and successors of the
    met nodes.
(1) Determine the actual value of $side$ (Observation 2) and update $sname(v_0)$ to this value
    (that is either $left$ or $right$);
(2) Include node $v_1$ into the neighborlist of $v_0$ (embedding of the initial edge of the segment
    $S(e)$) and then the node $w_k$ into the list of the endpoint $w_0$ of the segment $S(e)$ (by
    (PI) and (PE));
(3) Mark node $v_1$ in the neighborlist of $v_0$;
(4) Mark node $w_k$ ($\tilde{e} = (w_k, w_0)$) in the neighborlist of $w_0$. Now, the segments of the
    daughters can be included directly before and behind $w_k$.
(5) For all daughter edges $e_i$ do
        Call the procedure $planarembedding(e_i, side)$;
(6) Mark in the neighborlist of $w_0$ the correct two positions, where the next edges (sisters
    of $e$) can be included (by (PM)).
```

(PI) Embedding in the starting point of edge $\mathbf{e} = (\mathbf{v}, \mathbf{w})$

```
(1) If ($side = left$) then
(2)         $vlist := predlist$
(3) Else $vlist := succlist$;
(4) Call procedure $putinfront(e, vlist)$;
```

(PE) Embedding in the endpoint of the edge $\tilde{\mathbf{e}} = (\mathbf{w}, \mathbf{z})$

```
(1) If $z$ is normal then
(2)         If $z$ has not been a sprout so far, then
(3)                 Call the procedure $append(\tilde{e}, zlist)$;
(4)         Else { $sname(z) \in \{left, right\}$ }    $rangin(\tilde{e}, direction, zlist)$
(5) Else { $z$ is special }    $rangout(\tilde{e}, direction, zlist)$.
```

(PM) Determination of the pointers of the edge e

```
(1) Mark the correct two positions in the neighborlist of $w_0$ ($\tilde{e} = (w_k, w_0)$).
```

The variable *zlist* depends on various parameters. The detailed rules how to set *zlist*, *direction* and *side* as well as a description of the procedure (PM) is given in the proof of the correctness of the algorithm (see section 3.4). In the following example we give a detailed run of the embedding algorithm.

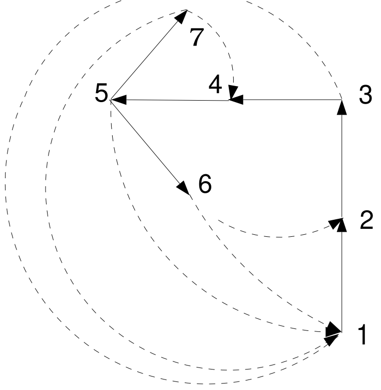
Example

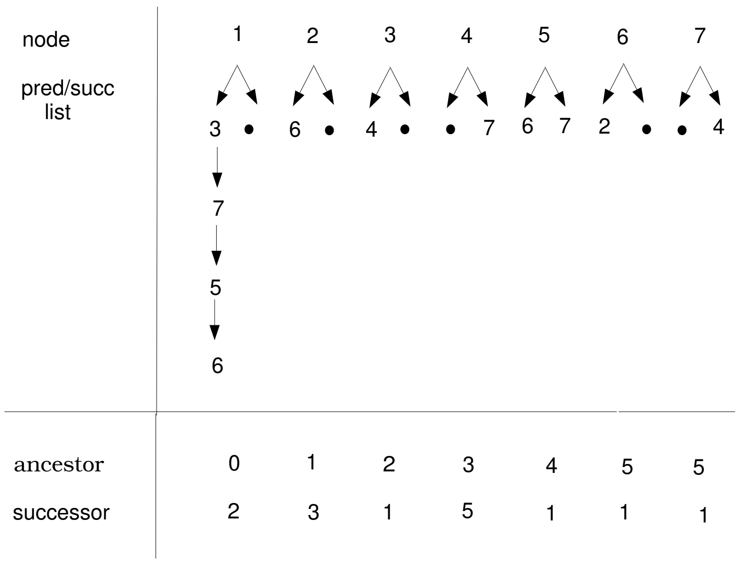
Let us consider the graph given by the following adjacency list, which is already sorted by (2.1).

node	1	2	3	4	5	6	7
neighbors	2	3	1 4	5 1	1 7 6	1 2	1 4

The algorithm *stronglyplanar* gives us the information that the graph is planar and the values of the function *alpha*. The first cycle C is $(1 \to 2 \to 3 \to 1)$ with its only daughter edge $e = (3, 4)$. The cycle $C(e)$ is $(3 \to 4 \to 5 \to 1 \to 2 \to 3)$. The daughter edges of that one are $e_1 = (5, 7)$ and $e_2 = (5, 6)$. Procedure *stronglyplanar*$((3,4), left)$ gives us $alpha(S(e_1)) = RS$ and $alpha(S(e_2)) = LS$ (because the two segments $S(e_1)$ and $S(e_2)$ interlace and the attachments of $A(e_2) \in \{w_0 + 1, w_r - 1\} = \{2\}$, see (2.2)). Observation 1 tells us to embed $S(e_2)$ into the cycle C, therefore $S(e_1)$ has to be embedded outside C. Having this information we can draw the graph by hand, it has to look like Figure 7. Let us see if our algorithm *planarembedding* gives the correct embedding. We start with the edge $(1, 2)$ and suppose that $(1, 2)$ is on the left side of its mother cycle (the loop at node 1). The position of the pointers *frmark* and *bhmark* are signified by the underlines or overlines respectively.

```
$planarembedding((1,2), left)$;        Cycle: $(1 \to 2 \to 3 \to 1)$;
  Enter $ancestor(i)$ and $successor(i)$ for $i = 1, 2$ and 3;
  (This step is trivial, therefore we will omit this step in the sequel);
  Add edge $(3,1)$ at its endpoint by $append$: $predlist(1)$: $\underline{*} \to \overline{\mathbf{3}}$;
  Mark edge $(3,1)$: $predlist(1)$: $\underline{*} \to \overline{\mathbf{3}}$;

  $planarembedding((3,4), left)$;        Cycle: $(3 \to 4 \to 5 \to 1 \to 2 \to 3)$;
    Add 4 to the starting point: $predlist(3)$: $\overline{\underline{*}} \to 4$;
    Add 5 to the endpoint by $rangout(behind)$: $predlist(1)$: $\underline{*} \to 3 \to \overline{\mathbf{5}}$;
    Mark node 4: $predlist(3)$: $\underline{*} \to \overline{\mathbf{4}}$;
    Mark edge $(5,1)$: $predlist(1)$: $* \to \underline{\mathbf{3}} \to \overline{\mathbf{5}}$;

    $planarembedding((5,6), left)$;        Cycle: $(5 \to 6 \to 1 \to 2 \to 3 \to 4 \to 5)$;
      Starting point (after marking): $predlist(5)$: $\underline{*} \to \overline{\mathbf{6}}$;
      Endpoint by $rangout(behind)$: $predlist(1)$: $* \to \underline{\mathbf{3}} \to 5 \to \overline{\mathbf{6}}$;
      Mark edge $(6,1)$: $predlist(1)$: $* \to 3 \to \underline{\mathbf{5}} \to \overline{\mathbf{6}}$;

      $planarembedding((6,2), left)$;        Cycle: $(6 \to 2 \to 3 \to 4 \to 5 \to 6)$;
        The change of the marks is not relevant for the embedding of edge $(4,2)$.
        Therefore it will not carried out here;
        Starting point (after marking): $predlist(6)$: $\underline{*} \to \overline{\mathbf{2}}$;
        Endpoint by $append$: $predlist(2)$: $\underline{*} \to \overline{\mathbf{6}}$;
      $End\ of\ planarembedding((6,2), left)$;

      Change marks: $predlist(1)$: $* \to \underline{\mathbf{3}} \to 5 \to \overline{\mathbf{6}}$;
    $End\ of\ planarembedding((5,6), left)$;

    $planarembedding((5,7), right)$;        Cycle: $(5 \to 7 \to 1 \to 2 \to 3 \to 4 \to 5)$;
      Starting point (after marking): $succlist(5)$ : $\underline{*} \to \overline{\mathbf{7}}$;
      Endpoint by $rangout(before)$: $predlist(1)$: $* \to \underline{\mathbf{3}} \to 7 \to 5 \to \overline{\mathbf{6}}$;
      Mark edge $(7,1)$: $predlist(1)$: $* \to \underline{\mathbf{3}} \to \overline{\mathbf{7}} \to 5 \to 6$;

      $planarembedding((7,4), right)$;        Cycle: $(7 \to 4 \to 5 \to 7)$;
        Here the changing of the marks are not relevant and compensate each other;
        Starting point (after marking): $succlist(7)$: $\underline{*} \to \overline{\mathbf{4}}$;
        Endpoint by $append$: $succlist(4)$: $\underline{*} \to \overline{\mathbf{7}}$;
      $End\ of\ planarembedding((7,4), right)$;

      Change marks: $predlist(1)$: $* \to \underline{\mathbf{3}} \to 7 \to 5 \to \overline{\mathbf{6}}$;
    $End\ of\ planarembedding((5,7), right)$;

    All daughter cycles of $C((3,4))$ are embedded correctly at this point;
    Change marks: $predlist(1)$: $\underline{*} \to 3 \to 7 \to 5 \to \overline{\mathbf{6}}$;
  $End\ of\ planarembedding((3,4), left)$;

$End\ of\ planarembedding((1,2), left)$;
```





node	1	2	3	4	5	6	7
pred/succ list	3 •	6 •	4 •	• 7	6 7	2 •	• 4
	7						
	5						
	6						
ancestor	0	1	2	3	4	5	5
successor	2	3	1	5	1	1	1

Figure 7. The plane graph. The neighbor record constructed by 'planarembedding'

The algorithm *planarembedding* constructed the neighborlist in Figure 7. In fact, after concatenating the lists we get a correct planar embedding of the graph.

node	1	2	3	4	5	6	7
neighbors	3 7 5 6 2	1 6 3	2 4 1	3 5 7	4 6 1 7	5 2 1	5 1 4

3.4 The correctness of the algorithm 'planarembedding'

We prove the correctness of the algorithm in two steps. The lemma below implies the correct embedding of the first daughter $C(e_0)$ in its mother cycle $C(e)$.

Lemma 4

(1) After the embedding of a complete daughter segment $S(e_0)$ into its mother cycle (after line (6) of planarembedding(e)) the correct two positions in every list nghbrec(z) with $z \in A(e)$ are marked.

(2) If after the embedding of the already handled sister cycles $C(e_j)$ of $C(e_0)$ the correct two positions are marked in every list nghbrec(z) with $z \in \bigcup_j A(e_j)$, then $C(e_0)$ will be included correctly. After that the correct two positions in the neighborlist of low(e_0) are marked.

Lemma 5

The correctness of the algorithm planarembedding is shown if (1) and (2) of Lemma 4 are proved.

Proof of Lemma 5: Start with the grandcycle $C((1,2))$. By the recursiveness of the algorithm the first daughter cycle of C will be embedded, then the first daughter cycle of that one, and so on. Let $C(e_1)$ be the first of these cycles without daughter. Because of *(2)* it will be embedded correctly into $C(e)$, its mother cycle. By *(2)* in line (6) the correct positions are marked and all sisters of $C(e_1)$ will be included correctly (using *(1)* and *(2)*).

$\diamond$

Point *(1)* of Lemma 4 depends on what to do in line (6). This will be described in detail in the second part of this section. In the first part point *(2)* of Lemma 4 is shown.

For the following proofs it is helpful to distinguish between the following four cases.

A: Father cycle $C(e)$ is *orientated left*, e is *left*
B: Father cycle $C(e)$ is *orientated right*, e is *right*
C: Father cycle $C(e)$ is *orientated left*, e is *right*
D: Father cycle $C(e)$ is *orientated right*, e is *left*

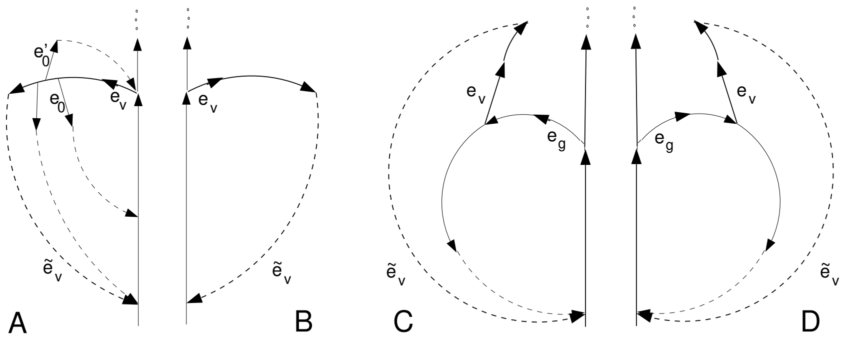


Figure 8.1. Figures of the four possible cases

Proof of (2) of Lemma 4:

The correctness of the embedding in the starting point

The embedding of the initial edge is independent of any marks and very easy to do. Let $C(e)$ be the mother cycle and $e = (w_r, w_{r+1})$. Claim 1 is obvious.

Claim 1: *The edge e_0 will be included in predlist(v_0) iff e_0 has the attribute left. If e_0 has the attribute right, then it will be included into the list succlist(v_0).*

Claim 2: *It is correct to include the edge $e_0 = (v_0, v_1)$ to the beginning of the neighborlist of v_0.*

Proof of Claim 2: If there is an already embedded edge e' in the list of v_0, then it must have a smaller or equal *low*-value. If the value is the same, then it is correct to include the new edge before edge e' into the neighborlist of v_0. If the edge e' has a smaller *low*-value, then it must be included behind the new edge e_0 in the neighborlist (see Figure 6a).

$\diamond$

The correctness of the embedding at the endpoint

Before this can be shown, we have to determine the variables *zlist* and *direction* needed in (PE). Let $\tilde{e}_0 = (w, z)$ be the only back edge in the cycle $C(e_0)$. The aim is to embed this edge. There are several cases:

```
CASE I     If $\tilde{e}_0$ is normal and $z$ is not a sprout, then
               If $side(e_0) = left$ then    $zlist(\tilde{e}_0) := predlist(z)$
               Else $\{side(e_0) = right\}$   $zlist(\tilde{e}_0) := succlist(z)$;

CASE II    If $\tilde{e}_0$ is normal and $z$ is a sprout, then
               If $sname(z) = left$ then    $zlist(\tilde{e}_0) := predlist(z)$
                   If $side(e_0) = left$ then  $direction := before$
                   Else $\{side(e_0) = right\}$ $direction := behind$;
               Else $\{sname(z) = right\}$   $zlist(\tilde{e}_0) := succlist(z)$
                   If $side(e_0) = left$ then  $direction := behind$
                   Else $\{side(e_0) = right\}$ $direction := before$;

CASE III   If $\tilde{e}_0$ is special $\{low(e_0) = low(e_v)\}$ and $z$ is not a sprout, then
               $zlist(\tilde{e}_0) := zlist(\tilde{e}_v)$;
               If $zlist(\tilde{e}_0) = predlist(z)$ then
                   If $side(e_0) = left$ then  $direction := behind$
                   Else $\{side(e_0) = right\}$ $direction := before$;
               Else $zlist(\tilde{e}_0) = succlist(z)$, then
                   If $side(e_0) = left$ then  $direction := before$
                   Else $\{side(e_0) = right\}$ $direction := behind$;

CASE IV    If $\tilde{e}_0$ is special and $z$ is a sprout, then
               If $sname(z) = left$ then    $zlist(\tilde{e}_0) := predlist(z)$
                   If $side(e_0) = left$ then  $direction := behind$
                   Else $\{side(e_0) = right\}$ $direction := before$;
               Else $\{sname(z) = right\}$    $zlist(\tilde{e}_0) := succlist(z)$
                   If $side(e_0) = left$ then  $direction := before$
                   Else $\{side(e_0) = right\}$ $direction := behind$;
```

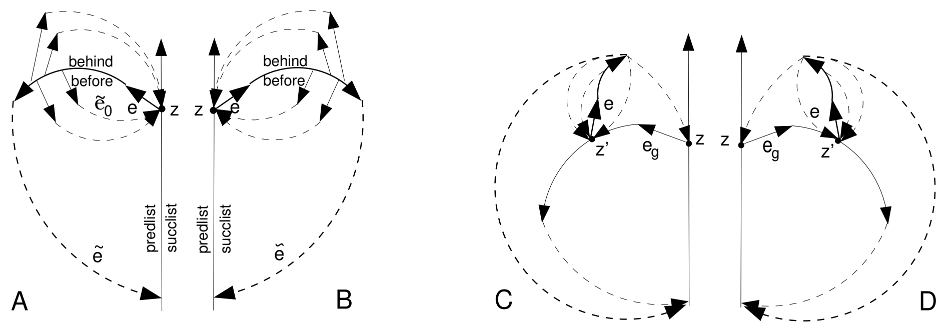


Figure 8.2. CASE II: $\tilde{e}_0$ is normal and z is a sprout

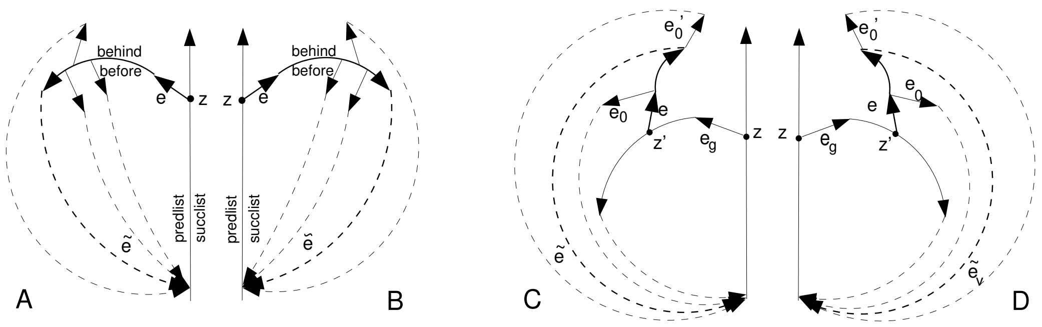


Figure 8.3. CASE III: $\tilde{e}_0$is special and z is not a sprout

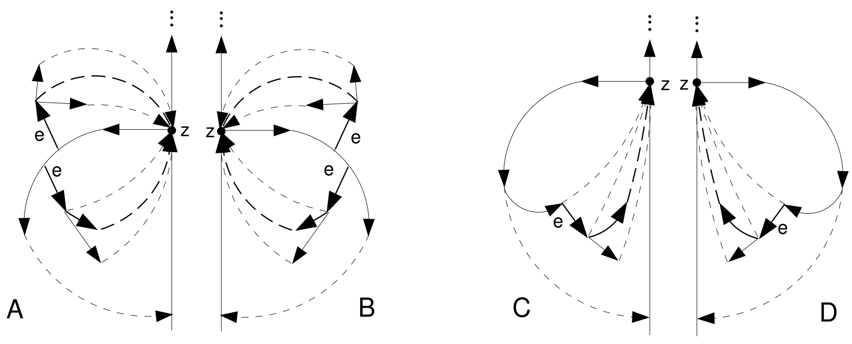


Figure 8.4. CASE IV: $\tilde{e}_0$is special and z is a sprout

Let us show the correctness of the first case.

CASE I: $\tilde{e}_0$ is normal and z has not been a sprout so far

Claim 1: *It is correct to include the edge $\tilde{e}_0$ in predlist(z) if e_0 is left. Otherwise $\tilde{e}_0$ has to appear in succlist(z).*

Proof: The spine path has to be embedded together with $\tilde{e}_0$ to the same side of the mother cycle. Claim 1 follows from the observation that each side of a cycle is related to exactly one list (see Figure 8.1).

◇

Claim 2: *The edge $\tilde{e}_0$ has to be appended to the end of the neighborlist of z. This will be done by procedure append.*

Proof: First consider the case that e_0 is the first daughter of its mother cycle. Then Claim 2 is shown if Claim 3 is proven.

> **Claim 3:** *If $C(e_0)$ is the first daughter of its mother cycle, then the list zlist($\tilde{e}_0$) in which $\tilde{e}_0$ should be included is empty.*
>
> Proof: The initial edge (z, w_j) of a previously embedded cycle cannot be included, because $v_0 > z$ (compare rules (2.1) and Figure 9.1, e_1). For the same reason, no endpoint of such a path can be embedded inside the cycle (see Figure 9.1, e_2). Outside the cycle this is not possible, because no such edges can exist in a depth-first-search tree (see Figure 9.1, e_3).

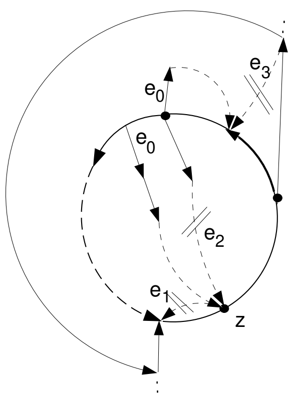


Figure 9.1

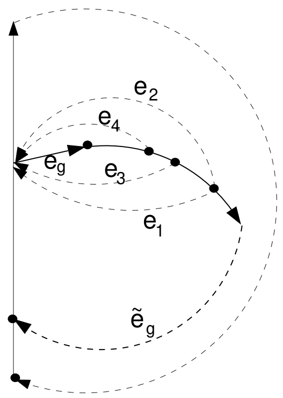


Figure 9.2

Proof of Claim 2: Assume now that the sister segments $S(e_j), j = 1, \cdots, l$ have already been embedded. With the assumption of *(2)* the correct two positions in the neighborlist of $low(e_l)$ and in every list of z with $z \in A(e_j)$ are marked. We have to show that $\tilde{e}_0$ will be included correctly and the two positions in $low(e_0)$ will be marked.

In the case that $low(e_l) = low(e_0) = z$, the last element of the list must be marked. With the procedure *append* $\tilde{e}_0$ will be included to the end of the list and the new element will be marked. If $low(e_l) < z$, then either the list of z is still empty or previous embedded sister segments contained the node z. With the assumption of *(2)* *bhmark*(z) must point to the last element of the list.

◇

CASE II: $\tilde{e}_0$ is normal and z is a sprout

The speciality in this case is that the edge $e_g = (v_0, v_1)$ was emanating from a cycle. The edge e_g has been included to the list of v_0 before. The array *sname* contains also information about the attribute of the edge e_g, that is if it is emanating on the left or on the right side of its mother cycle. In general $C(e_g)$ is not the mother cycle of $C(e_0)$ (see Figure 8.2, z). All the edges, which emanate from $C(e_0)$ and end in v_0, have to be included into the same list *predlist*(z) or *succlist*(z) of v_0. On the other hand it is not correct to include them at the beginning or at the end of the list. The edge e_g is central in the neighborlist of v_0. All the back edges ending in v_0 have to be included directly next to this edge. It is obvious that the following claims hold when considering the four cases of Figure 8.2.

Claim 1: *The procedure rangin works correctly. It includes new edges directly next to the central edge* e_g.

Claim 2: *It is correct to include the edge* $\tilde{e}_0$ *in list predlist*(z) *if and only if sname*(v_0) = *left and to include the edge* $\tilde{e}_0$ *in list succlist*(z) *if and only if sname*(v_0) = *right* .

Claim 3: *In the case that* $\tilde{e}_0$ *is included into predlist*(z), *then if* e_0 *is left the variable direction has to be before, otherwise it is behind. On the other hand: If* $\tilde{e}_0$ *is included in succlist*(z), *then if* e_0 *is left the variable direction has the value behind, otherwise it is before.*

In the same way we can show cases III and IV.

Proof of (1) of Lemma 4:

The correct determination of the pointers 'frmark' and 'bhmark'

The correctness of the algorithm depends on the correct position of the pointers *frmark* and *bhmark*. They must point to the correct positions in every list. As soon as $C(e_0)$ turns into a mother cycle, the edge $\tilde{e}_0$ must be marked in the neighborlist. After the embedding of segment $S(e_0)$ the control is given back to its mother cycle. At this point some changes of the pointers *frmark* and *bhmark* are necessary. These changes will be determined in the sequel. In the described algorithm line (6) of (P) can be substituted by the following procedure.

(PM) Determination of the pointers of the edge $\tilde{e} = (w_k, w_0)$

```
(1) If in line (2) procedure rangin was not called, then
(2)     If in line (2) the variable direction had the value behind, then
(3)         $frmark(w_0) := oldfr$;
(4)     Else $bhmark(w_0) := oldbh$;
```

```
(5) Else { $rangin$ }
(6)     If in line (2) the variable $direction$ had the value $behind$, then
(7)         $bhmark(w_0) := frmark(w_0)$; $frmark(w_0) := oldfr$;
(8)     Else $frmark(w_0) := bhmark(w_0)$; $bhmark(w_0) := bhmark(w_0).next$;
```

The variables *oldfr* and *oldbh* denote the values of the pointers after including the edge $\tilde{e}_0$ into the list ((2) of (P)), but before changing them (in (4) of (P)).

Observation 4

Let $\tilde{e}_0$ be a daughter of the cycle $C(e)$. If $C(e_0)$ is without a daughter, then the effect of setting the pointers in line (4) of (P) is neutralized by the setting of pointers in line (7).

It remains to prove the following:

Claim 1: *Consider the algorithm during the embedding of the cycle $C(e_v)$. After the complete embedding of its daughter cycles (in (6) of (P)), the correct two positions in all lists of w with $w \in A(e_0)$ are marked.*

In the progress of the algorithm from the mothers to the daughters the correct two positions are marked. After returning from the daughter cycle $C(e_0)$ back to its mother cycle $C(e)$ some of the marked positions may have become wrong in the meantime. But this can only occur in the neighborlists of $low(e_0)$. In all the other lists the correct two positions are still marked. Hence, we can substitute the above claim by the following.

Claim 1': *After returning from the cycle $C(e_0)$ to its mother cycle $C(e)$ the correct two positions in the neighborlist of $low(e_0) = w_0$ are marked.*

Proof:

CASE I: The edge $\tilde{e}_0$ is normal and w_0 is not a sprout

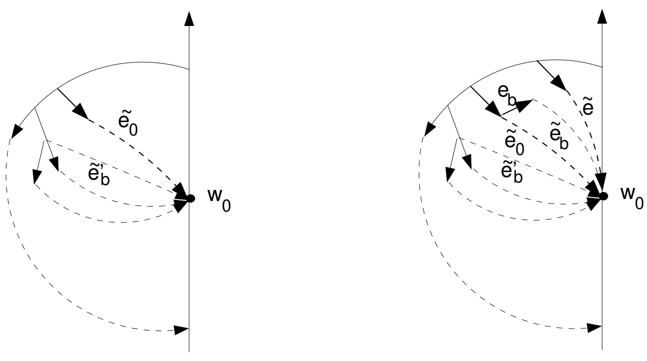


Figure 10.1. Embedding of edge e_0 Changing of the pointers

Consider node $w_0 = low(e_0)$. All back edges of $S(e_0)$ with endpoints in w_0 have been included by *rangout*. After including the last daughter, *frmark* points to the edge $\tilde{e}'_b$ and *bhmark* to the last edge $\tilde{e}_b$ included by *rangout(behind)* (see Figure 10.1).

In (7) of (PM) the pointer *bhmark* will not be changed. The only possible way to include the next daughter $C(e_j)$ of $C(e)$ is by procedure *append*. Hence, the pointer *frmark* is of no more interest in this case.

CASE II: The edge $\tilde{e}_0$ is normal and w_0 is a sprout

Consider node w_0 after the embedding of the daughter segments $S(e_i)$ of $C(e_0)$. If the attribute of both edges e_0 and e_g is the same, then the previous edges included into this list must have been included by *rangout* or *rangin*. The pointer *bhmark* points to the last edge $\tilde{e}_b$ included by *rangout(behind)* or *rangout(before)*. Because $\tilde{e}_0$ was included by *rangin(before)*, *frmark* is set to $\tilde{e}_b$ and *bhmark* to e_g in line (8) of the procedure. Hence, in the list of w_0 the correct positions are marked (see Figure 10.2). If the attribute of both edges e_0 and e_g is different, then *frmark* points to the edge e_g. The edge e_0 was included by *rangin(behind)*. Thus, *bhmark* is set to e_g and *frmark* is reset to the value of the pointer *frmark* before the change in line (4). Hence, the marks are correct.

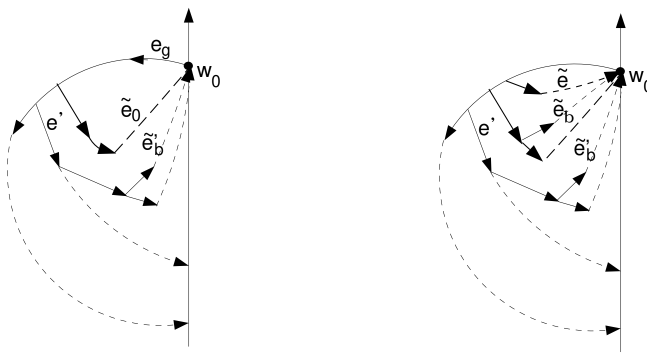


Figure 10.2. Embedding of edge $\tilde{e}_0$ Changing of the pointers

CASE III: The edge $\tilde{e}_0$ is special and w_0 is not a sprout

If both edges e and e_0 have the same attributes, then let $\tilde{e}_b$ be the last edge included by *rangout(behind)* in the list of w_0. The pointer *bhmark* points to that edge, *frmark* to $\tilde{e}'_b$ (see Figure 10.3). In line (3) *frmark* will be reset to the value before including e_0, which was $\tilde{e}_u$. The correct two positions for the next daughter of $C(e)$ are marked.

If both edges e and e_0 have different attributes, then the following situation occurs. Before the resetting *frmark* points to $\tilde{e}_u$, *bhmark* to $\tilde{e}_b$, but after line (4) *bhmark* points to $\tilde{e}$. The sisters of $C(e)$ can be included correctly.

CASE IV: $\tilde{e}_0$ is special and w_0 is a sprout

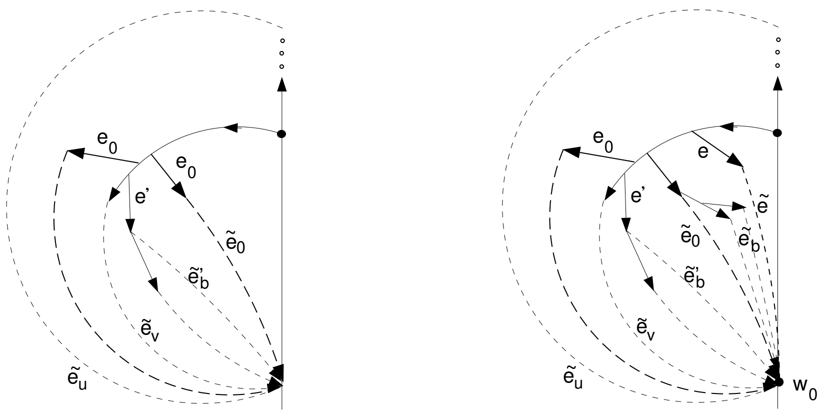


Figure 10.3. Embedding of edge $\tilde{e}_0$ Changing of the pointers

This case is similar to case III, because e_0 has to be included by *rangout* and the pointers *frmark* and *bhmark* point to the same positions.

◇

So, the correctness of the algorithm is proven. Let us recall the whole algorithm. In a first phase, the planarity testing algorithm *stronglyplanar* is run. The output is an entry $alpha(S(e))$ for every segment determining the side of its mother cycle on which it will be embedded. In the second phase, we embed cycle by cycle in a recursive way. In one iteration step we take a cycle and embed it. That is, we walk along the cycle while noting the ancestors and successors, and inserting the initial edge and the back edge of the cycle to the correct position in the remaining neighborlist. These steps and the additional updates need constant time. We consider the edges (cycles) in the same order as it is done in the linear planarity testing algorithm. Hence, it is obvious that the algorihm runs in linear time.

Theorem

Procedure planarembedding$((1, 2), \mathit{left})$ *generates a correct combinatorial embedding of the graph* G *in time* $O(|E|) = O(|V|)$.

4. Computational Results

Both procedures *stronglyplanar* and *planarembedding* have been implemented in Pascal on a SUN SPARCstation 2. We ran a series of randomly generated maximal planar graphs. These graphs were generated in the following way. Start with the complete graph on three nodes. In every step a face will be selected randomly, which will be divided into three new triangles.

A graph generated this way has $|E| = 3|V| - 6$ edges and is planar. The results can be seen in the table below. So, for example, to construct the embedding of a maximal planar graph with 1000 nodes, it took us less than 1 second including the 0.47 secons for reading the input data. To embed a graph of size 5 times larger, we needed less than 5 times more running time.

#nodes	#edges	input	prepare	planarity	embedding	utime/sec.
100	294	0.05	0.00	0.02	0.02	0.08
500	1494	0.23	0.05	0.08	0.03	0.40
1000	2994	0.47	0.10	0.18	0.07	0.82
5000	14994	2.38	0.48	0.73	0.30	4.00
10000	29994	5.08	0.92	1.45	0.58	8.17
20000	59994	10.33	1.70	2.85	1.05	16.32
30000	89994	15.32	2.82	4.30	1.60	24.52
40000	119994	20.60	3.87	5.80	2.18	33.22

In Via Minimization problems arising in the VLSI-layout, the graphs have less edges, that is about 2 times the number of nodes. Here the algorithm runs much faster. The table below shows some examples.

#nodes	#edges	input	prepare	planarity	embedding	utime/sec.
827	1389	0.40	0.05	0.03	0.03	0.51
828	1693	0.45	0.05	0.08	0.03	0.62
1201	2129	0.45	0.08	0.07	0.03	0.67
1326	2393	0.52	0.08	0.13	0.03	0.77
1327	2757	0.72	0.08	0.13	0.07	1.02
1365	2539	0.57	0.08	0.10	0.05	0.82

The storage space needed for running the algorithms is also linear. These experimental results show that our method is not just of theoretical interest but also suitable for practical applications.

Acknowledgements

I would like to thank Petra Bauer for very carefully reading the first version of this paper and for making a number of suggestions that substantially improved the presentation. I would also like to thank Professor Michael Jünger for his helpful comments.

References

[BL] Booth, K.S. and G.S. Leuker, Testing for the Consecutive Ones Property, Interval Graphs, and Graph Planarity Using PQ-tree Algorithms, *J. of Comp. and Sys. Sci.*, vol. 13, pp. 335-379, 1976.

[CN] Chiba, N. and T. Nishizeki, A Linear Algorithm for Embedding Planar Graphs Using PQ-Trees, *J. of Comp. and Sys. Sci.*, vol. 30, pp. 54-76,1985

[ET] Even, S. and R.E. Tarjan, Computing an st-numbering, *Th. Comp. Sci.*, vol. 2, pp. 339-344, 1976.

[HT] Hopcroft, J. and R. Tarjan, Efficient Planarity Testing, *JACM*, vol. 21, No. 4, pp. 549-568, 1974.

[LEC] Lempel, A., S. Even and I. Cederbaum, An Algorithm for Planarity Testing of Graphs, in *Theory of Graphs, International Symposium*, pp. 215-232, Rome, 1966.

[M] Mehlhorn, K., Graph Algorithms and NP-Completeness, *Data Structures and Algorithms*, vol. 2, pp. 93-122, Springer-Verlag, 1984.

[T] Tarjan, R., An efficient planarity algorithm, *STAN-CS-244-71*, Comput. Sci. Dep., Stanford U., 1971.