On the Embedding Phase of the Hopcroft and Tarjan Planarity Testing Algorithm

Kurt Mehlhorn and Petra Mutzel *†

Abstract

We give a detailed description of the embedding phase of the Hopcroft and Tarjan planarity testing algorithm. The embedding phase runs in linear time. An implementation based on this paper can be found in [MMN93].

An undirected graph $G = (V, E)$ is called *planar* if it can be mapped into the plane without edge crossings, i.e., the vertices of G are mapped into distinct points in the plane and the edges of G are mapped into disjoint Jordan curves connecting their endpoints. Such a mapping is called a *planar geometric embedding.* Two geometric planar embeddings are called equivalent if there is an homeomorphism of the plane transforming one into the other. An equivalence class of geometric planar embeddings is called a *planar topological embedding* or simply *planar embedding.*

A planar embedding of a planar graph induces a cyclic ordering on the edges incident to any fixed vertex, namely the clockwise ordering of the edges around their common endpoint. A graph G together with a cyclic ordering on the edges incident to each vertex is called a *combinatorial embedding*, it is called a *planar combinatorial embedding* if it is induced by some planar embedding. Different planar embeddings can give rise to the same combinatorial embedding. However, a planar combinatorial embedding of a connected graph uniquely determines its topological embedding on the sphere. In the plane, it determines the topological embedding up to selection of the outer face. Recall that an embedding into the plane can be obtained from an embedding on the sphere by polar projection. The pole can be put into any face. There are linear time algorithms [dFPP91, Sch90] to convert a planar combinatorial embedding into a geometric embedding, e.g., the algorithm by Schnyder puts the vertices of an n node graph onto an $(n-2) \times (n-2)$ grid and realizes the edges by straight-line segments.

Hopcroft and Tarjan [HT74] gave an algorithm that tests the planarity of an undirected graph in linear time. Alternative linear time algorithms were developed by Lempel, Even, and Cederbaum [LEC67, ET76], Booth and Lueker [BL76], and Fraysseix and Rosenstiehl [FR82]. Chiba, Nishizeki, Abe and Ozawa [CNAO85] have shown how to extend the algorithm of Booth and Lueker so as to also construct a planar combinatorial embedding. Hopcroft and Tarjan also stated but gave no details that their planarity testing algorithm can be extended to also construct a planar combinatorial embedding. The textbook of the first author [Meh84, vol. 2, page 112] attempts to give more details (in less than one page) but the presentation is incorrect. We conclude that there is no published correct presentation of the embedding phase of the Hopcroft and Tarjan algorithm.

*Max-Planck-Institut für Informatik, Im Stadtwald, 66123 Saarbrücken, Germany
†Revised version, 09.03.1995

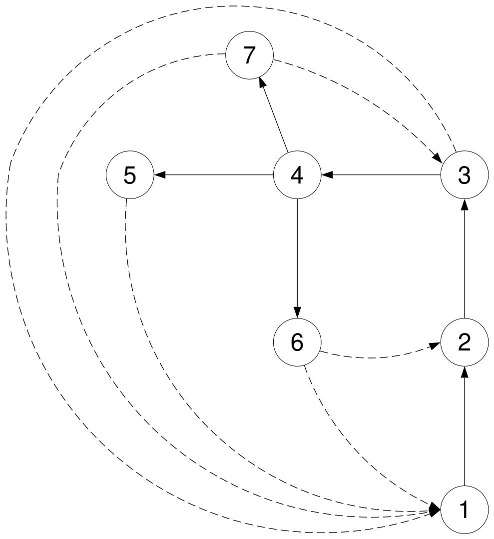

Figure 1: A DFS-tree of a planar graph. Tree edges are shown solid and back edges are shown dashed.

In this note we give a complete description of the embedding phase. An alternative presentation can be found in [Mut92]. Our embedding algorithm has the same recursive structure as the testing algorithm of Hopcroft and Tarjan and also runs in linear time. An implementation based on this note is described in [MMN93] and is distributed with the LEDA platform of combinatorial and geometric computing [Näh95, MN95] (anonymous ftp mpi-sb.mpg.de, directory pub/LEDA).

The testing phase of the Hopcroft and Tarjan algorithm is discussed in detail in [Meh84, vol. 2, pages 96 – 111]. We summarize that discussion. The graph G is assumed to be biconnected. We also fix a particular DFS-tree of G and identify the vertices of G with their DFS-numbers. We direct all tree edges from lower to higher DFS-number and all non-tree edges from higher to lower DFS-number. Non-tree edges are called back edges. Figure 1 shows an example. We use T and B to denote the set of tree edges and back edges respectively.

We associate a *segment* $S(e)$ and a *cycle* $C(e)$ with every edge $e = (x, y)$ of G. If e is a back edge then $C(e)$ and $S(e)$ consist of the tree path from y to x and the edge e. If e is a tree edge then let $V(e)$ be the set of tree successors of y (including y itself) and let w_0 be the lowest numbered endpoint of any back edge starting in $V(e)$. The cycle $C(e)$ consists of a tree path from the vertex w_0 to a vertex $w \in V(e)$ with $(w, w_0) \in B$ and the back edge (w, w_0) and the segment $S(e)$ consists of $C(e)$, the subgraph induced by $V(e)$, and all back edges starting in a node in $V(e)$. Note that the segment $S(e)$ is uniquely defined but that there may be several choices for the cycle $C(e)$. We will later fix one particular choice for $C(e)$. We divide the tree path underlying the cycle $C(e)$ into two parts, its stem and its spine. The *stem* consists of the part ending in x. The *spine* is empty if e is a back edge and it is the part starting in y if e is a tree edge.

In our example, the cycle $C((3,4))$ may consist of the tree path from 1 to 5 followed by the back edge $(5,1)$. The stem is the tree path from 1 to 3 and the spine is the tree path from 4 to 5. The cycle $C((1,2))$ may consist of the tree path from 1 to 3 and the back edge $(3,1)$.

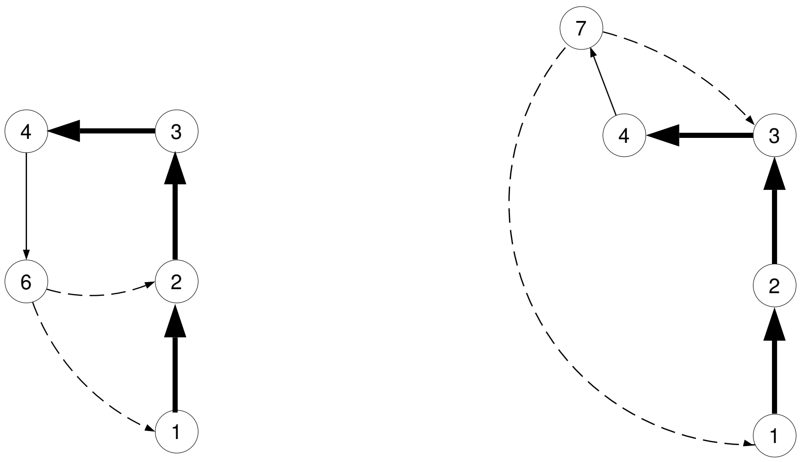

Figure 2: Embeddings of $S((4,6))$ and $S((4,7))$ induced by the embedding of G. Stems are shown in bold. The embedding of $S((4,6))$ is canonical and the embedding of $S((4,7))$ is reversed canonical.

Its stem is the node 1 and its spine is the tree path from 2 to 3. The segment $S((1,2))$ is the entire graph G and the segment $S((3,4))$ is the graph G minus the edge $(3,1)$.

A segment $S(e)$ is called *strongly planar* if there is a planar embedding of $S(e)$ and some face in that embedding such that the entire stem of the cycle $C(e)$ is contained in the border of the face. An embedding with such a property is called a strongly planar embedding of $S(e)$. If a segment $S(e)$ has a strongly planar embedding then it also has one where the stem of $C(e)$ borders the outer face. When we talk about a strongly planar embedding we assume from now on that the stem of $C(e)$ borders the outer face. Let $w_0, w_1, \ldots, w_r$ with $e = (w_r, y)$ be the stem of $C(e)$. A strongly planar embedding of $S(e)$ is called *canonical* (*reversed canonical*) if for all i, $0 < i < r$, the edge (w_i, w_{i+1}) immediately precedes (follows) the edge (w_i, w_{i-1}) in the clockwise ordering of edges incident to w_i.

Figure 2 shows the embeddings of $S((4,6))$ and $S((4,7))$ induced by the embedding of G shown in Figure 1.

Since G is assumed to be biconnected there is exactly one tree edge out of vertex 1, namely the edge $(1,2)$. Moreover $G = S((1,2))$ and G is planar iff $S((1,2))$ is strongly planar.

Let e_0 be any edge and let $C = C(e_0)$ be the cycle associated with e_0. An edge $e = (x,y)$ is said to *emanate* from C if x lies on the spine of C but e does not belong to C. We will also say that the segment $S(e)$ emanates from C. If $e_1, \ldots, e_m$ are the edges emanating from C then $S(e_0) = C + S(e_1) + \ldots + S(e_m)$, i.e., $S(e_0)$ is the union of the cycle C and the segments $S(e_1), \ldots, S(e_m)$.

We need some more concepts. As above, let $C = C(e_0)$ and let $e = (x,y)$ emanate from C. The set $A(e)$ of *attachments* of segment $S(e)$ to cycle C is defined to be the set $\{x,y\}$ if e is a back edge and the set $\{x\} \cup \{z;\ (w,z) \text{ is a back edge, } w \in V(e) \text{ and } z \notin V(e)\}$ if e is a tree edge. Two segments $S(e)$ and $S(e')$ where e and e' emanate from C are said to *interlace* if either there are nodes $x < y < z < u$ on cycle C such that $x, z \in A(e)$ and $y, u \in A(e')$ or $A(e)$ and $A(e')$ have three points in common.

The *interlacing graph* $IG(C)$ with respect to cycle $C = C(e_0)$ is defined as follows. The nodes of $IG(C)$ are the segments $S(e)$ where e emanates from C. Also, $S(e)$ and $S(e')$ are connected by an edge iff $S(e)$ and $S(e')$ interlace. The segment $S(e_0)$ is strongly planar iff the following conditions hold. Firstly, $S(e)$ is strongly planar for every e emanating from C. Secondly, there is a partition $\{L, R\}$ of the segments emanating from C such that no two segments in L or R interlace and such that $A(e) \cap \{w_1, ...w_{r-1}\} = \emptyset$ for any segment $S(e) \in R$, where $w_0, w_1, ...w_{r-1}, w_r$ is the stem of cycle C.

For reasons of efficiency, it is useful to order the adjacency list of any vertex v as follows: edge (v, w) is before edge (v, w') if min $A((v, w)) <$ min $A((v, w'))$ or if min $A((v, w)) =$ min $A((v, w'))$, $A((v, w))$ has cardinality two, and $A((v, w'))$ has cardinality three or more. In all other cases the order is irrelevant. We assume from now on that the cycle $C(e)$ for a tree edge $e = (x, y)$ is defined in the following way. Starting in y we construct a path by always taking the first edge out of each node until a back edge is encountered. The path *constructed* this way is the spine of the cycle $C(e)$.

The discussion above suggests a procedure *stronglyplanar* (e_0), cf. [Meh84, vol. 2, page 109] that given an edge e_0 decides the strong planarity of the segment $S(e_0)$. It first constructs the cycle $C = C(e_0)$, then recursively tests the strong planarity of all segments $S(e)$, where e emanates from C, and finally tests, whether there is an appropriate bipartition of the vertex set of the interlacing graph. The recursive calls are made in the following order. If $w_{r+1}, ..., w_k$ is the spine of the cycle C then the segments $S((w_k,\ ))$ are tested first, the segments $S((w_{k-1},\ ))$ are tested next,... . For each fixed i the segments $S((w_i,\ ))$ are tested in the order in which the edges $(w_i,\ )$ appear on the adjacency list of w_i. The call *stronglyplanar*$((1, 2))$ tests the strong planarity of segment $S((1, 2))$ and hence the planarity of G.

As shown in [Meh84, vol. 2, page 112] procedure *stronglyplanar* can also be used to compute a labelling α of the edges of G by L and R such that:

- an edge e is labelled iff *stronglyplanar*(e) is called
- edge $(1, 2)$ is labelled L
- if e_0 is a labelled edge and $e_1, ..., e_m$ are the edges emanating from $C = C(e_0)$ then α induces the appropriate bipartition of the interlacing graph, i.e., if $\alpha(e_i) = \alpha(e_j)$ then $S(e_i)$ and $S(e_j)$ do not interlace and if $\alpha(e_j) = R$ then $A(e_j) \cap \{w_1, ..., w_{r-1}\} = \emptyset$ where $w_0, ..., w_r$ is the stem of C.

The correctness proof of procedure *stronglyplanar* demonstrates how a strongly planar embedding of $S(e_0)$ can be obtained from strongly planar embeddings of the $S(e_i)$'s:

To construct a canonical embedding of $S(e_0)$ draw the path $w_0, \ldots, w_k$ (consisting of stem $w_0, \ldots, w_r$, edge $e_0 = (w_r, w_{r+1})$ and spine $(w_{r+1}, \ldots, w_k)$ as a vertical upwards directed path, add edge (w_k, w_0), and then for i, $1 \leq i \leq m$, and $\alpha(e_i) = L$ extend the embedding of $C + S(e_1) + \ldots S(e_{i-1})$ by glueing a canonical embedding of $S(e_i)$ onto the left side of the vertical path, and for i, $1 \leq i \leq m$, and $\alpha(e_i) = R$ extend the embedding of $C + S(e_1) + \ldots + S(e_{i-1})$ by glueing a reversed canonical embedding of $S(e_i)$ onto the right side of the vertical path. Similarly, if the goal is to construct a reversed canonical embedding of $S(e_0)$ then, if $\alpha(e_i) = L$, a reversed canonical embedding of $S(e_i)$ is glued onto the right side of the vertical path, and if $\alpha(e_i) = R$, then a canonical embedding of $S(e_i)$ is glued onto the left side of the vertical path. This completes the review of [Meh84].

We can now give the algorithmic details. We first use procedure *stronglyplanar* to compute the mapping α. We then use a procedure *embedding* to actually compute an embedding. The procedure *embedding* takes two parameters: a tree edge e_0 and a flag $t \in \{L, R\}$. A call *embedding*(e_0, L) computes a canonical embedding of $S(e_0)$ and a call *embedding*(e_0, R)

procedure *embedding*(e_0: edge, t: $\{L, R\}$)
(∗ computes an embedding of $S(e_0)$, $e_0 = (x, y)$, as described in the text;
it returns the lists T and A defined in the text ∗)
find the spine of segment $S(e_0)$ by starting in node y and always
taking the first edge of every adjacency list until a back edge is
encountered. This back edge leads to node w_0.
(∗ Let $w_0, \ldots, w_r$ be the tree path from w_0 to $x = w_r$ and
let $w_{r+1} = y, \ldots, w_k$ be the spine constructed above. ∗)
$AL \leftarrow AR \leftarrow$ empty list of edges;
$T \leftarrow (w_k, w_0)$;
for j **from** k **downto** $r + 1$
do for all edges e' (except the first) emanating from w_j
do if e' is a tree edge
then $(T', A') \leftarrow$ *embedding*$(e', t \oplus \alpha(e'))$
else $T' \leftarrow (e')$, $A' \leftarrow$ (reversal of e')
fi
if $t = \alpha(e')$
then $T \leftarrow T'$ **conc** T; $AL \leftarrow AL$ **conc** A'
else $T \leftarrow T$ **conc** T'; $AR \leftarrow A'$ **conc** AR
fi
od
output (w_j, w_{j-1}) **conc** T; (∗ the cyclic adjacency list of vertex w_j ∗)
let $AL = AL'$ **conc** T' and $AR = T''$ **conc** AR'
where T' and T'' contain all edges incident to w_{j-1};
$AL \leftarrow AL'$; $AR \leftarrow AR'$; $T \leftarrow T'$ **conc** (w_{j-1}, w_j) **conc** T''
od
$A \leftarrow AR$ **conc** (w_0, w_k) **conc** AL;
return T and A
end

Table 1: The procedure *embedding*

computes a reversed canonical embedding of $S(e_0)$. The call *embedding*$((1, 2), L)$ embeds the entire graph.

The embedding of $S(e_0)$ computed by *embedding*(e_0, t) is represented in the following non-standard way:

1. For the vertices $v \in V(e_0)$ we use the standard representation, i.e., the cyclic list of the incident edges corresponding to the clockwise ordering of the edges in the embedding.

2. For the vertices in the stem we use a non-standard representation. For each vertex $w_i \in \{w_0, \ldots, w_r\}$ let the lists $AL(w_i)$ and $AR(w_i)$ be such that the catenation of (w_i, w_{i+1}), $AR(w_i)$, (w_i, w_{i-1}), and $AL(w_i)$ corresponds to the clockwise ordering of the edges incident to w_i in the embedding. Here, $w_{-1} = w_k$. Note that $AR(w_i) = \emptyset$ for $1 \le i < r$ if $t = L$, and $AL(w_i) = \emptyset$ for $1 \le i < r$, if $t = R$. The lists $AL(w_i)$, $AR(w_i)$, $0 \le i \le r$, are returned in an implicit way: $AL(w_r)$ and $AR(w_r)$ are returned as the list $T = AL(w_r), (w_r, w_{r+1}), AR(w_r)$ and the other lists are returned as the list $A = AR(w_{r-1}), \ldots, AR(w_0), (w_0, w_k), AL(w_0), \ldots, AL(w_{r-1})$, cf. Figure 3.

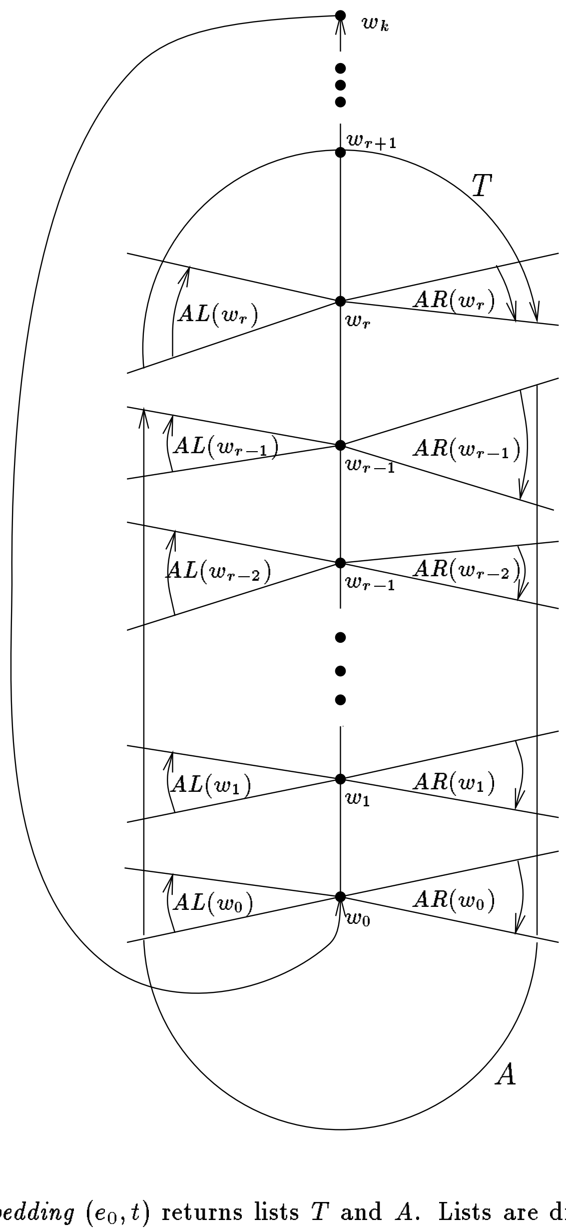

Figure 3: A call *embedding* (e_0, t) returns lists T and A. Lists are drawn as arrows. The arrowhead corresponds to the end of the list.

The procedure *embedding* has the same structure as the procedure *stronglyplanar* and is given in Table 1. It first constructs the stem and the spine (line (1)) of cycle $C(e_0)$, then walks down the spine (lines (3) to (17)), and finally computes the lists T and A to be returned (lines (18) and (19)).

We first discuss the walk down the spine. Suppose that the walk has reached vertex w_j. We first recursively process the edges emanating from w_j (lines (4) to (13)), and then compute the cyclic adjacency list of vertex w_j and prepare for the next iteration (lines (14) to (16)).

We discuss lines (4) to (13) first. In general, some number of edges emanating from w_j and all edges incident to vertices w_l with $l > j$ will have been processed already. Call the processed edges $e_1, \ldots, e_{i-1}$. We can now state the invariant of the loop (4) to (13):

- T concatenated with (w_j, w_{j-1}) is the cyclic adjacency list of vertex w_j in the embedding of $C + S(e_1) + \ldots + S(e_{i-1})$.
- AL and AR are the catenation of the lists $AL(w_0), \ldots, AL(w_{j-1})$ and $AR(w_{j-1}), \ldots, AR(w_0)$ respectively where (w_l, w_{l+1}), $AR(w_l), (w_l, w_{l-1}), AL(w_l)$ is the cyclic adjacency list of vertex w_l, $0 \leq l \leq j-1$, in the embedding of $C + S(e_0) + \ldots + S(e_{i-1})$.

When $i = 0$, i.e., before processing any of the emanating segments the adjacency list of w_j, $0 \leq j \leq k-1$, is $(w_j, w_{j+1}), (w_j, w_{j-1})$ and hence $AL(w_j) = AR(w_j) = \emptyset$. We conclude that T, AL and AR are initialized correctly in line (2).

Assume now that we process edge $e' = e_i$ emanating from w_j. The flag $\alpha(e')$ indicates what kind of embedding of $S(e_i)$ is needed to build a canonical embedding of $S(e_0)$; the opposite kind of embedding of $S(e_i)$ is needed to build a reversed canonical embedding of $S(e_0)$. So the required kind is given by $t \oplus \alpha(e')$, where $L \oplus L = R \oplus R = L$ and $L \oplus R = R \oplus L = R$.

If e' is a tree edge, the call *embedding*$(e', t \oplus \alpha(e'))$ computes the cyclic adjacency lists of the vertices in $V(e')$ and returns lists T' and A' as defined above. If e' is a back edge then T' is simply e' and A' is simply the reversal of e'. If $S(e_i)$ has to be glued to the left side of the vertical path $w_0, \ldots, w_k$, i.e., if $t = \alpha(e')$ then we append T' to the front of T and A' to the end of AL, cf. Figure 4. Analogously, if $S(e_i)$ has to be glued to the right side of the path $w_0, \ldots, w_k$, i.e., if $t \neq \alpha(e')$, then we append T' to the end of T and A' to the front of AR. This clearly implements the glueing process described above and also clearly maintains the invariants.

Suppose now that we have processed all edges emanating from w_j. Then (w_j, w_{j-1}) concatenated with T is the cyclic adjacency list of vertex w_j (line (14)).

We next prepare for the treatment of vertex w_{j-1}. Let T' and T'' be the list of edges incident to w_{j-1} from the left and from the right respectively and having their other endpoint in an already embedded segment. List T' is a suffix of AL and list T'' is a prefix of AR. The catenation of T', (w_{j-1}, w_j), T'', and (w_{j-1}, w_{j-2}) is the current clockwise adjacency list of vertex w_{j-1}. Thus lines (15) and (16) correctly initialize AL, AR, and T for the next iteration.

Suppose now that all edges emanating from the spine of $C(e_0)$ have been processed, i.e., control reaches line (18). At this point, list T is the ordered list of edges incident to w_r (except (w_r, w_{r-1})) and the two lists AL and AR are the ordered list of edges incident to the two sides of the stem of $C(e_0)$. Thus T and the catenation of AR, (w_0, w_k), and AL are the two components of the output of *embedding*(e_0, t). We summarize in

Theorem 1 *Let $G = (V, E)$ be a planar graph. Then G can be turned into a planar combinatorial embedding in linear time.*

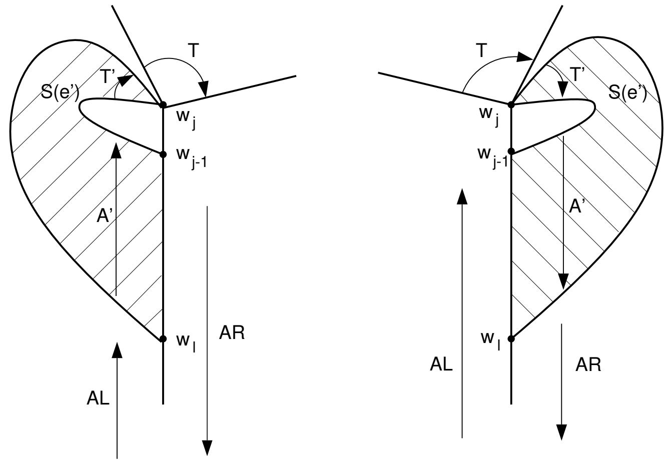

Figure 4: Glueing $S(e')$ to the left or right side of the path $w_0, \ldots, w_k$ respectively.

Proof. The correctness follows immediately from the correctness of procedure *stronglyplanar*, from the fact that *stronglyplanar* correctly computes the mapping α, and from the observation that *embedding* realizes the glueing process described above.

For the running time analysis we only have to observe that every edge is moved at most once from one of the lists AL and AR onto the list T (onto lists T' and T'' in line (15) and then to T in line (16)), that every edge belongs to at most one spine (line (14)) and that all lines except lines (1) and (15) take constant time. □

For an example let us consider the DFS-tree of G given in Figure 1. Consider the situation in the call of *embedding*$((3,4), L)$. The call *embedding*$((4,6), L)$ computes the cyclic adjacency lists of the vertices in $V((4,6))$ and returns lists $T' = (4,6)$ and $A' = (1,6)(2,6)$. In line (10), $T = (4,6)(4,5)$ and $AL = (1,6)(2,6)$. The call of *embedding*$((4,7), R)$ gives $T' = (4,7)$ and $A' = (3,7),(1,7)$. Thus in line (10) we have $T = (4,6)(4,5)(4,7)$ and $AR = (3,7)(1,7)$. The adjacency list of node $w_j = 4$ is completed in line (14). It is $(4,3),(4,6),(4,5),(4,7)$. In line (16) we get $AL = (1,6)(2,6)$, $AR = (1,7)$ and $T = (3,4)(3,7)$. At the end of *embedding*$((3,4), L)$ we have $A = (1,7)(1,5)(1,6)(2,6)$.

An implementation based on this note is described in [MMN93] and is distributed with the LEDA platform of combinatorial and geometric computing [Näh95, MN95] (anonymous ftp (mpi-sb.mpg.de, directory pub/LEDA)). It first determines the connected and biconnected components and then adds edges to make the graph biconnected. It then tests planarity (using procedure *stronglyplanar*). If the graph is found to be non-planar, a subdivision of K_5 or $K_{3,3}$ is identified to prove non-planarity (a trivial method is used for that purpose: the following test is applied to every edge. The edge is removed provisionaly and planarity is tested again. If the graph is still non-planar then the edge is removed. If the graph is now planar the edge is kept. In this way, a subdivision of K_5 or $K_{3,3}$ is found in quadratic time. We should mention that there is a linear time algorithm to identify a K_5 or $K_{3,3}$ in a non-planar graph [Wil84]). If the graph is found to be planar a planar combinatorial embedding is constructed. The resulting planar combinatorial embedding is triangulated, a straight-line embedding is constructed (using either the algorithm in [Sch90] or [dFPP91]), and the result is displayed.

The implementation was extensively tested on three kinds of graphs: hand-crafted examples

number of nodes	1000	2000	4000	8000	16000	32000
number of edges	1921	3890	7842	15777	31684	63558
initializing	0.33	0.80	1.88	5.02	10.88	33.02
planarity testing	0.13	0.27	0.58	1.28	2.73	6.07
embedding	0.15	0.30	0.77	1.80	4.22	9.57
total time	0.61	1.37	3.23	8.10	17.83	48.66

Table 2: Running times in seconds on a SUN SPARC 10 for pseudo-random planar graphs. The first row shows the time to prepare the input graph, i.e., to copy it, to make it biconnected and bidirected, the second row shows the time to test planarity, and the third row gives the time for constructing the embedding.

of small size, random sparse graphs, and pseudeo-random planar graphs. The latter graphs were generated by choosing an appropriate number of random line segments in the unit square, computing their intersections, and putting a vertex on every endpoint and intersection. The running time of the implementation is about 50 times the running time of the LEDA strongly connected components algorithm. Table 2 gives more details.

The measured running times grow slightly more than linear. This is due to the increased number of cache faults for larger input graphs. Many of the actions of the algorithm follow the following pattern: an edge, say (v, w), is explored and then a number of labels of node w are inspected and processed. We have chosen LEDA's node arrays to realize node labels. Thus inspecting k node label corresponds to accesses in k arrays and hence to up to k cache faults. Since processing a node label is typically a simple operation these cache faults show up in the measured running time. An alternative implementation where all node labels are stored directly in the node would incur less slow-down due to cache faults.

References

[BL76] K. Booth and L. Lueker. Testing for the consecutive ones property, interval graphs and graph planarity using PQ-tree algorithms. *J. of Computer and System Sciences*, 13:335–379, 1976.

[CNAO85] N. Chiba, T. Nishizeki, S. Abe, and T. Ozawa. A linear algorithm for embedding planar graphs using PQ-trees. *J. of Computer and System Sciences*, 30(1):54–76, 1985.

[dFPP91] H. de Fraysseix, J. Pach, and R. Pollack. How to draw a planar graph on a grid. *Combinatorica*, 10:41–51, 1991.

[ET76] S. Even and R.E. Tarjan. Computing an st-numbering. *Theoretical Computer Science*, 2:339–344, 1976.

[FR82] H.de Fraysseix and P. Rosenstiehl. A depth–first–search characterization of planarity. *Annals of Discrete Mathematics*, 13:75–80, 1982.

[HT74] J. Hopcroft and R. Tarjan. Efficient planarity testing. *J. ACM*, 21(4):549–568, 1974.

[LEC67] A. Lempel, S. Even, and I. Cederbaum. An algorithm for planarity testing of graphs. *Theory of Graphs, Int. Symp.(Rome 1966)*, pages 215–232, 1967.

[Meh84] K. Mehlhorn. *Data Structures and Efficient Algorithms*, volume I, II, III. Springer Verlag, Berlin, 1984.

[MMN93] K. Mehlhorn, P. Mutzel, and St. Näher. An implementation of the Hopcroft and Tarjan planarity test and embedding algorithm. Technical Report MPI–I–93–151, Max–Planck–Institut für Informatik, Saarbrücken, 1993.

[MN95] K. Mehlhorn and St. Näher. LEDA: A library of efficient data types and algorithms. *CACM*, 38(1):96–102, 1995.

[Mut92] P. Mutzel. A fast linear time embedding algorithm based on the Hopcroft-Tarjan planarity test. Technical report, Universität zu Köln, 1992.

[Näh95] St. Näher. LEDA Manual Version 3.1. Technical Report MPI-I-95-1-002, Max-Planck-Institut für Informatik, 1995.

[Sch90] W. Schnyder. Embedding planar graphs on the grid. In *Proc. 1st ACM-SIAM Symp. Discr. Alg. (SODA), San Francisco*, pages 138–148, 1990.

[Wil84] S.G. Williamson. Depth-first search and Kuratowksi subgraphs. *Journal of the ACM*, 11:681–693, 1984.