JOURNAL OF COMPUTER AND SYSTEM SCIENCES 30, 54–76 (1985)

A Linear Algorithm for Embedding Planar Graphs Using *PQ*-Trees

NORISHIGE CHIBA AND TAKAO NISHIZEKI

Faculty of Engineering, Tohoku University, Sendai 980, Japan

SHIGENOBU ABE

Basic Software Corporation, Sendai 980, Japan

AND

TAKAO OZAWA

Faculty of Engineering, Kyoto University, Kyoto 606, Japan

Received October 28, 1983; revised August 7, 1984

This paper presents a simple linear algorithm for embedding (or drawing) a planar graph in the plane. The algorithm is based on the "vertex-addition" algorithm of Lempel, Even, and Cederbaum ("Theory of Graphs," Intl. Sympos. Rome, July 1966, pp. 215–232, Gordon & Breach, New York, 1967) for the planarity testing, and is a modification of Booth and Lueker's (*J. Comput. System Sci.* **13** (1976), 335–379) implementation of the testing algorithm using a *PQ*-tree. Compared with the known embedding algorithm of Hopcroft and Tarjan (*J. Assoc. Comput. Mach.* **21**, No. 4 (1974), 549–568), this algorithm is conceptually simple and easy to understand or implement. Moreover this embedding algorithm can find all the embeddings of a planar graph. © 1985 Academic Press, Inc.

1. INTRODUCTION

Planarity testing, that is, determining whether a given graph is planar or not, has many applications, such as the design of VLSI circuits and determining isomorphism of chemical structures. Two planarity testing algorithms of different types are known, both running in linear time. One is called a "path addition algorithm," and the other a "vertex addition algorithm." These terms "path addition" and "vertex addition" express well the principles of the algorithms. The path addition algorithm was first presented by Auslander and Parter [1] and Goldstein [6], and improved later into a linear algorithm by Hopcroft and Tarjan [9]. The vertex addition algorithm was first presented by Lempel, Even, and Cederbaum [10], and improved later into a linear algorithm by Booth and Lueker

[2] employing an st-numbering algorithm [5] and a data structure called a "PQ-tree."

Many applications require not only testing the planarity but also embedding (or drawing) a planar graph in the plane [3]. Hopcroft and Tarjan mentioned that an embedding algorithm can be constructed by modifying their testing algorithm [9]. 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 a "special path."

In this paper we present a very simple linear algorithm for embedding planar graphs, which is based on the vertex addition algorithm of Booth and Lueker. Their testing algorithm adds one vertex in each step. Previously embedded edges incident to this vertex are connected to it, and new edges incident to it are embedded and their end vertices are left unconnected. Sometimes whole pieces have to be flipped around or permuted. If the representation of the embedded subgraph is updated with each alteration of the embedding, then the final representation will be an actual embedding of a given whole graph. Thus the testing algorithm directly yields an $O(n^2)$ time embedding algorithm. Throughout this paper n denotes the number of vertices of a given graph.

Our $O(n)$ embedding algorithm runs in two phases. In the first phase the algorithm embeds the directed graph obtained from a given planar (undirected) graph by assigning a direction to every edge from the end having a greater st-number to the other end. In the second phase the algorithm extends the embedding of the directed graph into an embedding of the given planar (undirected) graph.

If a planar graph is not 3-connected then an embedding of the graph is not unique. We also show that our algorithm can be easily modified so as to construct an expression for all the embeddings of a planar graph.

2. Preliminary

In this section, we define some terms and present some known results. Let $G = (V, E)$ be a graph with vertex set V and edge set E. We assume that G is simple, that is, has no multiedges or loops. Throughout this paper n denotes the number of vertices of G, that is, $n = |V|$. A graph G is *planar* if it is embeddable in the plane without any crossing edges. A *neighbour* of a vertex $v \in V$ is a vertex adjacent to v. A graph G is represented by a set A of n lists, called "adjacency lists"; the list $A(v)$ for vertex $v \in V$ contains all the neighbours of v. For each $v \in V$ an actual drawing of a planar graph G determines, within a cyclic permutation, the order of v's neighbours embedded around v. *Embedding* a planar graph G means to construct adjacency lists A of G such that, in each $A(v)$, all the neighbours of v appear in clockwise order with respect to an actual drawing. Such a set A of adjacency lists is called an *embedding* of G. A graph G is planar if and only if the nonseparable components of G are planar [7]. Moreover, one can easily obtain an embedding of the entire graph G from embeddings of all the nonseparable components of G. Hence we assume that G is nonseparable. Since we are interested in embedding a graph rather than testing the planarity, we assume that a planar graph is given. An *st-numbering* is a numbering of the vertices of G by $1, 2, \ldots, n$ such that the two vertices "1" and "n" are necessarily adjacent and each j of the other vertices is adjacent to two vertices i and k satisfying $i<j<k$. The vertex "1" (resp. n) is called a *source* (resp. *sink*) and denoted by s (resp. t). Every nonseparable graph G has an st-numbering [10], and Even and Tarjan gave a linear algorithm for it [5]. The st-numbering plays a crucial role in a vertex addition algorithm. From now on we refer the vertices of G by their st-numbers. Let $G_k=(V_k, E_k)$ be the subgraph of G induced by the vertices $1, 2, \ldots, k$. If $k<n$ then there must exist an edge of G with one end in V_k and the other in $V-V_k$. Let G'_k be the graph formed by adding to G_k all these edges. These edges are called *virtual edges*, and their ends in $V-V_k$ are called *virtual vertices* and labeled as their counterparts in G, but they are kept separate; i.e., there may be several virtual vertices with the same label, each with exactly one entering edge. Let B_k be an embedding of G'_k such that all the virtual vertices are placed on the outer face. B'_k is called a *bush form* of G'_k. The virtual vertices are usually placed on a horizontal line. G, G_k, and B_k are illustrated in Fig. 1. Lemma 1 implies that every planar graph G has a bush form B_k for $1 \leqslant k \leqslant n$.

LEMMA 1 [4]. *Let $1 \leqslant k \leqslant n$. If edge (s, t) is drawn on the boundary of the outer face in an embedding of G, then all the vertices and edges of $G-G_k$ are drawn in the outer face of the plane subgraph G_k of G.*

An *upward graph* D_u of a graph G is defined to be a directed graph obtained from G by assigning a direction to every edge so that it goes from the larger end to the smaller. An *upward embedding* A_u of G is an embedding of the directed graph D_u. In an embedding A of an *undirected* graph G, a vertex v appears in list $A(w)$ and w appears in list $A(v)$ for every edge (v, w). However, in an upward embedding A_u of G, the head w appears in $A_u(v)$ but the tail v does not appear in $A_u(w)$ for every directed edge $\langle v, w\rangle$. Figure 2 depicts an upward graph D_u and an upward embedding A_u for the graph G in Fig. 1a.

Booth and Lueker [2] improved the vertex addition algorithm given by Lempel,

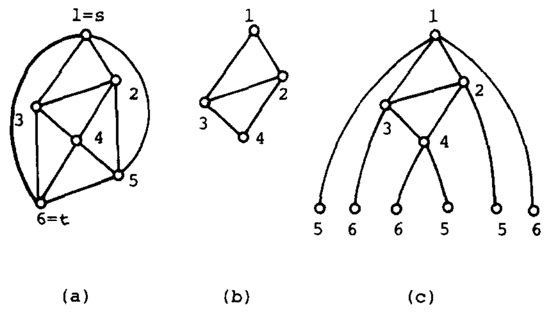


FIG. 1. Illustrations of (a) st-numbered graph G, (b) G_4, and (c) bush form B_4.

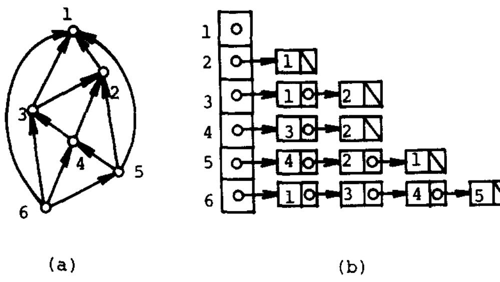


FIG. 2. Illustrations of (a) upward graph D_u, and (b) upward embedding A_u for a graph G in Fig. 1.

Even, and Cederbaum into a linear time algorithm by using a data structure "*PQ*-tree" to represent B_k. A *PQ*-tree consists of "*P*-nodes," "*Q*-nodes," and "leaves." A *P*-node repreents a cut vertex of G_k, so the sons of a *P*-node can be permuted arbitrarily. A *Q*-node represents a nonseparable component of G_k, and the sons of a *Q*-node are allowed only to reverse (flip over). A leaf indicates a virtual vertex of B_k. In an illustration of a *PQ*-tree, a *P*-node is drawn by a circle and a *Q*-node by a rectangle. A bush form B_k and a *PQ*-tree representing B_k are illustrated in Fig. 3. Thus, a *PQ*-tree represents all the permutations and the reversions possible in a bush form B_k. The key idea of the vertex addition algorithm is to reduce the planarity testing of G_{k+1} to the problem which asks for permutations and reversions to make all the virtual vertices labelled $k+1$ occupy consecutive positions. Lemma 2 guarantees that this reduction is possible.

LEMMA 2 [4]. *Let B_k be any bush form of subgraph G_k of a planar graph G. Then there exists a sequence of permutations and reversions to make all the virtual vertices labelled $k+1$ occupy consecutive positions on the horizontal lines.*

Booth and Lueker [2] showed that such permutations and reversions can be found by repeatedly applying the nine transformation rules called the *template matchings* to the *PQ*-tree. A leaf labelled $k+1$ is said to be *pertinent* in the *PQ*-tree corresponding to B_k. The *pertinent subtree* is the minimal subtree of a *PQ*-tree containing all the pertinent leaves [2]. A node of a *PQ*-tree is said to be *full* if all the

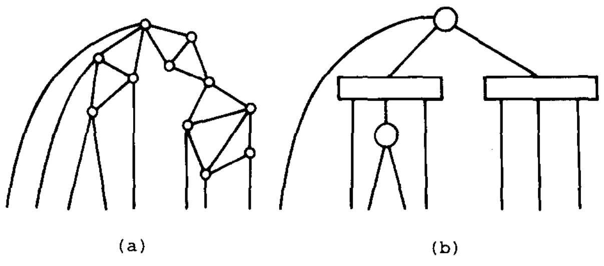


FIG. 3. Illustrations of (a) bush form B_k, and (b) *PQ*-tree.

leaves among its descendants are pertinent. The following is the outline of the planarity testing algorithm PLANAR using a PQ-tree [2].

```
procedure PLANAR ($G$);
  {$G$ is a given graph}
  begin
    assign $st$-numbers to all the vertices of $G$;
    construct a $PQ$-tree corresponding $G_1'$; {a single $P$-node with virtual edges
    incident to source $s = 1$}
    for $v := 2$ to $n$
      do begin
        {reduction step}
          try to gather all the pertinent leaves by repeatedly applying the tem-
          plate matchings from the leaves to the root of the pertinent subtree;
          if the reduction fails
```

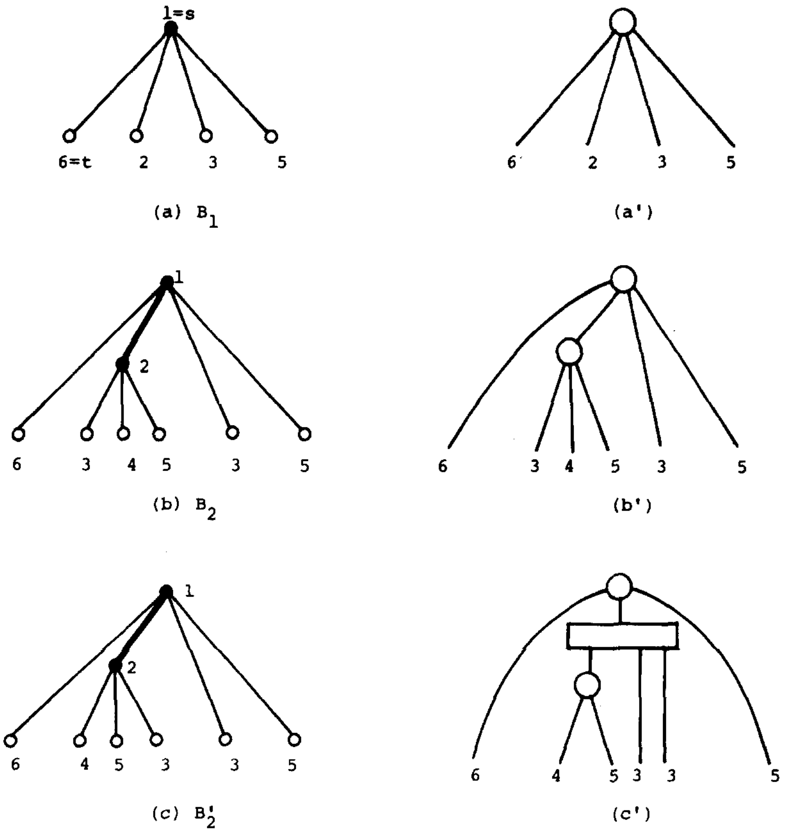
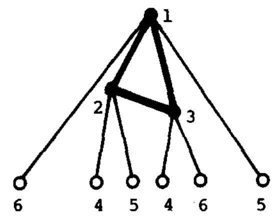
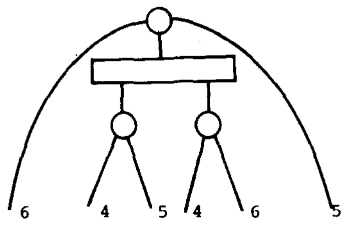


FIG. 4. Reduction process of PLANAR for the graph G in Fig. 1a; (a)–(i) bush forms, and (a')–(i') PQ-trees.

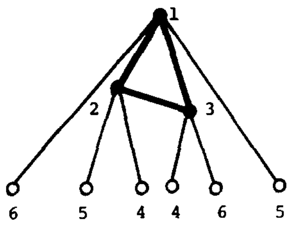
```
            then begin
                    print out the message "$G$ is nonplanar";
                    return
                 end;
        {vertex addition step}
          replace all the full nodes of the $PQ$-tree by a new $P$-node (which
          corresponds to a cut vertex $v$ in $G'_v$);
          add to the $PQ$-tree all the neighbours of $v$ larger than $v$ as the sons of
          the $P$-node
        end;
    print out the message "$G$ is planar"
end;
```



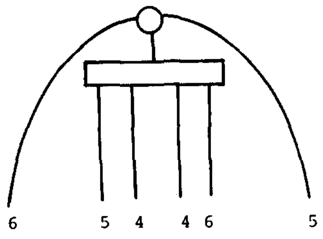
(d) B_3



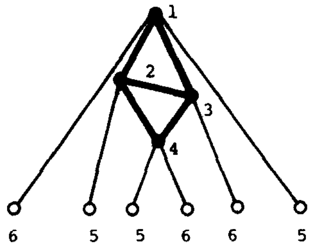
(d')



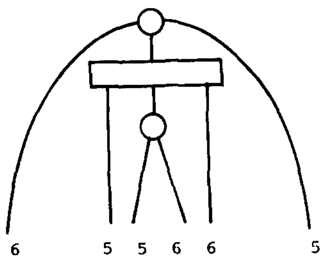
(e) B'_3



(e')



(f) B_4

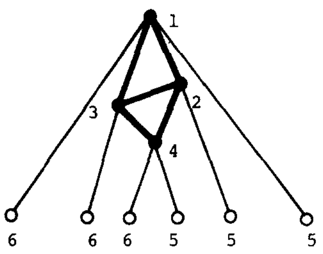


(f')

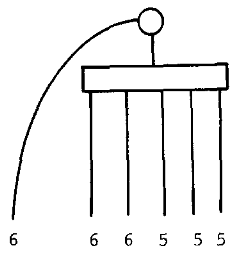
FIGURE 4 *(continued)*

Figure 4 illustrates a sequence of bush forms together with the corresponding *PQ*-trees appeared in the execution of PLANAR for the graph G in Fig. 1a. B'_k is a bush form just after the "reduction step" for vertex $k+1$.

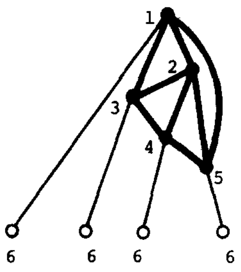
Clearly the time spent by the vertex addition step for v is proportional to the degree of v. Therefore the step spends at most $O(n)$ time in total. The time spent by the reduction step for v is proportional to the number of leaves plus the number of unary nodes in the pertinent tree. Therefore it is not straightforward to show that all the reduction steps spend at most $O(n)$ time in total. However Booth and Lueker [2] showed that the time is proportional to the number of vertices. Thus the algorithm spends at most $O(n)$ time in total.



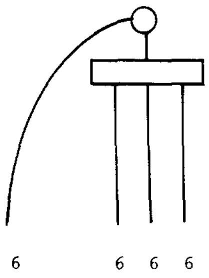
(g) B'_4



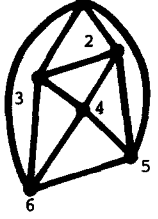
(g')



(h) B_5



(h')



(i) $G=G_6=B_6$

(i')

FIGURE 4 *(continued)*

3. Linear Time Embedding Algorithm

In this section, we present a linear time embedding algorithm EMBED based on the planarity testing algorithm PLANAR.

One can easily have the following naive embedding algorithm: first write down the partial embedding of the graph corresponding to B_1; and, with each reduction of the PQ-tree, rewrite (the adjacency lists of) the bush form. Clearly the final bush form is indeed an embedding of the graph. Unfortunately the algorithm spends $O(n^2)$ time, since it takes $O(n)$ time per reduction of the PQ-tree to update the adjacency lists of the bush form.

Our linear algorithm EMBED runs in two phases: in the first phase EMBED obtains an upward embedding A_u of a planar graph G (by an efficient implementation of the above naive algorithm); in the second phase EMBED constructs an entire embedding A of G from A_u.

3.1. *Algorithm for Extending A_u into A*

In this subsection we describe an algorithm for the second phase. One can easily observe

Lemma 3. *Let A be an embedding of a planar graph G obtained by the naive algorithm above, and let v be a vertex of G. Then all the neighbours smaller than v are embedded consecutively around v. (See Fig. 5.) That is, $A(v)$ does not contain four neighbours w_1, w_2, w_3, and w_4, appearing in this order and satisfying $w_1, w_3 < v$ and $w_2, w_4 > v$.*

Proof. Immediately follows from Lemmas 1 and 2. Q.E.D.

As shown later in Subsection 3.2, EMBED finds an upward embedding A_u of G such that, for each vertex $v \in V$, the neighbours $x_1, x_2, \dots, x_i$ smaller than v appear in $A_u(v)$ in this order as indicated by a dotted arrow in Fig. 5. That is, the neighbour

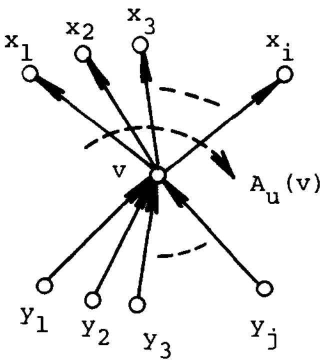


Fig. 5. Embedding of the neighbours of v. (Numbers $x_1, x_2, \dots, x_i$ are all less than v, and $y_1, y_2, \dots, y_j$ are greater than v.)

of v embedded around v counterclockwise next to the top entry x_i of $A_u(v)$ is greater than v. In particular, the top entry of $A_u(t)$, where $t = n$ is the sink, is the source $s(=1)$. Now we present the algorithm "ENTIRE-EMBED" for extending such an upward embedding A_u into an embedding A of a given graph. The algorithm executes once the depth-first search starting at sink t on a directed graph D_u. The algorithm adds vertex y_k to the top of list $A_u(v)$ when directed edge $\langle y_k, v \rangle$ is searched.

```
procedure ENTIRE-EMBED;
  begin
    copy the upward embedding $A_u$ to the lists $A$;
    mark every vertex "new";
    $T := \emptyset$; {A DFS-tree $T$ is constructed only for analysis of the algorithm}
    DFS($t$)
  end;

procedure DFS($y$);
  begin
    mark vertex $y$ "old";
    for each vertex $v \in A_u(y)$
      do begin
           insert vertex "$y$" to the top of $A_u(v)$;
           if $v$ is marked "new"
             then begin
                      add edge $\langle y, v \rangle$ to $T$;
                      DFS($v$)
                  end
         end
  end;
```

We have the following result on the algorithm.

LEMMA 4. *Let D_u be an upward graph of a given graph G, and let A_u be an upward embedding of D_u. Then, the algorithm ENTIRE-EMBED extends A_u into an embedding A of G within linear time.*

Proof. Clearly the algorithm terminates within linear time since the algorithm merely executes the depth-first search once. Thus we concentrate on the correctness. The definition of an st-numbering implies that there exists a directed path from t to every vertex. Therefore DFS(t) traverses all the vertices and so all the directed edges of D_u. (This is not necessarily true for an arbitrary directed graph.) Hence the final list $A(v)$ contains not only the neighbours of v larger than v but also those smaller than v. That is, the final lists A are surely adjacency lists of a given (undirected) graph G. Hence we shall prove that all the entries of A are stored correctly in clockwise order.

By Lemma 3 all of v's neighbours $x_1, x_2, \ldots, x_i$ smaller than v appear in $A_u(v)$ in this order. The algorithm first copies list $A_u(v)$ to list $A(v)$ and then adds each neighbour y of v larger than v to the top of $A(v)$ in order of directed edge $\langle y, v\rangle$ being searched. Therefore it suffices to show that directed edges $\langle y_1, v\rangle$, $\langle y_2, v\rangle, \ldots, \langle y_j, v\rangle$ are searched in this order. (See Fig. 5.) Assume to the contrary that $\langle y_k, v\rangle$ and $\langle y_l, v\rangle$ are searched in this order although $k > l$. Let P_k be the path from t to y_k, and let P_l be the path from t to y_l in the DFS-tree T. (See Fig. 6.) Let z be the vertex at which path P_l leaves P_k, and let $\langle z, y'_k\rangle \in P_k$ and $\langle z, y'_l\rangle \in P_l$. Thus the vertex y'_k precedes y'_l in $A_u(z)$. Moreover the subpaths $P'_k = z \cdot y'_k \cdots y_k$ of P_k and $P'_l = z \cdot y'_l \cdots y_l$ of P_l have no common vertices other than z. Therefore the two paths P'_k and P'_l together with two edges (y_l, v) and (y_k, v) form a cycle C. All the vertices of $A_u(v)$ must lie in the interior of the cycle; otherwise Lemma 3 would be violated. Since source $s(=1)$ is located on the boundary of the exterior face, the vertex v is not s. By the definition of an st-numbering, the DFS-tree T contains a descending path P from v to s, all the vertices of which are smaller than or equal to v. Since s lies in the exterior of the cycle C, P must intersect the cycle C. However all the vertices of C are larger than or equal to v. This is a contradiction. Q.E.D.

3.2. *Algorithm for Constructing A_u*

In this subsection we give an algorithm for constructing A_u. One can easily obtain list $A_u(v)$ or its reversion by scanning the leaves labelled v in the vertex addition step for v. If $A_u(v)$ is correctly determined in the step, then, counting the number of subsequent v's reversions, one can correct the direction of $A_u(v)$ simply by reversing $A_u(v)$ if the number is odd. However a naive counting algorithm takes

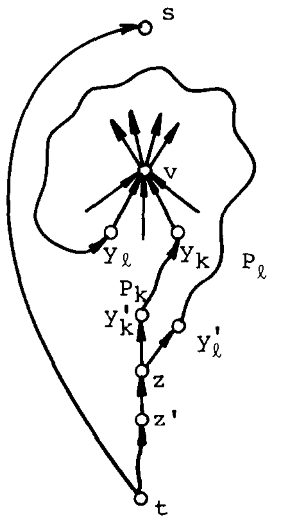


FIG. 6. Illustration for the proof of Lemma 4.

$O(n^2)$ time. Moreover, the information on v may disappear from the PQ-tree. Thus an appropriate device is necessary to trace $A_u(v)$'s reversions.

We first show how to scan all the leaves labelled v. Find the root r of the pertinent subtree by using the "bubble up" procedure of Booth and Lueker [2]. Let $b_1, b_2, \ldots, b_m$ be the maximal sequence of full brothers that are sons of r. (See Fig. 7a.) To obtain $A_u(v)$, we scan the subtree rooted at b_i by the depth-first search for $i = 1, 2, \ldots, m$ in this order. In a schematic illustration of a PQ-tree, one can easily recognize the direction of the maximal sequence, that is, whether $b_1, b_2, \ldots, b_m$ are in left-to-right or right-to-left order. However in the data structure of a PQ-tree, a Q-node is doubly linked only with the endmost sons, and a son of a Q-node has pointers only to the immediate brothers [2]. Therefore we must traverse sons of a Q-node from a full son to one of the endmost sons, and then check the direction of the sequence by using the pointer between the endmost son and the Q-node. Thus such a straightforward method requires $O(n)$ time to determine the direction of the sequence, that is, to know whether the constructed list is either $A_u(v)$ or its reversion.

Our algorithm does not determine the direction of $A_u(v)$ at the vertex addition step for v, but adds a new special node to the PQ-tree as one of r's sons at an arbitrary position among them. The new node is called a "direction indicator," also labelled v, and depicted by a triangle, as illustrated in Fig. 7b. The indicator v plays

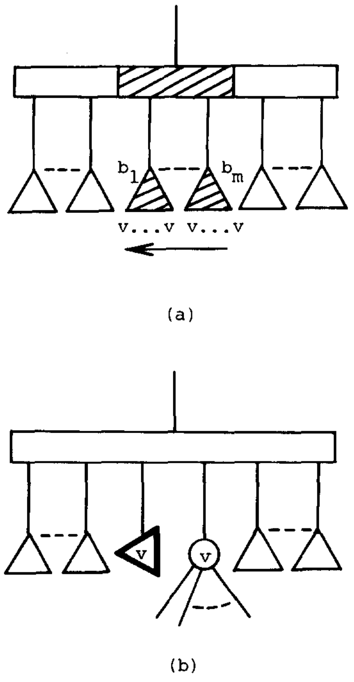


FIG. 7. Illustrations of (a) direction of a scanning, and (b) direction indicator v.

two roles. The first is to trace the subsequent reversions of $A_u(v)$. The indicator will be reversed with each reversion of its father. (No physical action is taken in the indicator's reversion—it is only done implicitly.) The second is to bear the relative direction of node v to its brothers. When the rightmost or leftmost brother of v is subsequently scanned together with the indicator v, the direction of the constructed $A_u(v)$ is known and so is corrected if necessary.

In our template matching algorithm, we ignore the presence of the direction indicators; our matching algorithm is essentially that of [2]. When we access an immediate brother b of a node v, we skip the direction indicators between v and b, if any. When we change pointers of a PQ-tree in a reduction step, we treat a direction indicator as a usual node of a PQ-tree. Note that all the direction indicators in a PQ-tree are necessarily leaves: none of the direction indicators has a son. Now we redefine a node to be full if all the leaves of its descendants which are not indicators are labelled v. Thus we modify the vertex addition step in PLANAR as follows.

```
{vertex addition step}
begin
  let $l_1, l_2,..., l_j$ be the leaves labelled "$v$" and $f_1, f_2,..., f_k$ be the direction indicators
  scanned (using the DFS procedure just described) in this order {it is not
  necessary to recognize here whether $l_1, l_2,..., l_j$ are in left-to-right order};
  $A_u(v) := \{l_1, l_2,..., l_j\}$;
  if root $r$ of the pertinent subtree is not full {the subtree has a leaf which is not an
     indicator and not labelled "$v$"}
  if then {the root $r$ is a $Q$-node}
      begin add a indicator "$v$" directed from $l_j$ to $l_1$ to the $PQ$-tree as a son of the
            $Q$-node $r$ {at an arbitrary position among the sons};
            add the direction indicators $f_1, f_2,..., f_k$ as sons of the $Q$-node $r$ {at
            arbitrary positions among the sons}
      end
    else
      begin {the pertinent subtree corresponds to a reversible component in an
            embedding of $G$, that is, the cut vertex of $G$ corresponding to root $r$
            forms a "separation pair" with vertext $v$. Therefore we may assume
            that $A_u(v)$ is correctly in clockwise order.}
      delete $f_1, f_2,..., f_k$ from the $PQ$-tree;
      for $i := 1$ to $k$
         do if indicator $f_i$ is directed from $l_1$ to $l_k$
              then reverse the adjacency list $A_u(f_i)$;
                   {The order of $A_u(f_i)$ is corrected with the assumption that
                   $A_u(v)$ is in clockwise order.}
      end;
  if root $r$ is not full
    then replace all the full sons of $r$ by a $P$-node {which corresponds a cut vertex
         $v$ of $G'_v$}
    else replace the pertinent subtree by a $P$-node;
add all the virtual vertices adjacent to $v$ (i.e. all neighbours of $v$ in $G$ greater than $v$)
to the $PQ$-tree as the sons of the $P$-node
end;
```

We call this revised algorithm UPWARD-EMBED, on which we have

LEMMA 5. *The algorithm UPWARD-EMBED obtains an upward embedding A_u of a given planar graph G.*

Proof. Let $v \in V$. Clearly the list $A_u(v)$ obtained by UPWARD-EMBED contains all the neighbours of v smaller than v. Clearly these vertices appear in either clockwise or counterclockwise order around v. Therefore we shall show that the vertices in each $A_u(v)$ appear in clockwise order. It suffices to consider the following two cases.

Case 1. *The direction indicator v is not added to the PQ-tree.* The leaves of the pertinent subtree which are not indicators are all labelled v at the vertex addition step for v. Such a pertinent subtree corresponds to a reversible component in a plane embedding of G. (See Fig. 8.) Therefore one may assume that the vertices in $A_u(v)$ appear in clockwise order even in the final embedding.

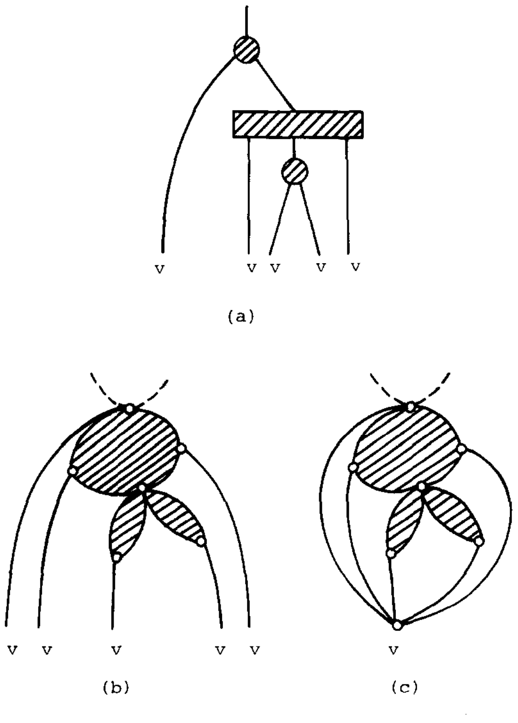


FIG. 8. Reversible component: (a) pertinent subtree, (b) B_{v-1}, and (c) G_v.

Case 2. The direction indicator v is added to the PQ-tree. When the algorithm terminates, the PQ-tree consists of exactly one isolated P-node, and hence has no direction indicators in particular. That is, every indicator will be eventually deleted. Therefore one can assume that the indicator v is deleted in the vertex addition step for a vertex $w(>v)$. The direction indicator v follows reversions of the Q-node which is the father of node v as long as v remains in a PQ-tree. Therefore if the direction of indicator v is opposite relative to the scanning of the leaves $l_1, l_2, ..., l_j$ labelled w, then either the order (clockwise or counterclockwise) of $A_u(v)$ is the same as $A_u(w)$ and vertex v is reversed an odd number of times, or the order of $A_u(v)$ is opposite to that of $A_u(w)$ and the vertex v is reversed an even number of times. In either case, we can correct adjacency list $A_u(v)$ simply by reversing it. Since the pertinent subtree for w corresponds to a reversible component of G, the direction indicator w is not added to the PQ-tree. Hence the adjacency lists $A_u(v)$ and $A_u(w)$ are never reversed after the vertex addition step for w. Thus $A_u(v)$ remains to be correctly in clockwise order. Q.E.D.

However, algorithm UPWARD-EMBED, as it is, requires $O(n^2)$ time since it may scan the same indicator many times, say up to $O(n)$ times. Thus we shall refine the algorithm so that it runs in $O(n)$ time.

Now consider the role of a direction indicator in detail. Assume that root r of a pertinent subtree is not full, and define indicators v and $f_1, f_2, ..., f_k$ as in the algorithm. After the direction indicator v is added to a PQ-tree, indicators v and $f_1, f_2, ..., f_k$ are reversed all together. Therefore it suffices to remember the directions of $f_1, f_2, ..., f_k$ relative to that of v. Thus we delete the indicators $f_1, f_2, ..., f_k$ from the PQ-tree and store them in $A_u(v)$ together with vertices $l_1, l_2, ..., l_j$. Once the correct order of adjacency list $A_u(v)$ is known, we can easily correct the orders of adjacency lists $A_u(f_i)$, $1 \leqslant i \leqslant k$, simply by checking the direction of indicator f_i in $A_u(v)$. We execute such a correction for each v, $v = n, n-1, ..., 1$ in this order.

The following is the algorithm UPWARD-EMBED refined as above. Figure 9 ilustrates an UPWARD-EMBED applied to the graph in Fig. 1a.

```
procedure UPWARD-EMBED($G$);
  {$G$ is a given planar graph}
  begin
    assign $st$-numbers to all the vertices of $G$;
    construct a $PQ$-tree corresponding $G'_1$;
    for $v := 2$ to $n$
      do begin
        {reduction step}
          apply the template matchings to the $PQ$-tree, ignoring the direction
          indicators in it, so that the leaves labelled "$v$" occupy consecutive
          positions;
        {vertex addition step}
          let $l_1, l_2, ..., l_k$ be the leaves labelled "$v$" and direction indicators scan-
          ned in this order;
        delete $l_1, l_2, \ldots, l_k$ from the $PQ$-tree and store them in $A_u(v)$;
        if root $r$ of the pertinent subtree is not full
          then
            begin
              add an indicator "$v$", directed from $l_k$ to $l_1$, to the $PQ$-tree as a
              son of root $r$ at an arbitrary position among the sons;
              replace all the ful sons of $r$ by a new $P$-node
            end
          else replace the pertinent subtree by a new $P$-node;
        add to the $PQ$-tree all the virtual vertices adjacent to $v$ as the sons of
        the $P$-node
    end;
```

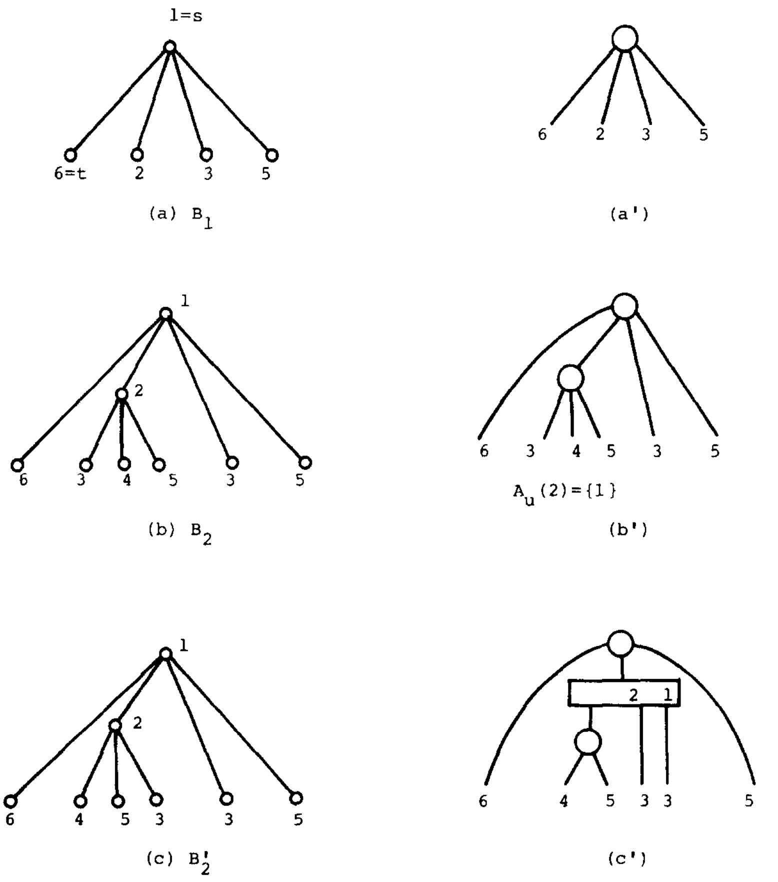
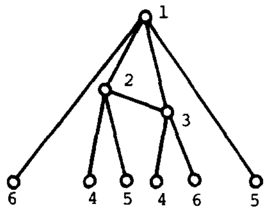


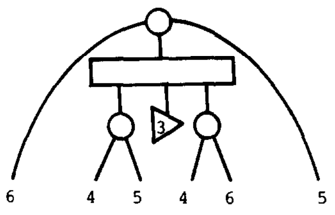
FIG. 9. Process of UPWARD-EMBED applied to G in Fig. 1a; (a)–(i) bush forms, (a′)–(i′) P-Q-trees and lists A_u, and (j) corrected lists A_u.

```
{correction step}
  for $v := n$ downto 1
    do for each element $x$ in $A_u(v)$
        do if $x$ is a direction indicator
            then begin
                    delete $x$ from $A_u(v)$;
                    let $w$ be the label of $x$;
                    if the direction of indicator $x$ is opposite to that of $A_u(v)$
                      then reverse list $A_u(w)$;
                 end
end;
```

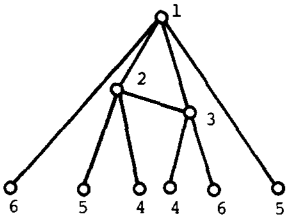
We have the following result on the revised UPWARD-EMBED above.



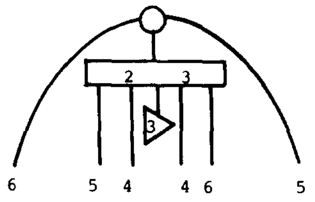
(d) B_3



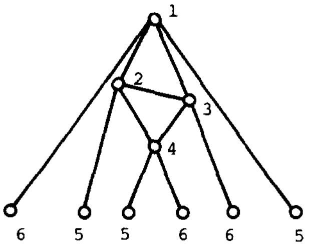
$A_u(3)=\{2,1\}$
(d')



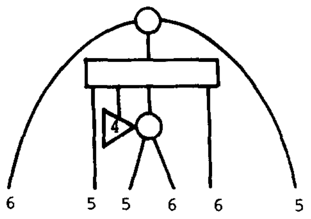
(e) B_3'



(e')



(f) B_4

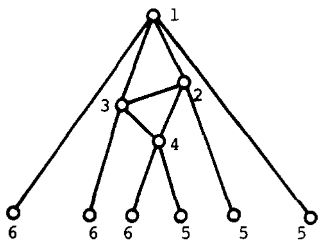


$A_u(4)=\{2, \triangleright 3, 3\}$
(f')

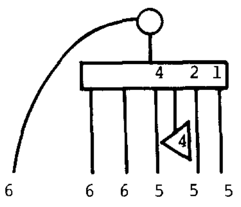
FIGURE 9 *(continued)*

LEMMA 6. *Algorithm UPWARD-EMBED obtains the upward embedding A_u of a given planar graph within linear time.*

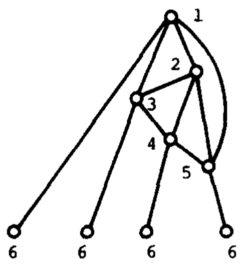
Proof. Noting the role of a direction indicator, one can easily verify the correctness of the algorithm. Therefore we consider the time required by the algorithm. At most $O(n)$ direction indicators are generated during an execution of the algorithm. A direction indicator scanned in a reduction step will be necessarily deleted from a PQ-tree in the succeeding vertex addition step. Therefore each direction indicator is scanned at most once. Thus UPWARD-EMBED requires at most



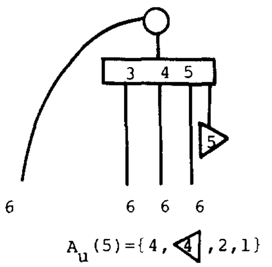
(g) B'_4



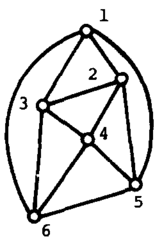
(g')



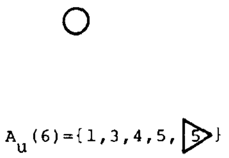
(h) B_5



(h')



(i) $G=G_6=B_6$



(i')

$A_u(1)=\{\}$	$A_u(4)=\{3,2\}$
$A_u(2)=\{1\}$	$A_u(5)=\{4,2,1\}$
$A_u(3)=\{1,2\}$	$A_u(6)=\{1,3,4,5\}$

(j)

FIGURE 9 *(continued)*

$O(n)$ time in addition to the time required by the linear testing algorithm PLANAR. Therefore UPWARD-EMBED runs in linear time. Q.E.D.

The following is the entire algorithm EMBED for embedding a planar graph.

```
procedure EMBED($G$);
  begin
    UPWARD-EMBED;  {phase 1}
    ENTIRE-EMBED   {phase 2}
  end;
```

We have Theorem 1 from Lemmas 1 to 6.

THEOREM 1. *Algorithm EMBED obtains a plane embedding A of a given planar graph within linear time.*

4. Finding All the Embeddings

In this section we present an algorithm for finding all the embeddings of a planar graph. Assume as in the previous sections, that $G = (V, E)$ is a 2-connected planar graph in which vertices are st-numbered. We use edge (t, s) as the reference of all the embeddings, that is, we consider the embeddings of G such that (t, s) lies on the boundary of the outer face in clockwise direction. We define a "separation pair" and a "split component" [8] with slight modification as follows.

Let $\{x, y\}$ be a pair of vertices in G. Then we can partition the edge set E into equivalence classes $E_1, E_2, \ldots, E_l$ such that two edges which lie on a common path not containing any vertex of $\{x, y\}$ except as an end vertex are in the same class. If there are at least two equivalence classes E_i, E_j such that $|E_i|, |E_j| \geqslant 2$ then $\{x, y\}$ is called a *separation pair* of G. A subgraph $G_i = (V_i, E_i)$ of G induced by E_i, $1 \leqslant i \leqslant l$, is called an *$\{x, y\}$-split component* of G if $|E_i| \geqslant 2$ and $(s, t) \notin E_i$, where V_i is the set of vertices to which at least one edge in E_i is incident. If $\{s, t\}$ is not a separation pair, the graph obtained from G by deleting edge (s, t) is called an *$\{s, t\}$-component.*

Now for a particular separation pair $\{x, y\}$ of G we get different embeddings of G by the following operations:

(i) permute the $\{x, y\}$-split components and edge (x, y) if any (see Fig. 10a), or

(ii) reverse (flip over) any number of the $\{x, y\}$-split components (see Fig. 10b).

If we exhaust all the possible operations (i) and (ii), we get all the possible embeddings obtained from the pair $\{x, y\}$. We also see that if $\{s, t\}$ is not a separation pair, a different embedding of G is obtained by the following operation.

(iii) reverse the $\{s, t\}$-component (see Fig. 10c).

From the definition of st-numbering of G, we get the following lemma for a separation pair $\{x, y\}$ of G.

LEMMA 7. *Let $x < y$, and let v be any vertex in an $\{x, y\}$-split component. Then v satisfies $x \leqslant v \leqslant y$.*

Proof. The component contains neither s nor t except the case $x = s$ or $y = t$. By the definition of an st-numbering there exist a descending path from v to $s = 1$ and also an ascending path from v to $t = n$. Both must pass through x or y. Therefore we have $x \leqslant v \leqslant y$. Q.E.D.

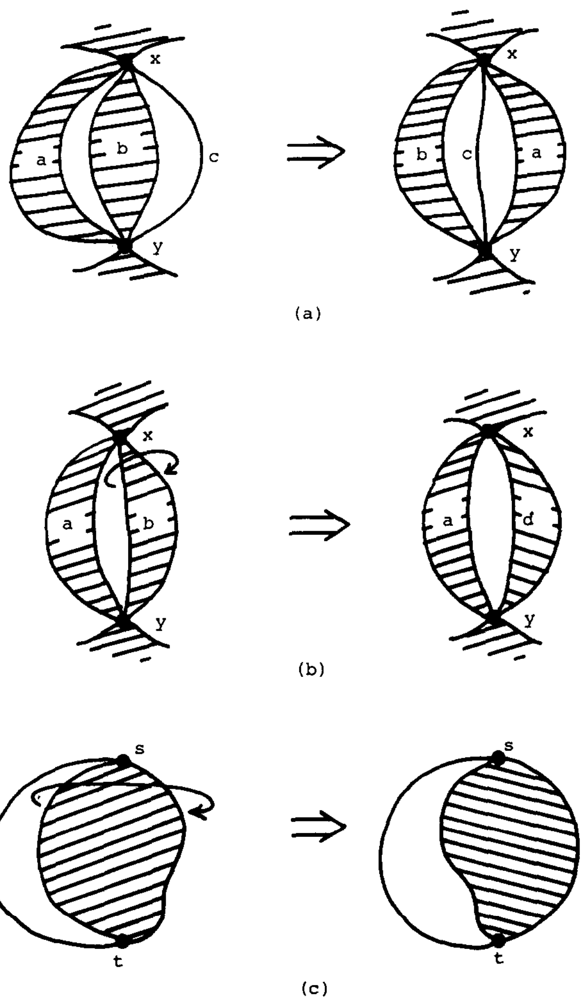


FIG. 10. Illustrations for operations (a) (i), (b) (ii), and (c) (iii).

Lemma 7 implies that any separation pair $\{x, y\}$ can be detected in the *PQ*-tree at the vertex addition step for y. If the reduction step for a vertex y results in a pertinent tree having a full node, y constitutes a separation pair with another vertex, and the pertinent tree represents split components. Therefore the alterations of an embedding due to the three operations (i), (ii), and (iii) above can be virtually realized by the following operations (i'), (ii'), and (iii') on a *PQ*-tree at some vertex-addition step:

(i') If a full *P*-node which is not the root is scanned at the vertex addition step for y, then permute the sons $u_1, u_2, ..., u_l$ of the *P*-node;

(ii') If, at the vertex addition step for t, $u_1, u_2, ..., u_l$ are the sons of the root with u_1 corresponding to the virtual edge (s, t), then permute $u_2, ..., u_l$; and

(iii') reverse a full *Q*-node.

The operation (i) corresponds to (i') or (ii'), and (ii) and (iii) correspond to (iii').

Moreover Lemma 7 implies that every neighbour v of y contained in an $\{x, y\}$-split component is necessarily a leaf of the *PQ*-tree at the vertex addition step for y. Therefore all these neighbours are contained in an upward adjacency list $A_u(y)$ constructed by UPWARD-EMBED. For a node w of a *PQ*-tree let $L(w)$ be the list containing all the leaves that are descendants of w. The operations (i'), (ii'), and (iii') for the *PQ*-tree can be realized by the following operations (a), (b), and (c) for the upward adjacency lists A_u:

(a) If a full *P*-node which is not the root is scanned at the vertex addition step for y, then permute the sublists $L(u_1), L(u_2), ..., L(u_l)$ of $A_u(y)$, where $u_1, u_2, ..., u_l$ are the sons of the *P*-node (see Fig. 11);

(b) If, at the vertex addition step for t, $u_1, u_2, ..., u_l$ are the sons of the root of a *PQ*-tree with u_1 corresponding to the virtual edge (s, t), then permute the sublists $L(u_2), ..., L(u_l)$ of $A_u(t)$; and

(c) If a full *Q*-node is scanned at the vertex addition step for y, then reverse the sublists of $A_u(y)$ consisting of $L(u_1), L(u_2), ..., L(u_l)$, where $u_1, u_2, ..., u_l$ are the sons of the full *Q*-node. (See Fig. 12.)

In an upward adjacency list $A_u(y)$, we use parentheses to represent the permutable sublist $L(u_1), L(u_2), ..., L(u_l)$ in (a) and $L(u_2), ..., L(u_l)$ in (b) as follows:

$$A_u(y) = ...(L(u_1), L(u_2), ..., L(u_l))...,$$

and

$$A_u(t) = ...(L(u_2), ..., L(u_l)),$$

respectively. On the other hand we use brackets to represent the reversible sublists $L(u_1), L(u_2), ..., L(u_l)$ in (c) as

$$A_u(y) = ...[L(u_1), L(u_2), ..., L(u_l)]....$$

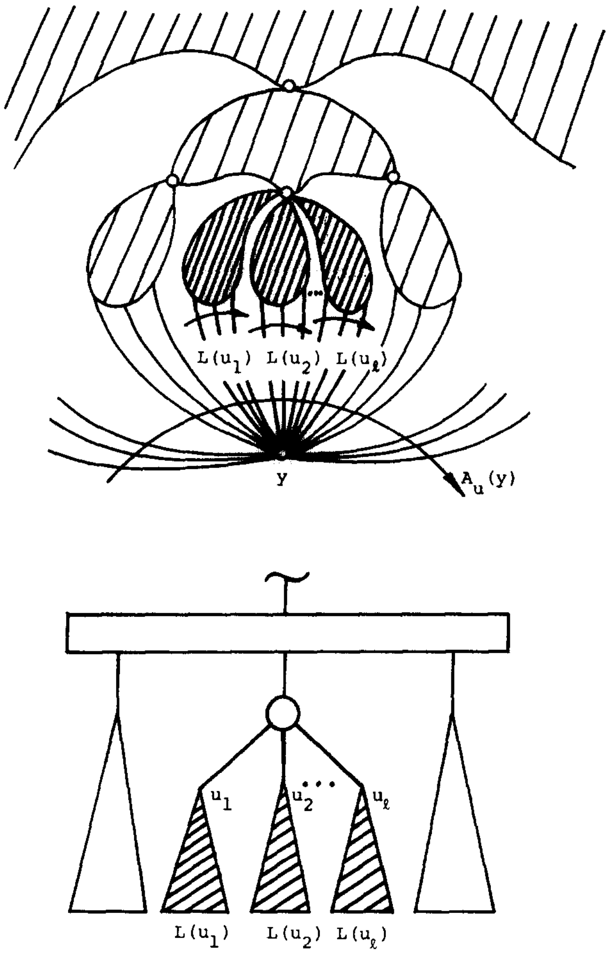


FIG. 11. Illustration for operation (a).

These expressions of $A_u(y)$ correspond to the formulae of [10]. From these A_u we can obtain any embedding of G by the following algorithm GENERATE. Thus A_u virtually represents all the embeddings of G.

```
procedure GENERATE;
  begin
    apply operations (a), (b), and (c) to the sublists of $A_u$ parenthesized or
    bracketed;
    obtain a correct upward embedding $A_u$ by applying the "correction step" of
    UPWARD-EMBED;
    ENTIRE-EMBED
  end;
```

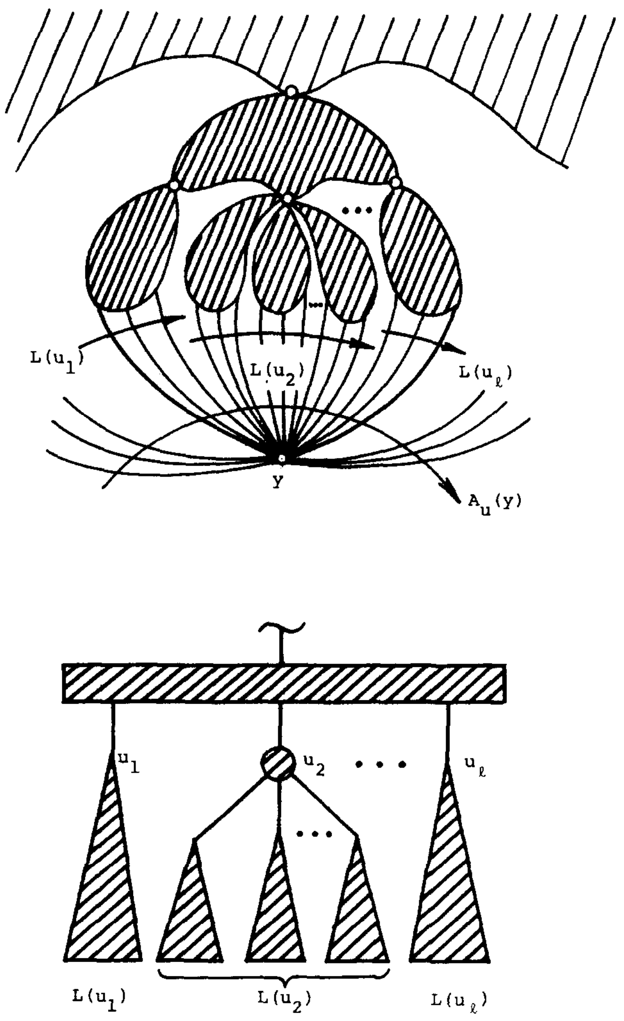


FIG. 12. Illustration for operation (c).

Using algorithm GENERATE one can generate all the embeddings of G without duplications. (The proof is left to the reader.) It is also easy to decide the total number of possible embeddings of G from the expression of A_u.

ACKNOWLEDGMENT

We wish to thank Professor Nobuji Saito of Tohoku University for stimulating discussion on the subject and also the referees for many helpful comments.

REFERENCES

1. L. AUSLANDER AND S. V. PARTER, On imbedding graphs in plane, *J. Math. Mech.* **11**, No. 3 (1961), 517–523.
2. K. S. BOOTH AND G. S. LUEKER, Testing the consecutive ones property, interval graphs, and graph planarity using *PQ*-tree algorithms, *J. Comput. System Sci.* **13** (1976), 335–379.
3. N. CHIBA, T. YAMANOUCHI, AND T. NISHIZEKI, Linear algorithms for convex drawings of planar graphs, *in* "Proceedings of Silver Jubilee Conference on Combinatorics," Academic Press, in press.
4. S. EVEN, "Graph Algorithms," Computer Sci. Press, Potomac, M., 1979.
5. S. EVEN AND R. E. TARJAN, Computing an *st*-numbering, *Theoret. Comput. Sci.* **2** (1976), 339–344.
6. A. J. GOLDSTEIN, "An Efficient and Constructive Algorithm for Testing Whether a Graph Can Be Embedded in a Plane," Graph and Combinatories Conf., Contract No. NONR 1858-(21), Office of Naval Research Logistics Proj., Dept. of Math., Princeton University, 1963.
7. F. HARARY, "Graph Theory" (revised), Addison-Wesley, Reading, Mass., 1972.
8. J. E. HOPCROFT AND R. E. TARJAN, Dividing a graph into triconnected components, *SIAM J. Comput.* **2**, No. 3 (1973), 135–158.
9. J. E. HOPCROFT AND R. E. TARJAN, Efficient planarity testing, *J. Assoc. Comput. Mach.* **21**, No. 4 (1974), 549–568.
10. A. LEMPEL, S. EVEN, AND I. CEDERBAUM, An algorithm for planarity testing of graphs, *in* "Theory of Graphs, Internat. Sympos. Rome, July 1966" (P. Rosenstiel, Ed.), pp. 215–232, Gordon & Breach, New York, 1967.