Planar Embedding: Linear-Time Algorithms for Vertex Placement and Edge Ordering

R. JAYAKUMAR, K. THULASIRAMAN, SENIOR MEMBER, IEEE, AND M. N. S. SWAMY, FELLOW, IEEE

Abstract—**The problem of obtaining a planar embedding of a biconnected planar graph is discussed. Our approach is based on the planarity testing algorithm of Lempel, Even, and Cederbaum [15] and its implementation using *PQ*-trees [20]. In the planar embedding, the vertices of the planar graph are placed in the plane at different horizontal and vertical levels such that no two distinct vertices appear in the same horizontal or vertical level, and higher numbered vertices appear at higher vertical levels. The left-to-right order of the vertices in such a planar embedding is called the *vertex order*. For each vertex i, the anticlockwise order in which edges enter i from lower numbered neighbors is denoted by $\tau(i)$. Linear-time algorithms to determine $\tau(i)$'s for all i, and the vertex order are developed. The vertex order captures the structural information about the relative placement of vertices in a planar embedding provided by the *PQ*-tree reduction algorithm. A systematic procedure to obtain an intersection-free drawing of the edges is described. A linear-time algorithm to construct a very compact horvert representation [9] of a planar graph is also presented. This algorithm, unlike those in [12], [13] does not require the construction of the dual of the original graph.**

I. INTRODUCTION

A GRAPH is *planar* if it can be drawn on a plane with no two edges crossing each other except at their end vertices. Testing a graph for planarity and embedding a planar graph in the plane have several applications. For example, the design of integrated circuits and the layout of printed circuit boards require testing whether a circuit can be embedded in the plane without edge crossings. Optimum single-row routing problem with no edge crossings involves planarity testing and planar embedding [1], [2]. Several cases of the routing problem have been shown to be equivalent to constructing planar embedding of special classes of graphs [3]. Determining isomorphism of chemical structures is simplified if the structures are known to be planar [4], [5]. A maximum cut in a graph can be determined efficiently if the graph is planar [6], whereas this problem is NP-complete for an arbitrary graph [7]. Because of the great practical interest, these two problems—planarity testing and planar embedding—have been extensively studied in the literature.

This paper is concerned with the problem of obtaining a planar embedding of a planar graph. One of the earliest algorithms to construct a planar embedding of a planar triconnected graph was proposed by Tutte [8]. In [9] Otten and van Wijk presented an algorithm to construct what is called the *horvert representation* of a planar graph. In this representation vertices are represented by horizontal straight-line segments and edges are drawn as vertical straight-line segments. More recent works on the planar embedding problem may be found in [10]–[13].

There are two efficient $O(n)$-time algorithms to test the planarity of a graph G with n vertices [14], [15]. These algorithms test G for planarity by trying to construct an embedding of G in the plane. Tarjan [16] has shown that his planarity testing algorithm can be used to construct a planar embedding and given the details of how to do this "by hand." Williamson [17] and Brehaut [18], [19] have discussed planar embedding algorithms based on the ideas of Hopcroft and Tarjan's planarity testing algorithm. In [10], Chiba *et al.* study the planar embedding problem using Lempel, Even, and Cederbaum's planarity testing algorithm, in short the LEC algorithm and its implementation based on *PQ*-trees [20].

In this paper we discuss the problem of obtaining a planar embedding of a planar graph using the LEC algorithm. To make our presentation self-contained, we briefly discuss the LEC algorithm in the next section. We describe in Section III the principle underlying our approach for the planar embedding problem and also summarize the main contributions to be presented in the subsequent sections.

II. LEMPEL, EVEN, AND CEDERBAUM'S PLANARITY TESTING ALGORITHM AND ITS IMPLEMENTATION USING *PQ*-TREES

Consider a simple biconnected graph $G=(V, E)$ with $n=|V|$ vertices and $m=|E|$ edges. The LEC algorithm first labels the vertices of G as $1,2,\cdots,n$ using what is called an *st-numbering* [21], [22]. The graph G is then called an *st*-graph. Usually in an *st*-graph, edge (i, j), $i<j$, is oriented from i to j. Also in an *st*-numbering,

Manuscript received June 9, 1987; revised October 22, 1987. This work was supported by the Natural Sciences and Engineering Research Council of Canada under Grant A0904, under Grant A4680, and under Grant A7739. Part of this work is from the Ph.D. dissertation of R. Jayakumar submitted to Concordia University, Montreal, PQ, Canada, in August 1984. This paper was recommended by K. Thulasiraman and N. Deo, Guest Editors for this issue.

The authors are with the Faculty of Engineering and Computer Science, Concordia University, Montreal, PQ, Canada H3G 1M8.

IEEE Log Number 8718928.

0098-4094/88/0300-0334$01.00 ©1988 IEEE

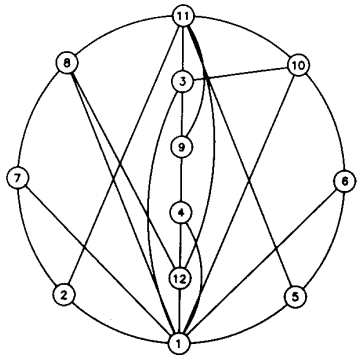


Fig. 1. *st*-graph G.

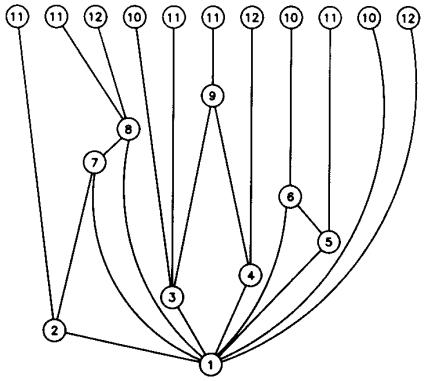


Fig. 2. Bush form B_9.

vertices 1 and n are adjacent and each vertex other than 1 and n is adjacent to at least one lower numbered vertex and to at least one higher numbered vertex. Let G_k, $1 \leqslant k \leqslant n$, denote the subgraph of G induced by the vertex set $V_k = \{1,2,\cdots,k\}$. We define the subgraph B_k as follows. G_k is a subgraph of B_k. In addition to G_k, B_k consists of all the edges of G which emanate from vertices of V_k and enter, in G, vertices of $V - V_k$. These edges are called *virtual edges* and the vertices they enter in $V - V_k$ are called *virtual vertices*. The virtual vertices are labeled as their counterparts in G; but kept separate. Thus in B_k there may be several vertices with the same label, each with exactly one entering edge.

Let $\hat{G}$ be a planar embedding of a planar *st*-graph G. It can be shown [23] that if $\hat{G}_k$ is a planar embedding of G_k, then all the edges and vertices of $\hat{G} - \hat{G}_k$ can be drawn on one face of $\hat{G}_k$. Thus we can assume, without loss of generality, that for a planar *st*-graph, there exists a planar embedding of B_k in which all the vertices are drawn on the outside face. Then we can draw B_k such that vertex 1 is at the lowest level; all the virtual vertices appear at the highest level on one horizontal line; and the remaining vertices of G_k are drawn such that vertices with higher labels are drawn higher. Such a realization of B_k is called a *bush form* of B_k. For example, the bush form B_9 of the *st*-graph G of Fig. 1 is shown in Fig. 2.

It has been proven [23] that the *st*-graph G is planar if and only if for every B_k, $2 \leqslant k \leqslant n-2$, there exists a B'_k isomorphic to B_k such that in B'_k all the virtual vertices labeled $k+1$ appear consecutively. Let v be a cut vertex in B_k. Then, as a consequence of *st*-numbering, v will be the lowest vertex in all the maximal biconnected components (except the component containing 1) with respect to v. These biconnected components, called *blocks*, will have the same structure as a bush form. Hence they are called *subbushes*. It can be shown that B'_k, whenever it exists, can be obtained by performing an appropriate sequence of flippings and/or permutations of the subbushes around cut vertices in B_k. Bush form B_{k+1} can then be formed by merging all the virtual vertices labeled $k+1$. Booth and Lueker [20] have shown that this could be done efficiently if each B_k is represented by a data structure called PQ-tree.

The PQ-tree T_k corresponding to the bush form B_k consists of three types of vertices.

(i) *Leaves* in T_k represent virtual vertices in B_k. A leaf has the same label as the virtual edge which enters the corresponding virtual vertex.

(ii) *P-nodes* in T_k represent cut vertices in B_k. A P-node is labeled as the cut vertex it represents. The leaf corresponding to the virtual edge (i, j), $i < j$, is a child of the P-node representing i.

(iii) *Q-nodes* of T_k represent the maximal biconnected components in B_k. Let $y_i, y_2, \cdots, y_q$ be the cut vertices (except the lowest vertex) appearing in that order on the outside window of a maximal biconnected component. Then this component is represented by a Q-node whose children are the P-nodes corresponding to $y_1, y_2, \cdots, y_q$. Furthermore, these children appear in the same left-to-right order as the order of the corresponding cut vertices on the outside window of the maximal biconnected component. The Q-node representing a biconnected component is a child of the P-node which represents the lowest cut vertex in this component.

From the above it can be seen that there is a natural correspondence between bush form B_k and the PQ-tree T_k. A few definitions are now in order. Let $S(k+1)$ denote the set of leaves in T_k which correspond to the virtual vertex $k+1$. A node X in T_k is said to be *full* if all its descendant leaves are in $S(k+1)$; *empty* if none of its descendant leaves are in $S(k+1)$; otherwise, X is *partial*. If X is full or partial, then it is called a *pertinent node*. The *frontier* of T_k is the sequence of all the leaves read from left to right. The *pertinent subtree* of T_k with respect to $S(k+1)$ is the subtree of minimum height whose frontier contains all the leaves in $S(k+1)$. The root of the pertinent subtree is called the *pertinent root*. For example, the PQ-tree T_9 corresponding to the bush form B_9 of graph G is shown in Fig. 3.

Two PQ-trees are considered equivalent if one can be obtained from the other by performing one or more of the following types of operations.

(i) Reversing the order of the children of a Q-node.
(ii) Permuting the children of a P-node.

It can be shown that B'_k exists if and only if T_k can be converted into an equivalent PQ-tree T'_k such that all the

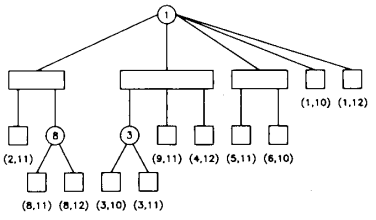


Fig. 3. PQ-tree T_9 corresponding to B_9.

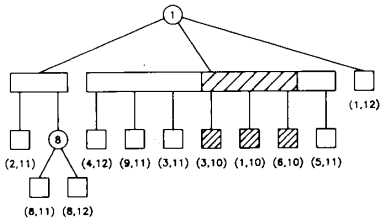


Fig. 4. PQ-tree T_9*.

pertinent leaves appear consecutively in the frontier of T_k'. Booth and Lueker have defined a set of patterns and replacements using which T_k can be reduced into a PQ-tree T_k* in which all the pertinent leaves appear as children of a single node. Such a T_k* can be constructed whenever T_k' exists. PQ-tree T_{k+1} can be constructed from T_k* by replacing all the leaves corresponding to the virtual vertex $k+1$ by a P-node whose children are the leaves corresponding to the virtual edges which enter vertices higher than $k+1$ in G. For example, for the PQ-tree T_9 of Fig. 3, T_9* and T_{10} are shown in Figs. 4 and 5, respectively.

The LEC algorithm starts with T_1 and constructs the sequence of PQ-trees $T_1, T_2, \cdots$. If the graph G is planar, then the algorithm terminates after constructing T_{n-1}; otherwise, it terminates after detecting the impossibility of reducing some T_k into T_k'. The crucial result in the complexity analysis of the LEC algorithm is stated in the following theorem [20].

Theorem 1:

The sum of all the pertinent nodes in the PQ-trees $T_1, T_2, \cdots, T_{n-1}$ of a planar graph is $O(m+n)$. ■

III. Bush Forms and τ'-Order

In this section we first discuss the principle underlying our procedure for drawing a planar embedding of a planar graph G using the different bush forms constructed by the LEC algorithm. We assume, without loss of generality, that every vertex in G is of degree at least three. For any vertex $i, 2 \leqslant i \leqslant n$, let $\Gamma^+(i)$ be the set of lower numbered neighbors of i. Let $B_1 = B_1', B_2, B_2', \cdots, B_i, B_i', \cdots, B_{n-1}$ be the sequence of bush forms generated by the LEC algorithm. Consider now the virtual edges entering vertex i in B_{i-1}'. The left-to-right order of these edges imposes an anticlockwise order around i among the vertices in $\Gamma^+(i)$. We call this order as the τ'-order in B_i for vertex $i, \tau'(i)$. In general, τ-order of vertex i in a planar embedding of G will refer to the anticlockwise order around i of the edges

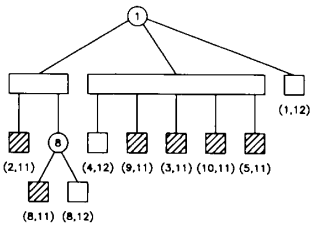
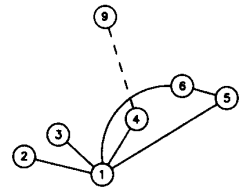


Fig. 5. PQ-tree T_{10}.

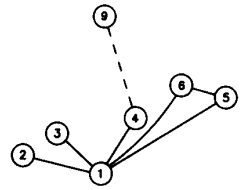
entering i from lower numbered vertices as well as to the corresponding order of the lower numbered vertices. In the PQ-tree T_{i-1}*, the pertinent leaves corresponding to the virtual edges entering vertex i in B_{i-1}' appear consecutively as children of a single node in the same left-to-right order as the virtual edges. Also, the pertinent root is a Q-node provided $|\tau'(i)| > 1$. So, if $(v_1, i), (v_2, i), \cdots, (v_j, i)$, $j \geqslant 1$, is the left-to-right order of these pertinent leaves in T_{i-1}*, then $\tau'(i) = (v_1, v_2, \cdots, v_j)$. Thus $\tau'(i)$ for each i can be constructed from the corresponding T_{i-1}*. For example, from the PQ-tree T_9* shown in Fig. 4, we get $\tau'(10) = (3,1,6)$.

In T_{n-1}*, the leaf corresponding to the virtual edge $(1, n)$ is a child of the P-node labeled 1. Since each vertex of G has degree at least three, all the other children of this P-node are Q-nodes. These Q-nodes can be merged into a single Q-node because all of them are full Q-nodes. The order of all the edges incident into vertex n, except the edge $(1, n)$, is determined by the left-to-right order of their appearance as children of this new Q-node. Let (v_1, n), $(v_2, n), \cdots, (v_k, n)$, $k \geqslant 1$, be this order. The edge $(1, n)$ has the freedom to appear either on the left or on the right of this sequence of edges. Moreover, vertex 1 will appear in the τ'-order of some other vertex less than n. So we omit vertex 1 from $\tau'(n)$, and thus $\tau'(n) = (v_1, v_2, \cdots, v_k)$.

Note that during the bush growing process blocks undergo flippings. The τ'-order for a vertex i gets reversed whenever a block containing i is flipped while growing a bush B_k, $k > i$. Thus the τ-order of vertex i in the final embedding of G may not be the same as $\tau'(i)$. In Section IV we develop an algorithm to obtain the τ-orders for all the vertices in the final planar embedding of G. An alternate algorithm to determine τ-orders may be found in [10]. In the final embedding we shall place the vertices at different vertical levels. If we assume that no two distinct vertices of G appear on the same horizontal level, then by scanning such an embedding left-to-right we can obtain a horizontal order of the vertices of G. Let us call this horizontal order the *vertex order*. In general, the vertex order is not unique. We impose a property on the vertex order which reflects the information provided by the PQ-tree reduction process as to the relative placement of the vertices. In Section V we develop an $O(n)$ algorithm to obtain such a vertex order from the τ-orders of the vertices in the final planar embedding of G. While $\tau(i)$ specifies the anticlockwise order around vertex i in which the edges entering vertex i should be drawn, unfortunately, this information alone is not sufficient to construct an intersec-



(a)



(b)

Fig. 6. (a) A drawing with intersection. (b) An intersection-free drawing.

tion-free drawing. For example, consider Fig. 6(a). Here $\tau(6) = (1,5)$. So the edges (6,1) and (6,5) have to be drawn in that order. If these edges were drawn as shown in Fig. 6(a), then when vertex 9 is embedded at a later time, there would be no way to draw the edge (9,4) without intersecting some of the edges already drawn. To avoid this problem, we should have drawn the edges (6,1) and (6,5) as shown in Fig. 6(b). This example shows that to obtain an intersection-free drawing, the edges should also be drawn in an appropriate way if we wish to avoid redrawing any of the edges already drawn. In Section V we study this question further and present a procedure to draw the edges of G. Finally, in Section VI, we present an $O(n)$ algorithm to construct a very compact horvert representation of a planar graph. This algorithm is quite simple and uses only the τ-orders of the vertices. Unlike those in [12], [13], it does not require construction of the dual of the given planar graph.

IV. Block Graph and τ-Order

In order to determine the τ-orders of the vertices in the final planar embedding of a planar graph G, we first discuss how the blocks are formed during the bush growing process. Consider the bush form B'_{i-1}. The virtual edges entering vertex i emerge from vertices on the outside window of the maximal biconnected components, or blocks, of B_{i-1}, say $C_{i(1)}$, $C_{i(2)}, \cdots$, and $C_{i(k)}$. When B_i is constructed by merging these virtual edges, $C_{i(1)}, C_{i(2)}, \cdots, C_{i(k)}$ merge to form a new block. We denote this block by C_i, indicating that i is the highest vertex in this block. The blocks $C_{i(1)}, C_{i(2)}, \cdots, C_{i(k)}$ are precisely those represented by the Q-nodes in the pertinent subtree of T_{i-1}. They are considered to be *enclosed* by C_i. For example, the block C_{10} of the graph in Fig. 1 is obtained by merging C_6, C_9 and the trivial block containing only the vertex 1. Thus C_{10} encloses C_6 and C_9 and C_1. It is easy to see that $C_{i(1)}, C_{i(2)}, \cdots, C_{i(k)}$ will not be blocks in the bush forms $B_i, B_{i+1}, \cdots, B_{n-1}$.

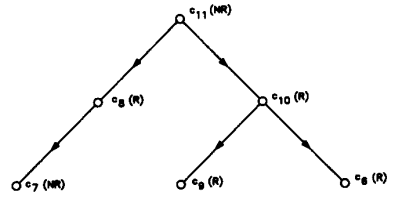


Fig. 7. Block graph.

While growing the bushes, C_i may be involved in several permutations and flippings. Permutations do not affect the τ'-order of any vertex in C_i. On the other hand, flipping C_i reverses the τ'-order of i. Furthermore, if C_i encloses C_j, then the τ'-order of j will also be reversed whenever C_i is flipped. Since our interest is to determine the τ-order of each vertex in the final embedding of G, we need to determine the status of each block, namely, reversed or not, in the final embedding. Let $\tau(i)$ denote the τ-order of vertex i in the final embedding of G. If $\tau'_{\text{rev}}(i)$ denotes the list obtained by reversing the list $\tau'(i)$, then $\tau(i)$ is equal to either $\tau'(i)$ or $\tau'_{\text{rev}}(i)$.

To develop an efficient algorithm for determining the status of each block in the final embedding, we construct a labelled directed graph called the *block graph*. If C_i is a block with only one edge then $\tau'(i)$ will have only one vertex in it. As a result, flipping C_i will have no effect on $\tau'(i)$ and so $\tau(i) = \tau'(i)$. So, in our discussion we will consider only those blocks which have at least three edges. Such blocks will be referred to as non-trivial blocks. In the block graph vertices represent non-trivial blocks and edges represent the enclosing relation among them. We denote the vertex representing block C_i as c_i, and with each vertex we associate a label. The label of vertex c_i is R if block C_i is reversed when the first block enclosing C_i is formed; otherwise the label is NR. If block C_i encloses the blocks $C_{i(1)}, C_{i(2)}, \cdots, C_{i(k)}$, then in the block graph we draw edges directed from vertex c_i to each one of the vertices $c_{i(1)}, c_{i(2)}, \cdots, c_{i(k)}$. The blocks enclosed by C_i can be easily identified during the PQ-tree reduction process, since the corresponding Q-nodes, $Q_{i(1)}, Q_{i(2)}, \cdots, Q_{i(k)}$, are all present in the pertinent subtree of T_{i-1}. Note that during reduction $(i-1)$, when we transform T_{i-1} to T^*_{i-1}, these nodes are processed and merged into a single Q-node representing C_i. As an example, the block graph of the planar st-graph G shown in Fig. 1 is given in Fig. 7.

To determine the label of each vertex in the block graph, consider the reduction $(i-1)$. Since the label of each of the blocks which C_i encloses indicates the status of that block, namely, whether the block is reversed or not, when C_i is formed, it is necessary that we keep track of the flippings which the blocks undergo as reduction $(i-1)$ progresses. For this purpose, we construct what we call the ith *intermediate block graph* which is denoted by IBG(i).

To start with, we add to IBG(i) vertices to correspond to the Q-nodes in the pertinent subtree of T_{i-1} and associate with each one of these vertices the label NR. Suppose a node, say X, in T_{i-1} is being processed and that $Q_j, Q_k, \cdots, Q_l$ are the Q-nodes which are pertinent children

of X. First consider the case that X is a P-node. Let Q_X be the Q-node created after the processing of X is completed. Then we add to IBG(i) a vertex, say q_X, to correspond to Q_X and an edge directed from q_X to each one of the vertices representing $Q_j, Q_k, \cdots, Q_l$. The label of q_X is NR. On the other hand, if X is a Q-node, then IBG(i) will contain a vertex, say q_X, corresponding to X. In this case, as before, we add to IBG(i) an edge directed from q_X to each one of the vertices corresponding to $Q_j, Q_k, \cdots, Q_l$. If any of the nodes $Q_j, Q_k, \cdots, Q_l$ is reversed while processing X, then we change the label of the corresponding vertex in IBG(i) to R. Note that IBG(i) is essentially a directed tree in which each leaf corresponds to a Q-node in T_{i-1} representing a block of G_{i-1}. The root of IBG(i) will represent the block C_i and its label is NR.

The labels of the vertices in the block graph representing the blocks $C_{i(1)}, C_{i(2)}, \cdots, C_{i(k)}$ enclosed by C_i can be determined as follows. We traverse IBG(i) depth-first starting at its root. Suppose, during this traversal, we are at vertex y. If the label of y in IBG(i) is R, then we *switch* the labels of all the children of y in IBG(i). (By switching the label we mean setting the label to R if its current value is NR, or setting the label to NR if its current value is R.) At the end of the traversal of IBG(i), we can determine from the vertex labels whether a block enclosed by C_i is flipped in the embedding of C_i. These labels are then given to the appropriate vertices in the block graph.

In the following, FIND-LABEL-IBG(i) will refer to the procedure which constructs IBG(i) and determines the labels (in the block graph) of the vertices representing the blocks enclosed by C_i.

Theorem 2:

Cost of procedure FIND-LABEL-IBG(i) is $O(N_i)$, where N_i is the number of pertinent nodes in T_{i-1}.

Proof: It can be seen that the number of leaves in IBG(i) is no more than N_i, the number of pertinent nodes in T_{i-1}. Note that IBG(i) is a directed tree and each internal vertex of this tree has out-degree at least 2 because we consider only non-trivial blocks. This means that IBG(i) has $O(N_i)$ edges. So the cost of constructing IBG(i) and the cost of traversal of IBG(i) to determine the labels of its vertices are both $O(N_i)$. The theorem follows since the procedure FIND-LABEL-IBG(i) involves only these two costs. ■

We now give a formal presentation of our procedure to construct the block graph. In this procedure, the labels of the vertices of the block graph are stored in the array STATUS.

```
procedure BLOCK-GRAPH;
comment procedure BLOCK-GRAPH constructs the block
        graph and stores the status information of each
        block during the PQ-tree reduction process.
        STATUS(i) represents the status of block C_i in
        the block graph.

begin
  for i := 2 to n-1 do
  begin
    {Construct the block graph and determine the status
    of the blocks}
    FIND-LABEL-IBG(i);
    for each pertinent Q-node Q_j in T_{i-1} do
      begin
        draw a directed edge from c_i to c_j;
        STATUS (j) := label of Q_j {as obtained by
        FIND-LABEL-IBG(i)}
      end;
    obtain T_i;
    {The Q-node which is the pertinent root in T*_{i-1}
    represents block C_i}
    STATUS (i) := NR
  end
end BLOCK-GRAPH;
```

In Fig. 7 we have shown within parentheses the label of each vertex in the block graph for the planar graph of Fig. 1.

Theorem 3:

Procedure BLOCK-GRAPH correctly constructs the block graph and determines the status information of each block in $O(n)$ time.

Proof: Correctness of the procedure follows from our discussion so far. To find the complexity, note that the cost of procedure BLOCK-GRAPH during reduction $(i-1)$, exclusive of the cost for procedure FIND-LABEL-IBG(i), is proportional to the number of blocks enclosed by C_i. From Theorem 2 the cost of procedure FIND-LABEL-IBG(i) is proportional to the number of pertinent nodes in T_{i-1}. Hence the overall cost of procedure BLOCK-GRAPH is proportional to the number of pertinent nodes in all T_i's. Thus from Theorem 1 the complexity of procedure BLOCK-GRAPH is $O(m+n)$ which is $O(n)$ for a planar graph. ■

Having obtained the block graph and the status of each block in it, we now need to find whether a block will be reversed in the final embedding of G or not. This will determine $\tau(i)$ for each vertex i. Note that block C_n will not be present in the block graph because it is not processed during any reduction. Also blocks C_i, $2 \leqslant i \leqslant n-1$ will be present in the block graph if and only if $|\tau'(i)| > 1$. As in the procedure FIND-LABEL-IBG(i), we determine the τ-orders by traversing the block graph in a depth-first way. Suppose we are at a vertex, say c_i, of the block graph. If the status of the block C_i is R, then all the blocks enclosed by C_i require flippings. These blocks are represented in the block graph by the children of c_i and so we update their status by switching their labels. No updating of the labels is required if the status of C_i is NR. If the final status of C_i is NR then $\tau(i) = \tau'(i)$; otherwise, $\tau(i) = \tau'_{\text{rev}}(i)$. As an example, in Fig. 8, we give the final status of each block using the block graph of Fig. 7 and the τ-orders of the vertices in the final embedding of G.

We shall denote by FIND-STATUS the procedure which determines the status of each block in the final

Block	Status	τ'-order	τ-order
C_{12}	NR	(8,11,4)	(8,11,4)
C_{11}	NR	(8,2,5,10,3,9)	(8,2,5,10,3,9)
C_{10}	R	(3,1,6)	(6,1,3)
C_9	P NR	(3,4)	(3,4)
C_8	R	(7,1)	(1,7)
C_7	P R	(2,1)	(1,2)
C_6	P NR	(5,1)	(5,1)
C_5	--	(1)	(1)
C_4	--	(1)	(1)
C_3	--	(1)	(1)
C_2	--	(1)	(1)

Fig. 8. τ-orders obtained from status information.

embedding of G and hence $\tau(i)$, $2 \leqslant i \leqslant n$. The following theorem is easy to prove.

Theorem 4:

Procedure FIND-STATUS determines $\tau(i)$, $2 \leqslant i \leqslant n$, correctly in $O(n)$ time. ■

It can be easily seen that procedure BLOCK-GRAPH can be implemented along with the PQ-tree reduction procedure. From Theorems 2, 3, and 4, we conclude that the τ-orders in the final planar embedding of a planar graph can be constructed in $O(n)$ time.

V. Vertex Order and Planar Embedding

We now turn our attention to the problem of constructing a planar embedding of G using the τ-orders of the vertices in the final embedding. The embedding scheme envisaged in Section III places the vertices of G in the plane at different horizontal and vertical levels such that no two distinct vertices are placed in the same vertical or horizontal levels. Recall that the left-to-right order of the vertices of G in such a placement is called the vertex order, denoted by μ. Since the vertex order in general is not unique, we shall impose on μ a property which reflects the information provided by the PQ-tree reduction process as to the placement of vertices from $\tau(i)$ relative to the position of vertex i. First we develop an algorithm to determine such a vertex order and then discuss a method to draw a planar embedding of G.

We propose to construct an embedding of G by embedding the vertices $2, 3, \cdots, n$ in that order. By "embedding vertex i" we mean connecting i to its lower numbered neighbors using the order specified by $\tau(i)$. Thus when vertex i is to be embedded, the lower numbered vertices $1, 2, \cdots, i-1$ are already embedded. Some of these embedded vertices may be adjacent to vertices greater than i in G. We shall call these vertices as *Type 2* vertices relative to i. All the other vertices will be called *Type 1* vertices relative to i. In the following we shall refer to these vertices as simply Type 2 and Type 1 vertices, respectively, if the context makes it clear that they indeed have these properties relative to vertex i.

We represent the vertex order μ as a doubly linked list.

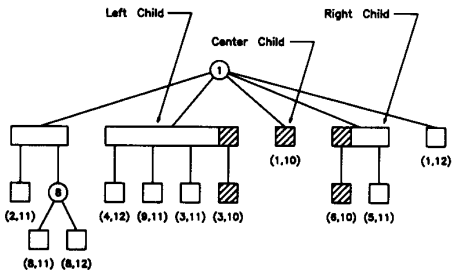


Fig. 9. PQ-tree T_9''. $\tau'_L(10) = (3)$, $\tau'_C(10) = (1)$, $\tau'_R(10) = (6)$.

To start with μ contains the vertex n and we add the vertices in $\tau(n), \tau(n-1), \cdots, \tau(2)$ to μ in that order. Whenever a vertex is placed in μ, we store the address of the element in μ corresponding to that vertex so that we can access any vertex in μ in constant time. When we add the vertices in $\tau(i)$ to μ, vertex i should be already present in μ since i should be in $\tau(j)$ for some $j > i$. Moreover, at this stage all the Type 2 vertices in $\tau(i)$ will also be present in μ. So we need to add to μ only the Type 1 vertices from $\tau(i)$. Note that we can check whether a vertex is Type 2 or not by simply testing for its presence in μ.

Consider reduction $(i-1)$ in which the PQ-tree T_{i-1} is transformed into the PQ-tree T_{i-1}^*. When the pertinent root in T_{i-1} is processed, it can have at most two partial children (which are partial Q-nodes) but any number of full children (some of which may be pertinent leaves). Just before the pertinent children of the pertinent root are merged to obtain T_{i-1}^*, one of the partial children should have its full children at its right end and the other should have its full children at its left end. We shall call these partial children as the *Left Child* and the *Right Child*, respectively. All the other pertinent children of the pertinent root will be called *Center Children*. Thus all the Center Children will be full. For example, for the planar graph G of Fig. 1, we have shown in Fig. 9 the PQ-tree T_9'' at the time the pertinent root of the PQ-tree T_9 is being processed. In this figure we have indicated the Left Child, Right Child, and the Center Child of the pertinent root.

It is easy to see that in $\tau'(i)$ the vertices corresponding to the pertinent leaves of the Left Child should appear consecutively and we denote this portion of $\tau'(i)$ as $\tau'_L(i)$. Similarly, the vertices in $\tau'(i)$ corresponding to the Center Children and Right Child should appear consecutively and we denote these portions of $\tau'(i)$ as $\tau'_C(i)$ and $\tau'_R(i)$, respectively. Thus $\tau'(i) = (\tau'_L(i), \tau'_C(i), \tau'_R(i))$ and at least one of $\tau'_L(i)$, $\tau'_C(i)$ and $\tau'_R(i)$ is not empty for every i, $2 \leqslant i \leqslant n$. Note that $\tau'_L(i)$, $\tau'_C(i)$ and $\tau'_R(i)$, $2 \leqslant i \leqslant n$, can easily be obtained during PQ-tree reduction without increasing the computational complexity of the reduction procedure. Furthermore, if $\tau'(i)$ is reversed to obtain the final τ-order $\tau(i)$, then $\tau_L(i)$, $\tau_C(i)$, and $\tau_R(i)$ will simply be the reversals of $\tau'_R(i)$, $\tau'_C(i)$ and $\tau'_L(i)$, respectively. Hence, $\tau_L(i)$, $\tau_C(i)$, and $\tau_R(i)$ can be obtained in $O(n)$ time using the algorithm discussed in Section IV. In our example, since the block C_{10} is reversed in the final

Vertex i	$\tau(i)$	$\tau_L(i)$	$\tau_C(i)$	$\tau_R(i)$
12	(8,11,4)	--	(8,11,4)	--
11	(8,2,5,10,3,9)	(8,2)	--	(5,10,3,9)
10	(6,1,3)	(6)	(1)	(3)
9	(3,4)	(3)	--	(4)
8	(1,7)	--	(1)	(7)
7	(1,2)	--	(1)	(2)
6	(5,1)	(5)	(1)	--
5	(1)	--	(1)	--
4	(1)	--	(1)	--
3	(1)	--	(1)	--
2	(1)	--	(1)	--

Fig. 10. τ_L, τ_C, and τ_R orders.

$\tau(i)$ placed in μ	Vertex order μ
Initial	12
$\tau(12)$	8,12,11,4
$\tau(11)$	8,12,2,11,5,10,3,9,4
$\tau(10)$	8,12,2,11,5,6,10,1,3,9,4
$\tau(9)$	8,12,2,11,5,6,10,1,3,9,4
$\tau(8)$	8,7,12,2,11,5,6,10,1,3,9,4
$\tau(7)$	8,7,12,2,11,5,6,10,1,3,9,4
$\tau(6)$	8,7,12,2,11,5,6,10,1,3,9,4
$\tau(5)$	8,7,12,2,11,5,6,10,1,3,9,4
$\tau(4)$	8,7,12,2,11,5,6,10,1,3,9,4
$\tau(3)$	8,7,12,2,11,5,6,10,1,3,9,4
$\tau(2)$	8,7,12,2,11,5,6,10,1,3,9,4

Fig. 11. Finding vertex order.

embedding of G, $\tau_L(10) = (6)$, $\tau_C(10) = (1)$ and $\tau_R(10) = (3)$. In Fig. 10 we show $\tau_L(i)$, $\tau_C(i)$ and $\tau_R(i)$ for all the vertices i of the planar graph shown in Fig. 1.

The orders $\tau_L(i)$, $\tau_R(i)$, and $\tau_C(i)$ for each i give a natural placement of the vertices of $\tau(i)$ relative to the position of i. Thus we want to construct μ such that for any vertex i, $2 \leqslant i \leqslant n$, all the vertices in $\tau_L(i)$ will appear to the left of i in μ, and all the vertices in $\tau_R(i)$ will appear to the right of i in μ. If the vertices are placed according to such a μ, then in the final embedding the blocks containing the vertices in $\tau_L(i)$ will be on the left side of i and those containing the vertices in $\tau_R(i)$ will be on the right side of i. A vertex order with this property would aid us in obtaining an elegant planar embedding.

To construct such a μ, we place the Type 1 vertices from $\tau_L(i)$ to the immediate left of vertex i, and the Type 1 vertices from $\tau_R(i)$ to the immediate right of i. After placing the vertices from $\tau_L(i)$ and $\tau_R(i)$, we place the Type 1 vertices from $\tau_C(i)$ around vertex i in μ. We split these Type 1 vertices into two halves and place the first half to the left of vertex i and the second half to the right of vertex i in μ such that the left-to-right order of these vertices in μ is the same as in $\tau_C(i)$.

Thus we can obtain the vertex order μ using the following procedure VERTEX-ORDER.

procedure VERTEX–ORDER;
comment procedure VERTEX–ORDER determines the vertex order μ from

$$\tau(i) = (\tau_L(i), \tau_C(i), \tau_R(i)), \qquad 2 \leqslant i \leqslant n.$$

begin
initialize μ to contain the vertex n;
for $i := n$ **downto** 2 **do**
begin
if $\tau_L(i)$ is not empty
then place in μ the Type 1 vertices from $\tau_L(i)$ to the left of vertex i;
if $\tau_R(i)$ is not empty
then place in μ the Type 1 vertices from $\tau_R(i)$ to the right of vertex i;
if $\tau_C(i)$ is not empty
then place in μ the vertices from $\tau_C(i)$ around vertex i such that the left-to-right order of these vertices in μ is the same as in $\tau_C(i)$
end
end VERTEX–ORDER;

We illustrate in Fig. 11 the above procedure to find the vertex order for the graph of Fig. 1. In this figure we show the progressive growth of the vertex order as we add the vertices in $\tau(i)$, $n \geqslant i \geqslant 2$.

We now prove that the vertex order constructed as above has the desired property.

Theorem 5:

In the vertex order constructed by procedure VERTEX–ORDER, the vertices in $\tau_L(i)$ will appear to the left of vertex i for any i, $2 \leqslant i \leqslant n$, and the vertices in $\tau_R(i)$ will appear to the right of vertex i.

Proof: Note that procedure VERTEX–ORDER places all the Type 1 vertices from $\tau_L(i)$ to the left of i in the vertex order μ in the same left-to-right order as in $\tau_L(i)$, and also places all the Type 1 vertices from $\tau_R(i)$ to the right of i in μ in the same left-to-right order as in $\tau_R(i)$. So we need only to prove that all the Type 2 vertices in $\tau_L(i)$ will appear to the left of i in μ and all such vertices in $\tau_R(i)$ will appear to the right of i in μ.

For any vertex v, let first(v) be the highest numbered neighbor of v. This means that v is in τ(first(v)) and it is placed in μ when we add the vertices from τ(first(v)). Also v is a Type 1 vertex in τ(first(v)). Hence procedure VERTEX–ORDER will place v around first(v) and no Type 2 vertex in τ(first(v)) will appear between v and first(v) in μ.

Now let j be a Type 2 vertex in $\tau_L(i)$. From the PQ-tree reduction procedure it should be clear that in T_i, the node corresponding to vertex j will appear to the left of the node corresponding to vertex i. Let i, first(i), first(first(i)),$\cdots$, x and j, first(j), first(first(j)),$\cdots$, y be the sequences of vertices such that first(x) = first(y) = k. Suppose we carry out the PQ-tree reduction procedure making sure that at each step the Q-nodes representing the different blocks of a bush form give rise to the τ-orders in the final embedding, then no reversal of these nodes at any

step will be required. So, in such T_{k-1}, the nodes corresponding to the vertices x and y should appear as children of a Q-node with the node corresponding to vertex y appearing to the left of the node corresponding to vertex x. Since both x and y are Type 1 vertices in $\tau(k)$, procedure VERTEX-ORDER will place y to the left of x in μ. This along with the fact that any vertex in the sequence i, first(i), first(first(i)),$\cdots$, x and in the sequence j, first(j), first(first(j)),$\cdots$, y is placed around its successor in the sequence in μ implies that j will be placed to the left of i in μ. Thus all the Type 2 vertices from $\tau_L(i)$ will be placed to the left of vertex i in μ.

Similarly we can prove that all the Type 2 vertices from $\tau_R(i)$ will be placed to the right of vertex i in μ. ■

The following theorem establishes the complexity of procedure VERTEX-ORDER.

Theorem 6:

Procedure VERTEX-ORDER determines the vertex order in $O(n)$ time.

Proof: It is easy to see that for a given i, the cost of placing in μ the Type 1 vertices from $\tau(i)$ is at most $|\tau_L(i)|+|\tau_C(i)|+|\tau_R(i)|$, which is the in-degree of vertex i in the st-graph G. Summing up these costs over all i, $2 \leqslant i \leqslant n$, we get the cost of computing the vertex order as $O(n)$ for a planar graph. ■

Having obtained the vertex order, we now describe our drawing procedure to obtain a planar embedding. We place the vertices of G in the plane at different horizontal and vertical levels. In the following, the horizontal line at vertical level r will be denoted by X_r and the vertical line at horizontal level r will be denoted by Y_r. Whereas the vertical level of a vertex in the placement is dictated by its st-number, the horizontal level is dictated by the position of the vertex in the vertex order μ. In such a placement no two vertices will appear in the same horizontal or vertical level. We then construct a planar embedding of G by constructing planar embeddings of the vertex induced subgraphs $G_2, G_3, \cdots, G_n = G$, successively. At each step of the embedding process, we have to ensure that the corresponding Type 2 vertices appear on the outside window. Clearly, this requirement is satisfied by G_2.

Suppose we have embedded G_{i-1} such that all the vertices connected to vertices numbered i or higher are on the outside window of G_{i-1}. When we embed vertex i, clearly it will appear on the outside window of G_i. However, the edges connecting i to vertices in $\tau(i)$ should be drawn so that in G_i all the Type 2 vertices appear on the outside window. Let $\tau(i) = (j_1, j_2, \cdots, j_k)$. Connecting vertex i to the vertices j_1 and j_k forms a circuit, say χ_i, in G_i. In addition to the edges (j_1, i) and (j_k, i), this circuit will contain the path from j_1 to j_k traced along the outside window of G_{i-1}.

Let R_1 denote the region bounded by the lines X_{j_1}, X_i, Y_{j_1}, and Y_i and R_2 denote the region bounded by the lines X_{j_k}, X_i, Y_{j_k}, and Y_i. Let P_i denote the path from j_1 to j_k traced along the outside window of G_{i-1}. We now show that the edges (i, j_1) and (i, j_k) can be drawn in the regions R_1 and R_2, respectively, so that the circuit χ_i does not enclose any Type 2 vertex which lie in these regions. Our arguments are based on the facts that

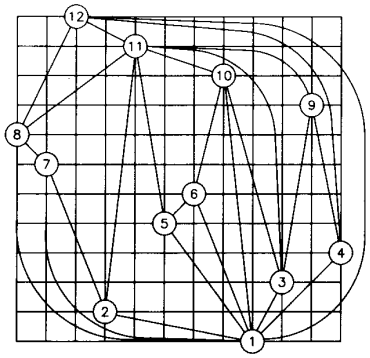


Fig. 12. Planar embedding of the graph in Fig. 1.

(i) in the vertex order μ constructed by procedure VERTEX-ORDER, no Type 2 vertex appears between vertex i and a Type 1 vertex,

(ii) in μ, all the vertices from $\tau_L(i)$ lie to the left of i and those from $\tau_R(i)$ lie to the right of i,

(iii) $\tau(i)$ can have at most two Type 2 vertices from each block of G_{i-1}, and

(iv) when the pertinent root of T_{i-1} is processed, it can have at most two partial children.

Suppose j_1 and j_k are both Type 1 vertices. Then the edges (i, j_1) and (i, j_k) can be drawn as desired in R_1 and R_2, respectively, because, as mentioned above, these regions will contain no Type 2 vertices.

Consider then the case when one of j_1 and j_k, say j_1, is a Type 2 vertex. Suppose R_1 contains a Type 2 vertex x and it is not possible to draw (i, j_1) in R_1 to the right of x. We may assume that x is not on the path P_i. Then x must be lying on the outside window of a block of G_{i-1} which intersects with R_1. This block may be the same as the one containing j_k or it may not. In either case, we can draw the edge (i, j_k) in R_2 to the left of x and the edge (i, j_1) in the region $R_1 - R_2$ without the circuit χ_i enclosing any Type 2 vertices from these regions.

Since every other Type 2 vertex which does not lie in R_1 or R_2 will necessarily be on the circuit χ_i or outside of it, we have the following theorem.

Theorem 7:

The edges (i, j_1) and (i, j_k) can be drawn in the regions R_1 or R_2 so that the circuit χ_i encloses no Type 2 vertices. ■

Thus to embed vertex i, we first draw the edge (j_1, i) within the region R_1 such that this edge lies to the right of all the Type 2 vertices from $\tau(i)$ which lie in $R_1 - R_2$. Next we draw the edges (j_2, i), (j_3, i),$\cdots$,(j_k, i) entering vertex i from below in such a way that any edge enters vertex i to the immediate right of its predecessor in the sequence. Note that the edge (j_k, i) has to be drawn so that this edge lies to the left of all the Type 2 vertices from $\tau(i)$ placed in the region R_2. Embedding vertex i this way we obtain a planar embedding of G_i. Repeating this procedure we can obtain planar embeddings of G_{i+1},

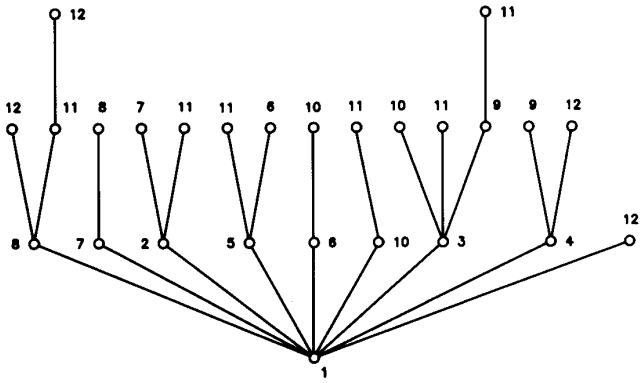


Fig. 13. Edge-order tree for Fig. 12.

$G_{i+2}, \cdots, G_n = G$. In Fig. 12 we show a planar embedding of the planar graph G shown in Fig. 1 obtained using the above procedure.

VI. Horvert Representation of a Planar Graph

In [9], Otten and van Wijk introduced the concept of horvert representation of a planar graph. As mentioned in Section I, in a horvert representation vertices are represented by horizontal segments and edges are drawn as vertical segments. Also, the segments representing the vertices are drawn at different vertical levels such that the one representing the vertex with *st*-number 1 is placed at level i. Otten and van Wijk also described a procedure to obtain a horvert representation using a planar embedding and the corresponding dual of the given graph. Linear-time algorithms to construct the horvert representation have recently been presented independently by Tamassia and Tollis [12], and Rosenstiehl and Tarjan [13]. These algorithms also use a dual of the given graph. In this section we present another linear-time algorithm to construct a horvert representation. Our algorithm does not involve the dual of the given graph.

Given a graph G and an *st*-numbering N of G. For each vertex $i < n$, let $\{i_1, i_2, \cdots, i_k\}$ be the set of higher numbered neighbors of i. Consider now a planar embedding of G. Suppose for each $i < n$, the edges $(i, i_1), (i, i_2), \cdots, (i, i_k)$ appear in that order as we scan them clockwise around i. Then we define the γ-order of i as $\gamma(i) = (i_1, i_2, \cdots, i_k)$.

$$\begin{bmatrix} 1 & 8 & 8 & 11 & 1 & 7 & 1 & 2 & 2 & 1 & 5 & 5 & 1 \\ 8 & 12 & 11 & 12 & 7 & 8 & 2 & 7 & 11 & 5 & 11 & 6 & 6 \end{bmatrix}$$

For example, $\gamma(3) = (10, 11, 9)$ in the planar graph embedded as in Fig. 12. In [10] Chiba *et al.* presented a very simple and elegant algorithm based on depth-first-search (DFS) to determine the γ-orders using the τ-orders of the vertices.

Our algorithm for horvert representation first constructs what we call the *edge-order tree* of the given graph G and then performs a DFS of this tree. The order in which the edges of the tree are traversed during this DFS would determine the order in which edges of G have to be drawn in the horvert representation.

The edge-order tree of G is a directed tree T each edge of which corresponds to a unique edge of G. Each node of T is assigned a label $i \in \{1, 2, \cdots, n\}$ and the unique node labelled 1 is the root of T. We construct T as follows. We start with a single node labelled 1. At the ith step, we pick the node labelled i and draw the directed edges (i, i_1), $(i, i_2), \cdots, (i, i_k)$ in that order (as prescribed by $\gamma(i)$) and clockwise around i. The end nodes of these edges are assigned the labels $i_1, i_2, \cdots, i_k$, respectively. The construction of T is completed after $(n-1)$ steps. Note that in T more than one node may have the same label. As an example, the edge-order tree of the graph embedded as in Fig. 12 is shown in Fig. 13.

After constructing the edge-order tree T, we perform a DFS on T, starting with the node labelled 1. During this DFS, the outgoing edges at a node are traversed in the order in which these edges were drawn around that node.

Let us now define a $2 \times m$ array Edge-Order as follows. If the edge (i, j), $i < j$, is the kth edge traversed during the DFS of T, then Edge-Order $(i, k) = i$ and Edge-Order $(2, k) = j$. In our horvert representation, the edge (i, j) will be drawn as a vertical line connecting the two points with coordinates (k, i) and (k, j). To complete the representation, each level is then examined. All the points on this level which represent a vertex of G are connected together by a straight line segment representing the corresponding vertex. As an example, using the edge-order tree of Fig. 13, we obtain the array Edge-Order as

$$\begin{bmatrix} 6 & 1 & 10 & 1 & 3 & 3 & 3 & 9 & 1 & 4 & 4 & 1 \\ 10 & 10 & 11 & 3 & 10 & 11 & 9 & 11 & 4 & 9 & 2 & 12 \end{bmatrix}.$$

It is easy to prove that the construction we have just described indeed produces a horvert representation. Suppose that, while constructing the edge-order tree T, we place at the same vertical level all nodes having the label i. Then the tree at the beginning of the ith step in the construction of T is a representation of the bush form B'_{i-1}. So, the nodes labelled i represent the corresponding virtual vertices of B'_{i-1} and must, therefore, appear consecutively on level i. This means that if we connect all the nodes of t labelled i by a horizontal segment, then this

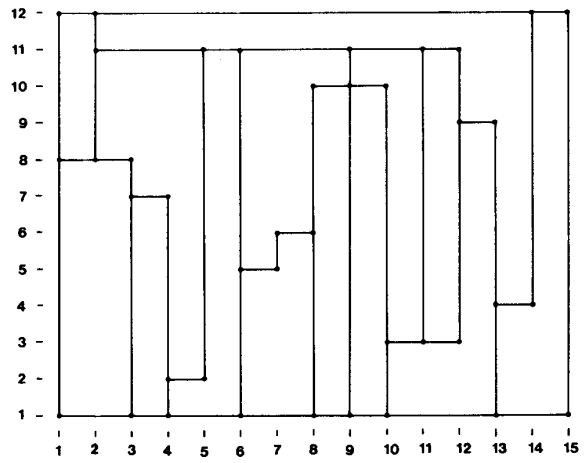


Fig. 14. A horvert representation for Fig. 1.

segment will not intersect with any of the edges of T except possibly at their end nodes. Since in our representation the vertical segments corresponding to the edges leaving vertex i in G appear around the horizontal segment corresponding to vertex i in the order specified by $\gamma(i)$, it follows that in this representation no vertical segment will intersect with any horizontal segment and hence it is a horvert representation.

We now observe that Edge-Order of a planar graph G is a very succinct representation of a planar embedding of G. For example, we can obtain $\tau(i)$ for any i from the Edge-Order as follows. First, we scan the second row of Edge-Order. If i occurs in columns $k_1, k_2, \cdots, k_p$, then the entries in position $(1, k_1), (1, k_2), \cdots, (1, k_p)$ will give $\tau(i)$. Similarly, we can also obtain the γ-orders by scanning the first row of Edge-Order.

Let us next define the width and height of a horvert representation as the number of horizontal levels and the number of vertical levels, respectively, required for the representation. Our procedure constructs a horvert representation with width m and height n. We now describe a procedure which obtains a very compact horvert representation with width at most $2n-4$.

Consider again the edge-order tree T. Let a DFS be performed on T as explained before. Suppose that T has r leaves $l_1, l_2, \cdots, l_r$. Assume that if $i < j$, then the DFS visits l_i before it visits l_j. Let E_i be the set of edges of T traversed by the DFS before visiting leaf l_i. Then the sets $E_1', E_2', \cdots, E_r'$, such that $E_1' = E_1$ and $E_i' = E_i - E_{i-1}$, $i \geqslant 2$, defines a partition of the edge set of T. It is easy to verify that all the edges in any E_i' can be assigned the same horizontal level. For example, the edge-order tree of Fig. 13 has 15 leaves and so at most 15 horizontal levels are required to construct a horvert representation of the graph of Fig. 1. One such horvert representation is shown in Fig. 14.

We next get an upperbound on r. Note that in an *st*-numbering each vertex $i \neq 1, n$ is adjacent to at least one higher numbered vertex and to at least one lower numbered vertex. Hence, it follows that for any vertex i, $1 < i < n$, there are at least two edges of the form (i, p) and (i, q) with $p > i$ and $q < i$ such that these two edges will both belong to the same E_k', $1 \leqslant k \leqslant r$. This means that $r \leqslant m - (n-2)$. Since for a simple planar graph $m \leqslant 3n-6$, it follows that for such a graph $r \leqslant 2n-4$. In the following EDGE–ORDER will refer to our procedure described above for assigning horizontal levels to the edges in a horvert representation. The following theorem follows from our discussion so far.

Theorem 8:

For a simple biconnected planar graph procedure EDGE–ORDER achieves a horvert representation of width no more than $2n-4$. ■

Clearly, the complexity of procedure EDGE–ORDER is $O(m) = O(n)$ since it essentially involves a depth-first-search of T. We wish to note that the algorithm due to Rosentiehl and Tarjan [13] also constructs a horvert representation of width no more than $2n-4$.

Our procedure for horvert representation assumes that γ-orders are available. We now show that this representation can also be obtained using τ-orders. Given a biconnected graph G and an *st*-numbering N of the vertices of G. Suppose we now renumber the vertices of G such that a vertex is assigned the number $n-i+1$ if its *st*-number is under N is i. It is easy to see that the new numbering N' is also an *st*-numbering. Furthermore, τ-orders under N become a set of γ-orders under N' and vice versa. Thus if only τ-orders under N are available, then we can treat them as γ-orders under numbering N' and apply the procedures of this section to obtain a horvert representation.

VII. Conclusion

In this paper, we have discussed the problem of constructing a planar embedding of a planar graph. Our discussion is based on Lempel, Even, and Cederbaum's planarity testing algorithm and its PQ-tree implementation. We first developed an $O(n)$ time algorithm to determine the τ-orders of the vertices in a planar embedding. The τ-order of vertex i is the anticlockwise order in which edges connecting i to lower numbered neighbors appear in the planar embedding. We then developed an $O(n)$ time algorithm to determine what we call the vertex order which gives the relative positions of the vertices in the planar embedding. The vertex order captures the structural information as to the relative placement of vertices provided by the PQ-tree reduction algorithm. We also described a procedure to obtain a planar embedding based on the vertex order and the τ-orders.

We next defined the concept of γ-orders of vertices. The γ-order of a vertex is the clockwise order in which edges appear in a planar embedding. We presented a linear-time algorithm to construct what we call the array Edge-Order and a compact horvert representation of a planar graph. Our algorithm does not involve the dual of the given graph and uses the γ-orders. We also showed that the width of the horvert representation obtained by this algorithm is at most $2n-4$. The algorithm in [13] also achieves a horvert

representation of width no more than $2n-4$. We showed that a horvert representation can also be obtained using τ-orders.

The Edge-Order is a very succinct representation of a planar embedding. For example, it contains all the information necessary to determine γ-orders as well as τ-orders. We note that in [10] Chiba *et al.* give a very elegant algorithm to determine γ-orders using τ-orders.

REFERENCES

[1] R. Raghavan and S. Sahni, "Single row routing," *IEEE Trans. Computers*, vol. C-32, pp. 209–220, Mar. 1983.

[2] K. Thulasiraman and R. Jayakumar, "Optimum single-row routing with no crossovers: An approach based on planar embedding," in *Proc. 1985 Canadian Conf. VLSI*, Toronto, Canada, pp. 192–195, Nov. 1985.

[3] M. Marek-Sadowska and T. T. Tarng, "Single-layer routing for VLSI: Analysis and algorithms," *IEEE Trans. Computer-aided Design*, vol. CAD-2, pp. 246–259, Oct. 1983.

[4] J. Hopcroft and R. Tarjan, "A VlogV algorithm for isomorphism of triconnected planar graphs," *J. Comp. Syst. Sci.*, vol. 7, no. 3, pp. 323–331, June 1973.

[5] J. Hopcroft and J. K. Wong, "Linear time algorithms for isomorphism of planar graphs," in *Proc. Sixth Ann. ACM Symp. on Theory of Computing*, pp. 172–184, 1974.

[6] F. Hadlock, "Finding a maximum cut in a planar graph in polynomial time," *SIAM J. Comput.*, vol. 4, no. 3, pp. 221–225, Sept. 1975.

[7] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-completeness*. San Francisco: Freeman, 1979.

[8] W. T. Tutte, "How to draw a graph," in *Proc. London Math. Soc.*, vol. 13, no. 3, pp. 743–768, Apr. 1963.

[9] R. H. J. M. Otten and J. G. van Wijk, "Graph representation in interactive layout design," in *Proc. IEEE Int. Symp. on Circuits and Systems*, pp. 914–918, 1978.

[10] N. Chiba, T. Nishizeki, S. Abe, and T. Ozawa, "A linear algorithm for embedding planar graphs using *PQ*-trees," *J. Comp. Syst. Sci.*, vol. 30, no. 1, pp. 54–76, Feb. 1985.

[11] N. Chiba, T. Yamanouchi, and T. Nishizeki, "Linear algorithms for convex drawings of planar graphs," in *Progress in Graph Theory*, J. A. Bondy and U. S. R. Murty (Eds.), New York: Academic, pp. 153–165, 1984.

[12] R. Tamassia and I. G. Tollis, "A unified approach to visibility representations of planar graphs," *Discrete and Computational Geometry*, Springer-Verlag, vol. 1, pp. 321–341, 1986.

[13] P. Rosenstiehl and R. E. Tarjan, "Rectilinear planar layout and bipolar orientations of planar graphs," *Discrete and Computational Geometry*, Springer-Verlag, vol. 1, pp. 343–353, 1986.

[14] J. Hopcroft and R. Tarjan, "Efficient planarity testing," *J. ACM*, vol. 21, no. 4, pp. 549–568, Oct. 1974.

[15] A. Lempel, S. Even, and I. Cederbaum, "An algorithm for planarity testing of graphs," in *Theory of Graphs: Int. Symp.*, Rome, July, 1966, P. Rosenstiehl (Ed.), New York: Gordon and Breach, 1967, pp. 215–232.

[16] R. Tarjan, "An efficient planarity algorithm," STAN-CS 244-71, Computer Sci. Dep., Stanford Univ., Nov. 1971.

[17] S. G. Williamson, "Embedding graphs in the plane—Algorithmic aspects," in *Combinatorial Mathematics, Optimal Designs and their Applications*, Annals of Discrete Mathematics, (Ed.) J. Srivastava, no. 6, New York: North-Holland, 1980, pp. 349–384.

[18] W. M. Brehaut, "On planar graphs and the plainer nonplanar graphs," Doctoral dissertation, Univ. of Waterloo, Sept. 1974.

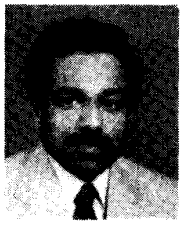
[19] ___, "Efficient planar embedding," in *Proc. 7th South-Eastern Conf. on Combinatorics, Graph Theory, and Computing*, 1976, pp. 177–190.

[20] K. S. Booth and G. S. Lueker, "Testing for the consecutive ones property, interval graphs, and graph planarity using *PQ*-tree algorithms," *J. Comp. Syst. Sci.*, vol. 13, no. 3, pp. 335–379, Dec. 1976.

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

[22] J. Ebert, "*St*-Ordering the vertices of biconnected graphs," *Computing*, vol. 30, pp. 19–33, 1983.

[23] S. Even, *Graph Algorithms*. Potomac, MD: Computer Science Press, 1979.



R. Jayakumar was born in India in 1955. He received the B.E. (hon's) degree in electronics and communication engineering from the University of Madras, Madras, India, in 1977, the M.S. degree in computer science from the Indian Institute of Technology, Madras, India, in 1980, and the Ph.D. degree in engineering and computer science from Concordia University, Montreal, Canada in 1984.

Since 1984 he is an Assistant Professor of Computer Science at Concordia University, Montreal, Canada. His research interests are in VLSI algorithms and architectures, fault-tolerant VLSI systems, VLSI design automation, and graph theory and graph algorithms.

Dr. Jayakumar is a member of the Association for Computing Machinery.

✠

K. Thulasiraman (M'72–SM'84), for a photograph and biography please see page 265 of this issue.

✠

M. N. S. Swamy (S'59–M'62–SM'74–F'80) was born on April 7, 1935. He received the B.Sc (with honors) degree in mathematics from Mysore University, Mysore, India, in 1954, the Diploma in electrical communication engineering from the Indian Institute of Science, Bangalore, India, in 1957, and the M.Sc. and Ph.D. degrees in electrical engineering from the University of Saskatchewan, Saskatoon, Sask, Canada, in 1960 and 1963, respectively.

He worked as a Senior Research Assistant at the Indian Institute of Science until 1959, when he began graduate study at the University of Saskatchewan. In 1963, he returned to India to work at the Indian Institute of Technology, Madras. From 1964 to 1965, he was an Assistant Professor of Mathematics at the University of Saskatchewan. He also taught as a Professor of Electrical Engineering at the Technical University of Nova Scotia, Halifax, N.S., Canada, and the University of Calgary, Calgary, Alta., Canada. He was Chairman of the Department of Electrical Engineering, Concordia University (formerly Sir George Williams University), Montreal, P. Q., Canada, until August 1977, when he became Dean of Engineering and Computer Science of the same university. He has published a number of papers on number theory, semiconductor circuits, control systems, and network theory. He is coauthor of the book *Graphs, Networks and Algorithms* (New York: Wiley, 1981). A Russian translation of this book was published by Mir Publishers Moscow, in 1984.

Dr. Swamy is a Fellow of several professional societies including the Institution of Electrical Engineers (U.K.), the Institution of Electronic and Radio Engineers (U.K.), the Engineering Institute of Canada, the Institution of Engineers (India), and the Institution of Electronics and Telecommunications Engineering (India). He is Associated Editor of the *Fibonacci Quarterly* and the new journal *Circuits, Systems and Signal Processing*. He was Vice-President of the IEEE Circuits and Systems Society in 1976, Program Chairman for the 1973 IEEE-CAS Symposium, and the General Chairman for the 1984 IEEE-CAS Symposium. He was an Associate Editor of the CAS TRANSACTIONS during 1985–1987. He is co-recipient, with Drs. L. M. Roytman and E. I. Plotkin, of the IEEE-CAS 1986 Guillemin–Cauer Best Paper Award.