$O(n^2)$ Algorithms for Graph Planarization

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

Abstract—**In this paper we present two $O(n^2)$ planarization algorithms—PLANARIZE and MAXIMAL-PLANARIZE. These algorithms are based on Lempel, Even, and Cederbaum's planarity testing algorithm [9] and its implementation using *PQ*-trees [8]. Algorithm PLANARIZE is for the construction of a spanning planar subgraph of an n-vertex nonplanar graph. This algorithm proceeds by embedding one vertex at a time and, at each step, adds the maximum number of edges possible without creating nonplanarity of the resultant graph. Given a biconnected spanning planar subgraph G_p of a nonplanar graph G, algorithm MAXIMAL-PLANARIZE constructs a maximal planar subgraph of G which contains G_p. This latter algorithm can also be used to maximally planarize a biconnected planar 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. A subgraph G' of a nonplanar graph G is a *maximal planar subgraph* of G if G' is planar and adding to G' any edge not present in G' results in a nonplanar subgraph of G. This process of removing a set of edges from G to obtain a maximal planar subgraph is known as *maximal planarization of the nonplanar graph* G. On the other hand, maximal planarization of a planar subgraph G refers to the process of adding a maximal set of edges to G without causing nonplanarity. Maximal planarization of a nonplanar graph is an important problem encountered in the automated design of printed circuit boards. If an electronic circuit cannot be wired on a single layer of a printed circuit board, then we would like to determine the minimum number of layers necessary to wire the circuit. Since only a planar circuit can be wired on a single layer board, we would like to decompose the nonplanar circuit into a minimum number of maximal planar circuits. In general, for a nonplanar graph, neither the set of edges to be removed to maximally planarize it nor the number of these edges is unique.

Determining the minimum number of edges whose removal from a nonplanar graph will yield a maximal planar subgraph is an NP-complete problem [1]. However, a few algorithms which attempt to produce maximal planar subgraphs having the largest possible number of edges have been reported [2]–[4]. Recently, Chiba, Nishioka, and Shirakawa [5] modified Hopcroft and Tarjan's planarity testing algorithm [6] to construct a maximal planar subgraph of a nonplanar graph. Their algorithm needs $O(mn)$ time and $O(mn)$ space for a nonplanar graph having n vertices and m edges. Ozawa and Takahashi [7] proposed another $O(mn)$ time and $O(m + n)$ space algorithm to planarize a nonplanar graph using the *PQ*-tree implementation [8] of Lempel, Even, and Cederbaum's planarity testing algorithm [9], [10]; in short the LEC algorithm. For a general graph this algorithm may not determine a maximal planar subgraph [11]. Moreover, in certain cases, this algorithm may terminate without considering all the vertices; in other words, it may not produce a spanning planar subgraph.

Whereas the planarization algorithm of [5] constructs the required planar subgraphs by considering one edge at a time, the algorithm of [7] proceeds by considering one vertex at a time. Since an $O(mn)$ maximal planarization algorithm can be constructed in a straightforward manner by adding one edge at a time and testing for planarity at each step, these two algorithms are not significant as far as their complexities are concerned. However, the algorithm of [7] is quite interesting because at each step of this algorithm as many edges as possible are added.

It seems that no maximal planarization algorithm of complexity better than $O(mn)$ will be possible. So, in this paper, we focus our attention on the design of $O(n^2)$ planarization algorithms. We present two planarization algorithms—PLANARIZE and MAXIMAL-PLANARIZE of time complexity $O(n^2)$ and space complexity $O(mn)$. These algorithms are based on Lempel, Even, and Cederbaum's planarity testing algorithm [9] and its implementation using PQ-trees [8]. Algorithm PLANARIZE is for the construction of a spanning planar subgraph of an n-vertex nonplanar graph. This algorithm proceeds by embedding one vertex at a time and, at each step, adds the maximum number of edges possible without creating nonplanarity of the resultant graph. Given a biconnected spanning planar subgraph G_p of a nonplanar graph G, algorithm MAXIMAL-PLANARIZE constructs a maximal planar subgraph of G which contains G_p. This latter algorithm can also be used to maximally planarize a biconnected planar graph.

In the following, proofs of some of the results are omitted in order to conserve space. These proofs may be found in [12].

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

Manuscript received April 11, 1988; revised October 21, 1988. This work was supported by the Natural Sciences and Engineering Council of Canada under Grant A0904, Grant A4680, and Grant A7739. The review of this paper was arranged by Associate Editor R. H. J. M. Otten.

The authors are with the Department of Computer Science, Concordia University, Montreal, P.Q., Canada H3G 1M8.

IEEE Log Number 8825916.

is called an *st-numbering* [13], [14]. The graph G is then called an *st*-graph. Let G_K, $1 \leq k \leq n$, 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. A drawing of B_k is called a *bush form* of B_k if in the drawing vertices with higher labels appear at higher levels and all the virtual vertices appear at the same level.

It can be shown [9], [10] that the *st*-graph G is planar if and only if for every B_k, $2 \leq k \leq n-2$, there exists a planar drawing B_k' isomorphic to B_k such that in B_k' all the virtual vertices labeled $k+1$ appear consecutively. As a consequence of *st*-numbering, a cut vertex v in B_k will be the lowest vertex in all the maximal biconnected components (except the one containing 1) with respect to v. These biconnected components, called *blocks*, will have the same structure as a bush form. 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, (ii) *P-nodes* in T_k represent cut vertices in B_k, and (iii) *Q-nodes* of T_k represent the maximal biconnected components in B_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 $S(k+1)$; X is said to be *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 *pruned pertinent subtree* of T_k is the smallest connected subgraph which contains all the leaves in $S(k+1)$. The root of the pertinent subtree is called the *pertinent root*. 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 [10] that B_k' exists if and only if T_k can be converted into an equivalent *PQ*-tree T_k' such that all the 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. The reduction process consists of two phases. In the first phase, called the *bubble up* phase, the pertinent subtree is identified. In the second phase, called the *reduction* phase, pattern matching and corresponding replacements are carried out using the two types of operations mentioned above.

To construct T_{k+1} from T_k, we first reduce T_k to T_k^* and then replace all the leaves corresponding to the virtual vertex $k+1$ by a *P*-node whose children are the leaves corresponding to the edges incident out of vertex $k+1$ in G. 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 [8].

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

More details on the LEC algorithm may be found in [8], [10], [15].

III. Principle of an Approach for Planarization

In this section, we discuss the basic principle of an approach for planarization due to Ozawa and Takahashi [7]. This approach is based on the LEC algorithm for a planarity testing. Let G denote a simple biconnected *st*-graph. Let $T_1, T_2, \cdots, T_{n-1}$ be the *PQ*-trees corresponding to the bush forms of G. Ozawa and Takahashi [7] classify the nodes of any *PQ*-tree according to their frontier as follows.

Type W: A node is said to be Type W if its frontier consists of only non-pertinent leaves.
Type B: A node is said to be Type B if its frontier consists of only pertinent leaves.
Type H: A node X is said to be Type H if the subtree rooted at X can be rearranged such that all the descendant pertinent leaves of X appear consecutively at either the left or the right end of the frontier.
Type A: A node X is said to be Type A if the subtree rooted at X can be rearranged such that all the descendant pertinent leaves of X appear consecutively in the middle of the frontier with at least one non-pertinent leaf appearing at each end of the frontier.

The central concept of the planarization algorithm is stated in the following theorem.

Theorem 2: An n-vertex graph G is planar if and only if the pertinent roots in all the *PQ*-trees $T_2, T_3, \cdots, T_{n-1}$ of G are Type B, H or A. //

We call a *PQ*-tree *reducible* if its pertinent root is Type B, H, or A; otherwise it is *irreducible*. Theorem 2 implies that the graph G is planar if and only if all the T_i's are reducible. If any T_i is irreducible, we can make it reducible by appropriately deleting some of the leaves from it. Of course, we would like to delete a minimum number of leaves while trying to make T_i reducible. If we make all the T_i's reducible this way, then a planar subgraph can be obtained by removing from the nonplanar graph the edges corresponding to the leaves that are deleted.

It is easy to see that the PQ-tree T_{n-1} is always reducible because its root is type B. The tree T_1 is also reducible because it has only one pertinent leaf—the leaf corresponding to the edge $(1, 2)$. Consider now an irreducible PQ-tree T_i of an n-vertex nonplanar graph. For a node X in T_i, let w, b, h, and a be the minimum number of descendant leaves of X which should be deleted from T_i so that X becomes Type W, B, H, and A, respectively. We denote these numbers of a node as $[w, b, h, a]$. Any node in T_i may be made Type W, B, H, or A by appropriately deciding the types of its children. So the $[w, b, h, a]$ number of any node can be computed from that of its children. Thus to make T_i reducible, we first traverse it bottom-up from the leaves to the pertinent root and compute the $[w, b, h, a]$ number for every node in T_i. Once the $[w, b, h, a]$ number of the pertinent root is computed, we make the pertinent root Type B, H, or A depending on which one of the numbers b, h, and a of the root is the smallest. After determining the type of the pertinent root, we traverse T_i top-down from the pertinent root to the leaves and decide the type of each node in the pertinent subtree of T_i. Note that the type of a node uniquely determines the types of its children and so the types of all the leaves in T_i can be determined by this top-down traversal. This information would help us decide the nodes to be deleted from T_i in order to make it reducible. After deleting these nodes from T_i, we can apply the reduction procedure to obtain T_i^*.

Repeating the above procedure for each irreducible T_i, we can obtain a planar subgraph of the nonplanar graph. It is easy to see that if the minimum of b, h, and a for the pertinent root in a PQ-tree T_i is zero, then T_i is reducible. Note that this algorithm may not determine a maximal planar subgraph (see [11]). However, in the case of complete graphs, this algorithm produces a maximal planar subgraph.

Computing the $[w, b, h, a]$ numbers for the nodes in a PQ-tree is a crucial step in procedure GRAPH-PLANARIZE. Ozawa and Takahashi [7] have presented formulas to compute these numbers. The main drawback of their algorithm arises from the fact that they permit deletion of both pertinent and non-pertinent leaves from a tree T_i to make it reducible. Since in T_i, the pertinent leaves correspond to the edges entering vertex $i+$ in the st-graph G and the non-pertinent leaves correspond to those entering vertices greater than $i+1$, it may so happen that as the algorithm proceeds, all the edges entering a vertex $k > i+1$ may get removed from G and thus vertex k and some of other vertices may not be present in the resulting planar subgraph. Thus the planar subgraph determined by Ozawa and Takahashi's algorithm may not even be a spanning subgraph of the given nonplanar graph.

IV. A New Graph-Planarization Algorithm

In this section we develop an efficient algorithm to determine a spanning planar subgraph of a nonplanar graph G. The planarization approach discussed in Section III will form the basis of this algorithm. We modify Ozawa and Takahashi's approach so that deletion of only pertinent leaves is permitted.

Theorem 3: The planarization algorithm of Section III will determine a spanning planar subgraph of a biconnected n-vertex nonplanar graph, if only pertinent leaves are considered for deletion while making any PQ-tree T_i, $3 \le i \le n-2$, reducible.

Proof: Note that a PQ-tree with only one pertinent leaf is always reducible. So it follows that from no PQ-tree all the pertinent leaves will get deleted, if only pertinent leaves are to be chosen for deletion. This means that in the subgraph that results at the end of the application of the algorithm, each vertex will be connected to at least one lower numbered vertex. Thus the subgraph determined will be a spanning subgraph of the given nonplanar graph. //

Let G be a nonplanar st-graph. Let E_i, $2 \le i \le n$, be the set of edges entering vertex i in G. We determine a planar subgraph of G by removing a sequence E_4', E_5', $\cdots$, E_{n-1}' ($E_i' \subset E_i$) of edges such that for each i the subgraph of G obtained by removing the edges in E_4', E_5', $\cdots$, E_i' contains a planar subgraph induced by the vertex set $\{1, 2, \cdots, i\}$. Thus after removing the edges in E_4', E_5', $\cdots$, E_{n-1}', we obtain a planar subgraph of G. It is easy to see that the edges in E_{i+1}', $3 \le i \le n-2$, correspond to the pertinent leaves in the PQ-tree T_i which should be deleted to make T_i reducible.

In order to make a PQ-tree T_i reducible, we first compute the $[w, b, h, a]$ number for each node in T_i. Recall that a node in T_i is full if the number of leaves in the pertinent subtree rooted at the node is equal to the number of pertinent leaves. Note that while processing T_i to make it reducible, a full node and all its descendants may be made Type W, or they will remain Type B. On the other hand partial nodes may be made Type W, H, or A; but never Type B because we delete only pertinent leaves from T_i. Thus any pertinent node in T_i may be made Type W, H, or A only. So we need to compute only the w, h, and a numbers for the pertinent nodes in T_i. We denote these numbers as $[w, h, a]$.

Now we develop formulas to compute the $[w, h, a]$ number for each pertinent node in T_i. We process T_i bottom-up from the pertinent leaves to the pertinent root. Note that we can compute the $[w, h, a]$ number for a node from the numbers of its pertinent children. In the following, $P(X)$ denotes the set of pertinent children of X and $\mathrm{Par}(X)$ denotes the set of partial children of X. Along with the $[w, h, a]$ number for each pertinent node, we also determine, for each pertinent node which is not a leaf, three children called h-child1(X), h-child2(X) and a-child(X) which will be used later to decide the type of each pertinent child of X in the reducible T_i.

(i) *X is a pertinent leaf.*

In this case $w = 1$, $h = 0$, and $a = 0$.

(ii) *X is a full node.*

In this case $h = 0$, $a = 0$, and

$$w = \sum_{i \in P(X)} w_i$$

(iii) *X is a partial P-node.*

To make X Type W, all its pertinent children should be made Type W. Thus

$$w = \sum_{i \in P(X)} w_i.$$

We can make X Type H by making all its full children Type B, one partial child Type H and all other partial children Type W. Thus the h number of X is given by

$$h = \sum_{i \in \text{Par}(X)} w_i - \max_{i \in \text{Par}(X)} \{(w_i - h_i)\}.$$

In this case the partial child which is made Type H will be the h-child1(X).

We can make X Type A in two different ways. We can make one partial child of X Type A and all other pertinent children Type W. In this case

$$\alpha_1 = \sum_{i \in \text{Par}(X)} w_i - \max_{i \in P(X)} \{(w_i - a_i)\}$$

descendant pertinent leaves of X will have to be deleted. The partial child which is made Type A will be the a-child(X). On the other hand, if we make two partial children Type H, all full children Type B and all other pertinent children Type W, then

$$\alpha_2 = \sum_{i \in \text{Par}(X)} w_i - \max1_{i \in \text{Par}(X)} \{(w_i - h_i)\} - \max2_{i \in \text{Par}(X)} \{(w_i - h_i)\}$$

descendant pertinent leaves will have to be deleted from T_i to make X Type A, where max1 is the first maximum and max2 is the second maximum. The partial child having max1 $\{(w_i - h_i)\}$ will be the h-child1(X) and the one having max2 $\{(w_i - h_i)\}$ will be the h-child2(X). Thus the P-node X can be made Type A by deleting

$$a = \min \{\alpha_1, \alpha_2\}$$

pertinent leaves from T_i. If the value of a is different from α_1, then we make a-child(X) empty.

(iv) *X is a partial Q-node.*

To make X Type W, all its pertinent children should be made Type W. Thus for X

$$w = \sum_{i \in P(X)} w_i.$$

To compute the h number of X, first note that X can be made Type H only if either its leftmost child or its rightmost child is pertinent. Suppose that the leftmost child of X is pertinent. Then let us traverse the children of X from left to right and find $P_L(X)$, the maximal consecutive sequence of pertinent children such that only the rightmost node in $P_L(X)$ may be partial. If the leftmost child of X is not pertinent, then $P_L(X)$ will be empty. Suppose, on the other hand, that the rightmost child of X is pertinent. As we traverse the children of X from right to left, let $P_R(X)$ be the maximal consecutive sequence of pertinent children such that only the leftmost node in $P_R(X)$ may be partial. If the rightmost child of X is not pertinent, then $P_R(X)$ is empty. We can easily see that X can be made Type H by deleting

$$h = \sum_{i \in P(X)} w_i - \max \cdot \left\{ \sum_{i \in P_L(X)} (w_i - h_i), \sum_{i \in P_R(X)} (w_i - h_i) \right\}$$

pertinent leaves from T_i. We let h-child1(X) be the rightmost node in $P_L(X)$ or the leftmost node in $P_R(X)$ depending on which one has the maximum $\Sigma (w_i - h_i)$ sum in the above formula for h.

Node X can be made Type A in two different ways. We can make one of the pertinent children of X Type A and all the other pertinent children Type W. This can be achieved by deleting

$$\beta_1 = \sum_{i \in P(X)} w_i - \max_{i \in P(X)} \{(w_i - a_i)\}$$

pertinent leaves from T_i. In this case the pertinent child having max $\{(w_i - a_i)\}$ will be the a-child(X). let $P_A(X)$ be a maximal consecutive sequence of pertinent children of X such that all the nodes in $P_A(X)$ except the leftmost and the rightmost ones are full. The endmost nodes may be full or partial. Then we can make X Type A by making all the full nodes in $P_A(X)$ Type B, the partial nodes in $P_A(X)$ Type H and all the other pertinent children of X Type W. Note that there may be more than one $P_A(X)$. Thus we can make X Type A by deleting

$$\beta_2 = \sum_{i \in P(X)} w_i - \max_{P_A(X)} \left\{ \sum_{i \in P_A(X)} (w_i - h_i) \right\}$$

pertinent leaves from T_i. In this case we let the leftmost node in the $P_A(X)$ selected be the h-child2(X). Thus node X can be made Type A with the deletion of

$$a = \min \{\beta_1, \beta_2\}$$

pertinent leaves from T_i. If the value of a is different from β_1, then we make a-child(X) empty.

Traversing T_i bottom-up we can compute the $[w, h, a]$ number for each pertinent node in T_i using the above formulas. The procedure which computes these numbers for a given T_i will be referred to as COMPUTE1(T_i).

Lemma 1: The $[w, h, a]$ numbers for all the pertinent nodes can be correctly computed in $O(n^2)$ time.

Proof: Proof of correctness follows from our discussions so far. As regards the complexity, note that for a Q-node in T_i procedure COMPUTE1(T_i) traverses all the children of the node. Thus the amount of work done for all the Q-nodes in a T_i is proportional to the number of children of all the Q-nodes in T_i. The children of a Q-node corresponding to a block represent vertices, except the lowest, on the outside window of the block. Moreover, any vertex in G which is represented as a child of a Q-node in T_i can appear on the outside window of only one block. Thus the total number of children of all the Q-nodes in T_i is less than or equal to n, the number of vertices in G. For a P-node, the work done by procedure COMPUTE1(T_i) is proportional to the number of its per-

tinent children. A pertinent child of a P-node is either a P- or Q-node or a leaf. Since a Q-node represents a block, there are no more than n Q-nodes in any T_i. Also the number of pertinent leaves in T_i is in-deg$(i + 1)$, where in-deg$(i + 1)$ is the number of edges entering vertex $i + 1$ in G. Furthermore the number of P-nodes in T_i is at most i. Thus the amount of work for all the P-nodes in T_i is $O(n +$ in-deg$(i + 1))$. It follows from the above that the amount of work done by procedure COMPUTE1(T_i) for all the Q-nodes and P-nodes in T_i is $O(n +$ in-deg$(i + 1))$. Summing up the work done for all T_i's, we get the complexity of computing the $[w, h, a]$ numbers as $O(m + n^2) = O(n^2)$. //

After computing the $[w, h, a]$ number for the pertinent root of T_i, we can determine whether T_i is reducible or not. If the minimum of h and a is zero for the pertinent root of T_i, then T_i is reducible. If T_i is not reducible, then we make the pertinent root of T_i Type H or A depending on which one of h and a is minimum, and make T_i reducible by deleting the necessary pertinent leaves from T_i. Now we need to determine the type of each pertinent node in T_i to obtain a reducible T_i. Note that T_i may have certain full nodes. If we decide to keep any such full node, then we mark it Type B.

Consider now a pertinent node X and T_i whose type has been determined. To start with X is the pertinent root. If X is Type B, then it is a full node and we would like to keep X as well as all its descendants in T_i. So no action needs to be taken in this case. On the other hand, if X is not Type B, then we traverse the pertinent descendants of X to determine their type. An easy case is when X is a leaf. Then it should be Type W and so we have to delete it from T_i. We also have to remove the edge corresponding to X from G. Thus in this case, the edge corresponding to X should be included in E'_{i+1}. If X is not a leaf, then we have the following different cases to consider.

Suppose X is Type W. Then all its pertinent children should be made Type W. Moreover, if any of these pertinent children is a full node, then the entire subtree of T_i rooted at that full child should be deleted from T_i.

If X is Type H and a P-node, then we make the partial child h-child1(X) Type H, all the full children Type B and all other partial children Type W. If X is Type H, but a Q-node, then we traverse the children of X from h-child1(X) towards the rightmost child and determine the maximal consecutive sequence of pertinent children $P_L(X)$ or $P_R(X)$. We then make all the nodes in this sequence Type B; the rightmost node in $P_L(X)$ or the leftmost node in $P_R(X)$ are made Type H and all other pertinent children of X are made Type W.

Suppose X is Type A and a P-node. Then we process the pertinent children of X as follows. If a-child(X) is not empty, then we make a-child(X) Type A and all other pertinent children Type W. On the other hand, if a-child(X) is empty, then we make the partial children h-child1(X) and h-child2(X) Type H, all full children of X Type B and all other partial children of X Type W. If X is Type A and a Q-node, then we should process its pertinent children as follows. If a-child(X) is not empty, then we make a-child(X) Type A and all other pertinent children Type W. If a-child(X) is empty, then we traverse the children of X from h-child2(X) towards the rightmost child and find the maximal consecutive sequence $P_A(X)$ of pertinent children of X. Then we make all nodes in $P_A(X)$ Type B, the endmost nodes in $P_A(X)$, if they are partial, Type H and all other pertinent children Type W.

From the above discussions it should be clear that the type of any pertinent node in T_i uniquely determines the types of its pertinent children. Hence we process the PQ-tree T_i top-down from the pertinent root, and determine the set of edges E'_{i+1} and delete from T_i the nodes which are full and marked Type W. The procedure which achieves these will be denoted by DELETE-NODES(T_i). Since certain pertinent leaves are deleted from T_i, we have to update, if necessary, for each node the number of descendant leaves. Procedure DELETE-NODES(T_i) performs this update also.

Lemma 2: All the edges in the sets E'_{i+1}, $3 \le i \le n - 2$, can be determined and removed using procedure DELETE-NODES(T_i) in $O(n^2)$ time. //

Having made T_i reducible, we can now reduce it to obtain T_1^* using Booth and Lueker's PQ-tree reduction algorithm. We can then obtain the next PQ-tree T_{i+1} and repeat our procedures to make T_{i+1} reducible. Note that the reduction of all the reducible PQ-trees can be performed in $O(m + n)$ time if we keep the parent pointers for all children of P-nodes and for the endmost children of Q-nodes. Thus in Booth and Lueker's algorithm [8], interior children of Q-nodes in any T_i are not assigned valid parent pointers and if any such interior child becomes pertinent, then its parent pointer will be determined during the bubble-up phase. In our discussions so far, we have assumed that the correct parent pointer for every pertinent node is available. So we have to determine the parent pointers of all the pertinent nodes in T_i before processing it. Booth and Lueker's planarity testing algorithm stops when it detects during the bubble-up phase that certain pertinent nodes cannot be assigned parent pointers, for that would imply nonplanarity of the given graph. However, since our aim is to planarize the nonplanar graph, we would like to proceed further to find parent pointers of all the pertinent nodes. As a result, our bubble-up algorithm described below is different from Booth and Lueker's.

Let X be a pertinent node in T_i. If X is a child of a P-node or one of the endmost children of a Q-node, then it has a valid parent pointer. On the other hand, if X is an interior child of a Q-node, then its parent pointer will be empty. To find the correct parent pointer for X, we traverse the siblings of X from X towards the rightmost child and obtain the parent pointer for X from that of the rightmost child. Let Y be the parent of X in T_i. If at a later time another child Z of Y is processed to find its parent pointer, then the above procedure would require traversing again all the children of Y up to the rightmost child

and may result in visiting certain nodes several times. To avoid these unnecessary visits, we proceed as follows. When we traverse the children of Y from X to the rightmost child, we assign the parent pointer of the rightmost child to all the nodes traversed and store these nodes in a queue called interior-queue. So when a child Z of Y is processed, if its parent pointer is empty, then we traverse the siblings of Z until we find a node with a non-empty parent pointer. Though this path compression technique makes our bubble-up procedure efficient, many non-pertinent children of Q-nodes may be assigned parent pointer. In order to make the parent pointer of such non-pertinent nodes empty, we process the interior-queue at the end of the bubble-up. If any node in this queue is not pertinent, then its parent pointer is made empty.

The efficiencies of our procedures COMPUTE1(T_i) and DELETE-NODES(T_i) arise from the fact that we process only the pertinent children of any P-node. In a PQ-tree the pertinent children of a P-node may appear in any arbitrary order and so we may have to traverse all the children of a P-node to find the pertinent children. In order to avoid this, we split the children of each pertinent P-node into two groups—one group consisting of pertinent children only and the other consisting of only non-pertinent children. The procedure which finds the parent pointer for all the pertinent nodes in a PQ-tree T_i and groups the pertinent children of P-nodes together as described above will be referred to as BUBBLE-UP(T_i). This procedure also computes the number of pertinent children as well as the number of descendant pertinent leaves of each pertinent node in the PQ-tree T_i. The following lemma shows that procedure BUBBLE-UP(T_i) has the same time complexity as the other procedures developed so far.

Lemma 3: The total cost of procedure BUBBLE-UP(T_i) for all $2 \leq i \leq n-2$ is $O(n^2)$. //

Procedures COMPUTE1(T_i) and DELETE-NODES(T_i) require that we should be able to determine whether a pertinent node in T_i is full or partial. A pertinent node is full if the number of descendant pertinent leaves of the node is equal to the number of its descendant leaves; otherwise it is partial. Procedure BUBBLE-UP(T_i) determines the number of descendant pertinent leaves of every pertinent node in T_i. Now we should find a way of determining the number of descendant leaves of every pertinent node in T_i. Clearly each leaf has one descendant leaf. In T_1, the only node which is not a leaf is the P-node corresponding to vertex 1. Thus the number of descendant leaves of this P-node is the number of edges incident out of vertex 1 in G. We determine the number of descendant leaves of any node in T_i, $2 \leq i \leq n-2$, from the tree T_{i-1} as follows.

Assume that the number of descendant leaves of each node in T_{i-1} is known. During the processing of T_{i-1} we may delete some leaves from it to make it reducible. Procedure DELETE-NODES(T_{i-1}) also updates the number of descendant leaves of the nodes in T_{i-1}. Thus in T^*_{i-1} the correct number of descendant leaves for each node is known. Let $E_i = \{(j_1, i), (j_2, i), \cdots, (j_k, i)\}$ be the set of edges entering vertex i in the planar subgraph obtained from G. In T^*_{i-1} the leaves corresponding to the edges in E_i appear as children of the same node, say X. Since these leaves are removed from T^*_{i-1} to form T_i, the number of descendant leaves of the nodes corresponding to the vertices $j_1, j_2, \cdots, j_k$, if they are present in T_i, should be decreased by one and the number of descendant leaves of node X and its ancestors in T_i should be decreased by in-deg(i). Moreover, we construct T'_i from T^*_{i-1} by adding a P-node corresponding to vertex i with leaves corresponding to the edges incident out of vertex i in G as its children. Clearly, the number of descendant leaves of this P-node is equal to out-deg(i) in G. Since this node is made a child of node X, the number of descendant leaves of nodes X and all its ancestors in T_i should be increased by out-deg(i). Thus for node X and for each one of its ancestors in T_i, the net increase in the number of descendant leaves is (out-deg(i)–in-deg(i)). The procedure which performs this updating will be referred to as UPDATE-DESCENDANTS(T_i).

Lemma 4: The total cost of procedure UPDATE-DESCENDANTS(T_i) for all $2 \leq i \leq n-2$ is $O(n^2)$. //

Now we present our planarization algorithm which uses the procedures described so far.

```
procedure PLANARIZE(G);
comment procedure PLANARIZE determines the set
        of edges E'_3 = φ, E'_4, ··· , E'_{n-1} to be
        removed from a nonplanar graph G to obtain a spanning planar subgraph G_p.
begin
  {DESCENDANT-LEAVES(X) denotes the number of descendant leaves of node X}
  construct the initial PQ-tree T_1 = T*_1;
  DESCENDANT-LEAVES(1) := out-deg(1);
  for each leaf X corresponding to an edge in E_2 do
    DESCENDANT-LEAVES(X) := 1;
  for i := 2 to n-2 do
    begin
      initialize E'_{i+1} to be empty;
      construct the PQ-tree T_i from T*_{i-1};
      UPDATE-DESCENDANTS(T_i);
      for the P-node X corresponding to vertex i do
        DESCENDANT-LEAVES(X) := out-deg(i);
      for each leaf X corresponding to an edge line in E_{i+1} do
        DESCENDANT-LEAVES(X) := 1;
      BUBBLE-UP(T_i);
      COMPUTE1(T_i);
      if min{h, a} for the pertinent root is not zero
        then begin
          make the pertinent root Types H or A corresponding to the minimum of h and a;
          DELETE-NODES(T_i)
        end;
```

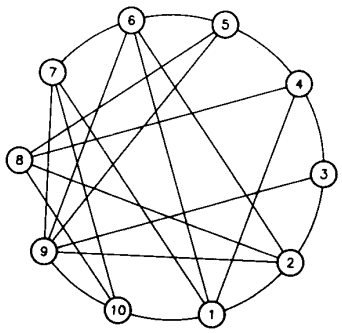


Fig. 1. Nonplanar s-t graph G.

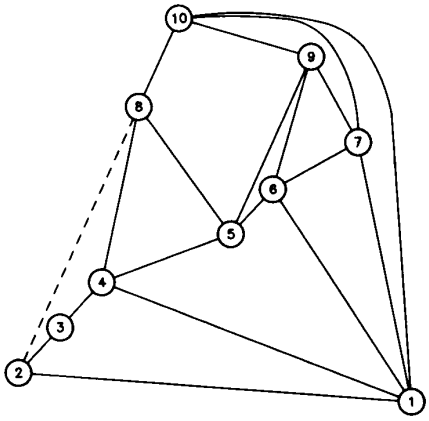


Fig. 2. Planar subgraph G_p.

reduce T_i to obtain T_i^*
end
end PLANARIZE;

Theorem 4: Procedure PLANARIZE determines a spanning planar subgraph of the nonplanar graph G in $O(n^2)$ time and $O(m + n)$ space. //

As an example, we applied our graph-planarization algorithm on the nonplanar graph G shown in Fig. 1. Our algorithm determines $E_6' = \{(2, 6)\}$, $E_8' = \{(2, 8)\}$, and $E_9' = \{(2, 9), (3, 9)\}$ as the sets of edges to be removed from G to planarize it and the spanning planar subgraph G_p is shown in Fig. 2. From this figure we can easily see that the planar subgraph obtained is not maximally planar, since the edge $(2, 8)$ in E_8' can be added to this embedding without affecting the planarity of the resultant graph. Thus the spanning subgraph determined by procedure PLANARIZE may not be maximally planar.

V. A Maximal Planarization Algorithm

For a given a nonplanar graph G, let G_p be a spanning planar subgraph of G obtained by the procedure PLANARIZE described in the previous section. Our interest in this section is to study the problem of constructing a maximal planar subgraph of G which contains G For a successful application of Lempel, Even, and Cederbaum's algorithm for determining the required maximal planar subgraph, it is necessary that G_p have an st-numbering. This requirement necessitates that we assume that G_p is biconnected, since a graph which is not biconnected may not have an st-numbering. Note that, in general, the planar subgraph produced by procedure PLANARIZE may not be connected.

With the assumption that G_p is biconnected, we now proceed to develop an $O(n^2)$ algorithm to construct a maximal planar subgraph of G which contains G_p. Let $E_3' = \phi$, E_4', $\cdots$, E_{n-1}' be the sets of edges removed by procedure PLANARIZE to obtain G_p. Since more than one maximal planar subgraphs containing G_p may exist, our aim will be to attempt to maximize the number of edges in the required graph. Thus, we attempt to add to G_p as many edges as possible from the sets E_3', E_4', $\cdots$, E_{n-1}' without affecting the planarity of the resultant graph. Our approach to maximally planarize G_p is to start with G and construct its PQ-trees. After constructing a PQ-tree, say T_i, we make it reducible by deleting a minimum number of leaves representing the edges in E_{i+1}'. (Note that T_i will become reducible if all the leaves from the set E_{i+1}' are deleted from T_i.) This can be easily done by computing the $[w, h, a]$ number of the pertinent nodes in T_i. Let $T_i(G_p)$ denote the smallest subtree of T_i whose frontier contains all the pertinent leaves from G_p. Since we would like to include G_p in the final maximal planar subgraph, we take care, during the reduction of T_i, not to make any node in $T_i(G_p)$ Type A except its root. This would ensure that the bottom-up reduction process proceeds at least up to the root of $T_i(G_p)$ and possibly beyond. Also, we note that, while computing the $[w, h, a]$ numbers we ignore the presence of empty leaves from $G - G_p$. In the following, the leaves in T_i corresponding to the edges in E_{i+1}' will be called the *new pertinent leaves* of T_i and the other pertinent leaves of T_i (corresponding to the edges entering vertex $i + 1$ in G_p) will be called *preferred leaves*.

A node is called full if its frontier has no empty leaf from G_p; it is empty if its frontier has only empty leaves from G_p; otherwise it is partial. We call pertinent node X a *preferred node* if it has some of the preferred leaves among its descendants. If X is not preferred, then it may either be retained in the reducible T_i or it may be deleted along with all its descendants to make T_i reducible.

The formulas for computing the $[w, h, a]$ numbers of pertinent nodes are the same as those given in Section IV. So, in the following we only consider the essential features of our Type assignment policy which guarantees that G_p is included in the final maximal planar graph. Let X be a pertinent node in T_i.

Case 1: X is a Partial P-Node

In this case X can have at most two partial preferred children.

(a) X has no partial preferred children

If X is the root of $T_i(G_p)$ or its ancestor in T_i, then it can be included in the reducible T_i by making it Type A or Type H. Otherwise it can be included only by making

it Type H. Note that in the latter situation we should set h-child2(X) and a-child(X) empty.

(b) X has exactly one partial preferred child

The partial preferred child has to be retained in the reducible T_i and it becomes h-child1(X). Moreover, if X is the root of $T_i(G_p)$ or its ancestor in T_i, then it can be included by making it Types A or H. Otherwise it can be included only by making it Type H. Note that if X is the root of $T_i(G_p)$, then none of its children can be made Type A and so a-child(X) will be empty in this case.

(c) X has two partial preferred children

In this case X is the pertinent root of the reducible T_i. One of the practical preferred children of X becomes h-child1(X) and the other becomes h-child2(X). We set a-child(X) empty and remember that the root is processed.

Case 2: X is a Partial Q-Node

In this case, all the preferred pertinent children of X should appear consecutively. We first traverse these children from the leftmost child towards the rightmost child and determine the maximal consecutive sequence $P'(X)$ with the properties

(i) $P'(X)$ contains all the preferred children of X;

(ii) only the leftmost node and/or the rightmost node in $P'(X)$ may be partial; that is, its frontier may contain an empty leaf from G_p; and

(iii) all the other nodes in $P'(X)$ are full; that is, the frontier of each of these nodes contains no empty leaf belonging to G_p.

Now X can be made Type H only when one of the following happens:

(i) $P'(X)$ appears at the left end of X and the leftmost node in $P'(X)$ has in its frontier no empty leaf from G_p. Then we set $P_L(X) = P'(X)$,

(ii) $P'(X)$ appears at the right end of X and the rightmost node in $P'(X)$ has in its frontier no empty leaf from G_p. Then we set $P_R(X) = P'(X)$.

(Note that $P_L(X)$ and $P_R(X)$ are as defined in Section IV.) In both the above cases, we set h-child1(X) to the leftmost node in $P'(X)$ and compute the h number for X.

Suppose neither of the above conditions is satisfied. Then if $P'(X)$ contains only one node, this node should be made Types H or A corresponding to the minimum of h and a. If $P'(X)$ is made Type A, then the only node in $P'(X)$ becomes a-child(X); otherwise it becomes h-child1(X). If $P'(X)$ has more than one node, then we set h-child2(X) to the leftmost node in $P'(X)$ and compute the a number for X. We also remember in this case that the pertinent root is processed.

Processing the pertinent nodes of T_i up to the pertinent root using the above ideas and using the formulas of Section IV, we can determine the $[w, h, a]$ numbers of all the pertinent nodes in T_i. The procedure which achieves this will be referred to COMPUTE2(T_i).

Before we proceed further, we wish to note that some of the nodes in $P'(X)$ and/or their descendants may have only empty leaves from $G - G_p$ in their frontier. While making T_i reducible during procedure PLANARIZE, these leaves must have caused deletion of certain pertinent leaves from T_i; but they themselves got deleted at a later step in the execution of PLANARIZE. Since they are no longer in G_p, it becomes possible to add to G_p some new pertinent leaves thereby making the graph constructed thus far maximal. This is exactly what we do while identifying the maximal sequence $P'(X)$. Thus procedure COMPUTE2(T_i) identifies a maximal set of new pertinent leaves to be added to G_p without causing nonplanarity. More important is the fact that the PQ-tree reduction procedure would ensure that those empty leaves from $G - G_p$ which appear in the frontier of $P'(X)$ and which made possible addition of certain new pertinent leaves will not be present in subsequent PQ-trees. Thus these empty leaves which would normally be new pertinent leaves in subsequent PQ-trees will not even be present in these PQ-trees and therefore will not be available for addition to G_p. Thus future addition of new pertinent leaves to the planar subgraph already constructed at step i would not cause nonplanarity.

The following lemma gives the total cost of procedure COMPUTE2(T_i), $2 \leq i \leq n - 2$.

Lemma 5: The $[w, h, a]$ numbers of the pertinent nodes in all the PQ-trees can be computed in $O(n^2)$ time using procedure COMPUTE2(T_i), $2 \leq i \leq n - 2$.

Having computed the $[w, h, a]$ numbers for the pertinent nodes in T_i, we can obtain a reducible T_i by traversing the pertinent subtree top-down from the pertinent root using procedure DELETE-NODES(T_i) (see Section IV). Note that while applying Procedure DELETE-NODES(T_i), the term "full", "empty" and "partial" should be understood as defined at the beginning of this section. During this processing some of the new pertinent leaves in T_i may not be processed at all. Clearly, such pertinent leaves should be deleted from T_i to make it reducible and the edges corresponding to these leaves should also be removed from the nonplanar graph G.

Recall that procedure COMPUTE2(T_i) requires that we are able to determine whether the frontier of a node X has an empty leaf from G_p. Suppose the frontier of X in G_p has only empty leaves. Then the procedure BUBBLE-UP(T_i) of Section IV will not even visit this node because this procedure traverses only the pruned pertinent subtree of T_i which does not include empty leaves. In order to overcome this problem, we modify BUBBLE-UP(T_i) so that in addition to traversing all the nodes in the pruned pertinent subtree of T_i, it also traverses every node whose frontier contains at least one empty leaf from G_p. Interestingly, as we show now, this modification does not affect the complexity of the bubble up process (given in Lemma 3), and we refer to this modified procedure as MODIFIED-BUBBLE-UP(T_i).

Lemma 6: The total cost of MODIFIED-BUBBLE-UP(T_i) for all $2 \leq i \leq n - 2$ is $O(n^2)$.

Proof: Let $n_p(T_i)$ be the total number of leaves of T_i belonging to G_p and let UNARY(T_i) be the number of unary nodes (nodes of degree one) except leaves traversed by MODIFIED-BUBBLE-UP(T_i). Then the

cost of MODIFIED-BUBBLE-UP(T_i) is $O(n_p(T_i)$ + UNARY(T_i)). But $n_p(T_i) = O(m_p)$ where m_p is the number of edges in G_p and UNARY(T_i) $= O(n)$. Since G_p is planar, $m_p = O(n)$. Hence cost of MODIFIED-BUBBLE-UP(T_i) is $O(n)$. Summing up this cost for all T_i, $2 \leq i \leq n-2$, we get the result. //

Processing the PQ-trees $T_2, T_3, \cdots T_{n-2}$ using the different procedures described above we obtain a maximal planar subgraph of the nonplanar graph G.

```
procedure MAXIMAL-PLANARIZE(G);
comment procedure MAXIMAL-PLANARIZE determines a maximal planar subgraph of the
        nonplanar graph G. This procedure uses the spanning planar subgraph obtained by
        procedure PLANARIZE. It is assumed that the spanning planar subgraph is bi-
        connected.
begin
  {Determine the spanning planar subgraph}
  PLANARIZE(G);
  {Maximally planarize the spanning planar subgraph}
  construct the initial PQ-tree T1 = T1*;
  DESCENDANT-LEAVES(1) := out-deg(1);
  for each leaf X corresponding to an edge in E2 do
    DESCENDANT-LEAVES(X) := 1;
  for i := 2 to n-2 do
    begin
      construct the PQ-tree Ti from Ti-1*;
    UPDATE-DESCENDANTS(Ti);
    for the P-node X corresponding to vertex i do
      DESCENDANT-LEAVES(X) := out-deg(i);
    for each leaf X corresponding to an edge in Ei+1
      do
      DESCENDANT-LEAVES(X) := 1;
    MODIFIED-BUBBLE-UP(Ti);
    COMPUTE2(Ti);
    if min{h, a} for the pertinent root is not zero
      then begin
        make the pertinent root Types H or A corresponding to the minimum of h and a;
        DELETE-NODES(Ti);
        delete the new pertinent leaves which are not processed from Ti
      end;
    reduce Ti and obtain Ti*
  end
end MAXIMAL-PLANARIZE;
```

Clearly, when algorithm MAXIMAL-PLANARIZE is applied on a nonplanar graph G, treating the edges of a planar subgraph G_p as preferred edges, it produces a planar subgraph G' containing G_p. We now prove that G' is indeed a maximal planar subgraph of G.

Suppose we apply algorithm MAXIMAL-PLANARIZE on G treating G' as the preferred graph. Then in the following theorem $T_1', T_2', \cdots, T_{n-1}'$ will refer to the PQ-trees which are generated as the algorithm progresses.

Theorem 5: Algorithm MAXIMAL-PLANARIZE when applied on a nonplanar graph G, treating a biconnected planar subgraph G_p as the preferred graph, produces a maximal planar graph G' which contains G_p.

Proof: As we noted before, G' is planar and contains G_p. So, we need only prove that G' is a maximal planar subgraph of G. Assume the contrary. Then there exists an edge $e = (j, k) \in G$, $j < k$, such that $e \notin G'$ and $G' \cup \{e\}$ is planar. Among all such edges select the one for which k is minimum and let this edge be $e = (j, i+1)$. This means that the leaf in T_i' representing e is a new pertinent one with respect to G'. Note that in T_i this leaf was also new pertinent with respect to G_p, and it was not added while procedure MAXIMAL-PLANARIZE constructed G' starting from G_p. Furthermore, T_i' is isomorphic to the corresponding PQ-tree T_i generated when G_p is treated as the preferred graph. Also, since $G_p \subseteq G'$ all the preferred pertinent leaves of T_i will also be preferred pertinent leaves in T_i'. Furthermore, some of the new pertinent leaves in T_i may become preferred ones in T_i'.

Since $G' \cup \{e\}$ is planar, through a sequence of Q-node flippings and permutations of children of P-nodes, T_i' can be converted into a tree T_i'' such that its frontier contains a maximal sequence L' with the following properties.

(i) Except the first and/or the last leaf in L', none of the others are empty leaves from G'. (Note: this means that except the first and/or the last leaf in L', none of the others are empty leaves from G_p.)

(ii) L' contains all the preferred pertinent leaves of G'. (Note: some of these leaves are, possibly, present in T_i as new pertinent ones; the remaining leaves are preferred ones in T_i.)

(iii) L' contains the new pertinent leaf e and possibly a few more new pertinent ones. (Note: these leaves are all new pertinent in T_i too.)

(iv) Empty leaves from $G - G'$ may be added to L' to make it maximal. (Note: these leaves belong to $G - G_p$ too, since $G_p \subseteq G'$.)

Since T_i' and T_i are isomorphic, it follows that T_i can also be converted into T_i''. Furthermore, the above observations imply that L' satisfies all the properties required to be satisfied by the sequence $P'(X)$ (where X is the pertinent root of T_i) identified by COMPUTE2(T_i). But, $P'(X)$ is a proper subset of L' since $e \notin P'(X)$. This is a contradiction because $P'(X)$ is not maximal as required. //

Theorem 6: Procedure MAXIMAL-PLANARIZE is of complexity $O(n^2)$ in time and $O(m+n)$ in space. //

As an example, applying procedure MAXIMAL-PLANARIZE on the planar subgraph G_p shown in Fig. 2, we obtain the maximal planar subgraph shown in Fig. 3. We have implemented procedure MAXIMAL-PLANARIZE in PASCAL and tested it on several nonplanar graphs using a CDC Cyber 170. In Table I we show the number of edges removed by procedure PLANARIZE and the number of edges added by procedure MAXIMAL-

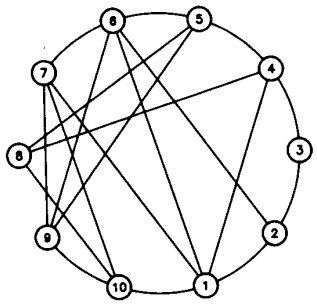


Fig. 3. Maximal planar subgraph of the graph G (Fig. 1).

TABLE I

Graph	Number of vertices	Number of edges	Number of edges removed by procedure PLANARIZE	Number of edges added by procedure MAXIMAL-PLANARIZE	Execution Time (sec)
G_1	10	35	21	3	0.263
G_2	20	60	24	0	0.672
G_3	30	95	42	5	0.976
G_4	40	125	39	2	1.321
G_5	50	150	47	4	1.985
G_6	60	180	53	3	3.126
G_7	70	225	57	0	4.795
G_8	80	250	78	7	5.013
G_9	90	300	103	5	6.792
G_{10}	100	350	124	8	7.863

PLANARIZE for some of the test graphs. It can be seen from Table I that procedure MAXIMAL-PLANARIZE adds only a very small number of edges to the spanning planar subgraph. We have also shown in Table I the execution time required to find the maximal planar subgraph for these graphs.

Given an n-vertex biconnected graph G_p, it is easy to see that by applying algorithm MAXIMAL-PLANARIZE on the n-vertex complete graph K_n and treating G_p as the preferred graph, we can construct a maximal planar subgraph which contains G_p. Thus, we can maximally planarize any biconnected planar graph using algorithm MAXIMAL-PLANARIZE and we have the following theorem.

Theorem 7: Maximal planarization of an n-vertex biconnected planar graph can be achieved in $O(n^2)$ time. //

We now return to the question of the maximal planarization problem when the subgraph G_p of G produced by algorithm PLANARIZE is not biconnected. In this case, the *st*-numbering of G which we used before applying algorithm PLANARIZE on the graph G may not be an *st*-numbering for G_p because a graph which is not biconnected may not have an *st*-numbering. Since the embedding produced by the LEC algorithm assumes that the vertices are placed at different levels dictated by the *st*-numbers, it follows that algorithm MAXIMAL-PLANARIZE when applied on G_p produces a maximal planar subgraph (containing G_p) of G which is consistent with the embedding of G_p as dictated by the original *st*-numbers. But such a subgraph may not be a maximal subgraph containing G_p. So, one way to proceed further for the construction of the required maximal planar graph is to first obtain the biconnected components of G_p. Then we should examine each edge for possible addition to one or more biconnected components of G_p. But, before doing so, we should obtain the new *st*-numbering for the graph obtained by adding the new edge to G_p. The complexity of such a maximal planarization algorithm will be $O(mn)$, which is the same as that of the straightforward algorithm.

VI. Summary and Conclusion

In this paper we present two $O(n^2)$ planarization algorithms—PLANARIZE and MAXIMAL-PLANARIZE. These algorithms are based on Lempel, Even, and Cederbaum's planarity testing algorithm [9] and its implementation using *PQ*-trees [8]. Algorithm PLANARIZE is for the construction of a spanning planar subgraph of an n-vertex nonplanar graph. This algorithm proceeds by embedding one vertex at a time and, at each step, adds the maximum number of edges possible without creating nonplanarity of the resultant graph. Given a biconnected spanning planar subgraph G_p of a nonplanar graph G, algorithm MAXIMAL-PLANARIZE constructs a maximal planar subgraph of G which contains G_p. This latter algorithm can also be used to maximally planarize a biconnected planar graph.

We conclude by pointing out that no non-trivial planarization algorithm of complexity $O(n^2)$ have been reported in the literature. The $O(mn)$ algorithms of [5] and [7] do not seem to lend themselves to easy modifications resulting in such planarization algorithms. Furthermore, no $O(n^2)$ algorithm for maximal planarization of a biconnected planar graph has been reported before in the literature. We have also pointed out how the algorithm PLANARIZE and MAXIMAL-PLANARIZE can be used to construct a maximal planar subgraph even when the graph produced by PLANARIZE is not biconnected. Though the worst-case complexity of such a maximal planarization algorithm will be the same as that of the algorithm in [5], we expect this algorithm to require, on the average, less computation time since construction of G_p requires only $O(n^2)$ time and while constructing G_p we have attempted to include as many edges as possible.

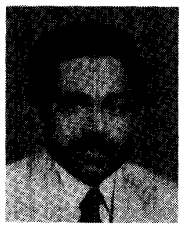
The question now remains whether we can design an $O(n^2)$ algorithm to construct a maximal planar subgraph of a nonplanar graph. Such an algorithm will be possible provided we can find an $O(n^2)$ algorithm to construct a spanning biconnected planar subgraph of a given nonplanar graph.

References

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

[2] G. J. Fisher and O. Wing, "Computer recognition and extraction of planar graphs from the incidence matrix," *IEEE Trans. Circuit Theory*, vol. CT-13, pp. 154-163, June 1966.
[3] K. Pasedach, "Criterion and algorithms for determination of bipartite subgraphs and their application to planarization of graphs," in *Graphen-Sprachen und Algorithmen auf Graphen*, Germany: Carl Hanser Verlag, pp. 175-183, 1976.
[4] M. Marek-Sadowska, "Planarization algorithm for integrated circuits engineering," in *Proc. 1978 IEEE Int. Symp. on Circuits and Systems*, pp. 919-923.
[5] T. Chiba, I. Nishioka, and I. Shirakawa, "An algorithm of maximal planarization of graphs," in *Proc. 1979 IEEE Int. Symp. on Circuits and Systems*, pp. 649-652.
[6] J. Hopcroft and R. Tarjan, "Efficient planarity testing," *J. Ass. Comput. Mach.*, vol. 21, no. 4, pp. 549-568, Oct. 1974.
[7] T. Ozawa and H. Takahashi, "A graph-planarization algorithm and its application to random graphs," in *Graph Theory and Algorithms*, Springer-Verlag Lecture Notes in Computer Science, vol. 108, pp. 95-107, 1981.
[8] 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.
[9] A. Lempel, S. Even, and I. Cederbaum, "An algorithm for planarity testing of graphs," in *Int. Symp. on Theory of Graphs*, Rome, Italy, July 1966. P. Rosenstiehl (Ed.), Gordon and Breach, New York, pp. 215-232, 1967.
[10] S. Even, *Graph Algorithms*. Potomac, MD: Computer Sci., Maryland, 1979.
[11] R. Jayakumar, K. Thulasiraman, and M. N. S. Swamy, "On maximal planarization of non-planar graphs," *IEEE Trans. Circuits Syst.*, vol. CAS-33, no. 8, August 1986, pp. 843-844.
[12] ——, "$O(n^2)$ algorithms for graph planarization," Tech. Rep. CSD-88-01, Dep. of Comp. Sci., Concordia Univ., Montreal, Canada, June 1988.
[13] S. Even and R. E. Tarjan, "Computing an *st*-numbering," *Theo. Comp. Sci.*, vol. 2, pp. 339-344, 1976.
[14] J. Ebert, "St-ordering the vertices of biconnected graphs," *Computing*, vol. 30, pp. 19-33, 1983.
[15] R. Jayakumar, K. Thulasiraman, and M. N. S. Swamy, "Planar embedding: linear time algorithms for vertex placement and edge ordering," *IEEE Trans. Circuits Syst.*, vol. CAS-35, pp. 334-344, Mar. 1988.

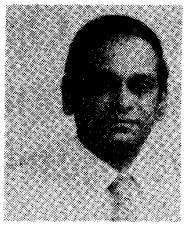
*



R. Jayakumar received the B.E. (hons) 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 has been an Assistant Professor of Computer Science at Concordia University. His research interests are in VLSI algorithms and architectures, fault-tolerant VLSI systems, VLSI design automation and graph theory and graph algorithms.

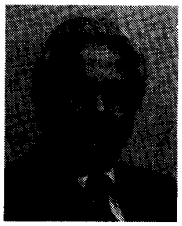
Dr. Jayakumar is a member of the Association for Computing Machinery.



K. Thulasiraman (M'72-SM'84) received the Bachelor's and Master's degrees in electrical engineering from the University of Madras, Madras, India, in 1963 and 1965, respectively, and the Ph.D. degree in electrical engineering from the Indian Institute of Technology, Madras, in 1968.

He joined the Indian Institute of Technology, Madras in 1965, where he was associated with the Department of Electrical Engineering from 1965 to 1973 and with the Department of Computer Science from 1973 to 1981. After serving for a year (1981-1982) at the Department of Electrical Engineering, Technical University of Nova Scotia, Halifax, Canada, he joined Concordia University, Montreal, as Professor at the Department of Mechanical Engineering where he was involved in the development of programs in Industrial Engineering at the undergraduate and graduate levels. Since 1984 he has been with the Department of Electrical and Computer Engineering at Concordia University. Earlier, he had held visiting positions at Concordia University during the periods of 1970-1972, 1975-1976, and 1979-1980. He has published over 50 technical papers on different aspects of Electrical Network Theory, Graph Theory and Design and Analysis of Algorithms. He has also coauthored the book, *Graphs, Networks and Algorithms* (New York: Wiley-Interscience, 1981). He was awarded a Senior Fellowship by the Japan Society for Promotion of Science. Under this fellowship he will be visiting the Tokyo Institute of Technology, Tokyo during March-July 1988. His current research interests are in network and systems theory, graph theory, parallel and distributed computations, operations research and computational graph theory with applications in VLSI design automation, computer networks, communication network planning, etc.

*



M. N. S. Swamy (S'59-M'62-SM'74-F'80) received the B.Sc. (hons) 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.