A NEW PLANARITY TEST

(To appear in Theoretical Computer Science)

Wei-Kuan Shih[1] and Wen-Lian Hsu[2]

ABSTRACT

Given an undirected graph, the planarity testing problem is to determine whether the graph can be drawn in the plane without any crossing edges. Linear time planarity testing algorithms have previously been designed by Hopcroft and Tarjan, and by Booth and Lueker. However, their approaches are quite involved. Several other approaches have also been developed for simplifying the planariy test. In this paper, we developed a very simple linear time testing algorithm based only on a depth-first search tree. When the given graph is not planar, our algorithm immediately produces explicit Kuratowski's subgraphs. A new data structure, *PC*-trees, is introduced, which can be viewed as abstract subembeddings of actual planar embeddings. A graph-reduction technique is adopted so that the embeddings for the planar biconnected components constructed at each iteration never have to be changed. The recognition and embedding are actually done simultaneously in our algorithm[1]. The implementation of our algorithm is quite straightforward.

1. Introduction

Given an undirected graph, the planarity testing problem is to determine whether there exists a clockwise edge ordering around each vertex such that the graph can be drawn in the plane without any crossing edges. Planarity test is a very useful tool in many disciplines of computer science. Linear time planarity testing algorithm was first established by Hopcroft and Tarjan [1974] based on a "*path addition* approach". A "*vertex addition* approach", originally developed by Lempel, Even and Cederbaum [1967], was later improved by Booth and Lueker [1976] to run in linear time using a data structure called a "*PQ*-tree". Both of these approaches are quite involved. Furthermore, the recognition and the embedding are two separate algorithms in both approaches (Chiba et al [1985]). Several other approaches have also been developed for simplifying the planariy test (see for example [2,8,9,10]) and the embedding algorithm [7]. In this paper, we developed a very simple linear time testing algorithm based only on a depth-first search tree. When the given graph is not planar, our algorithm immediately produces explicit Kuratowski's subgraphs. A new data structure, *PC*-trees, is introduced. A graph-reduction technique is adopted so that the embeddings for the planar biconnected components constructed at each iteration never have to be changed. The recognition and embedding are actually done simultaneously in our algorithm. The implementation of our algorithm is quite straightforward.

註解: Page: 1

To simplify our discussion, assume the given graph G is biconnected. Let n be the number of vertices of the graph G. Construct a depth-first search tree T for G. Note that every non-tree edge of G must be a back edge from a vertex to one of its ancestors. Let 1, ..., n be the order resulting from a postorder traversal of T. So the order of a child is always less than that of its parent. Denote the subtree of T with root i by T_i. Initially, we shall include all edges of T in the embedding. Then at iteration i we add all back edges from the descendants to node i and update the embedding. Whenever a 2-connected subgraph is created, we shall create its internal embedding and use a subset of vertices in its boundary cycle as representatives for future embedding.

註解: Page: 1

To facilitate the implementation of our algorithm, we introduce the notion of *PC*-trees: a tree is a *PC*-tree if its nodes can be divided into two types: *P*-nodes and *C*-nodes, where the neighbors of a *P*-node (denoted by a circle) can be permuted arbitrarily and the neighbors of a *C*-node (denoted by a double circle) observe a cyclic order which can only be reversed. We shall use a *PC*-tree to represent the partial embedding of the planar graph in which a *P*-node denotes a regular vertex of the graph and a *C*-node denotes a biconnected component with its representative vertices (to be defined later) as neighbors. In essence, *PC*-trees are similar to *PQ*-

---

[1] Department of Computer Science, National Tsing Hua University, Hsin-Chu, Taiwan, ROC. Email: wshih@cs.nthu.edu.tw

[2] Institute of Information Science, Academia Sinica, Taipei, Taiwan, ROC. Email: hsu@iis.sinica.edu.tw

trees. But *PC*-trees can be applied to unrooted trees. Furthermore, since *PC*-trees can be viewed as abstract subembeddings of actual planar embeddings, they are more natural to represent planar graphs .

2. Our Modified Vertex Addition Approach: the Simple Case

In the remainder of this paper we assume the given graph is planar. Denote the largest neighbor of a node i by $h(i)$. We shall assign the label $b(v)$ to each node v of tree T as follows: assign to each leaf u of T the label $h(u)$; assign to each internal node i the label

$b(i) = \max (h(i) , \max \{ b(v) \mid v \text{ is a child of } i \})$

It is easy to verify that labels assigned this way satisfy

(2.1) $b(i) = \max \{ h(v) \mid v \text{ is a node in } T_i \}$.

Because the graph G is biconnected we must have $b(i) > i$ for every internal node i of T except the root (for otherwise, i would be an articulation vertex of G). Sort the children of each node of T according to the ascending order of their labels. At each iteration j, we consider the embedding of the back edges from the descendants to j and revise the tree accordingly. Denote the revised tree at the end of iteration j by T^j.

Let i be the first iteration that there is a back edge from a descendent to i. Let (i',i) be such a back edge, then (i',i) together with the unique path from i to i' in T form a cycle. Thus, this is the first iteration that a 2-connected subgraph is formed. We shall describe the partial embedding for this iteration in this section. Note that it suffices to describe the embedding of i with each children subtree independently. Hence, consider a children subtree of i with root r.

Define a *terminal node* in T_r as a node t that satisfies (1) $b(t) > i$; (2) either t is adjacent to i or it has a descendent labeled i; (3) no other descendent of t satisfies both (1) and (2). A subtree T_j is said to be an *i-tree* if every node in T_j has the label i; it is an *i^*-tree* if every node has a label greater than i.

In Lemmas 2.2 and 2.3, let v be a node in T_r.

Lemma 2.2. *If $b(v) = i$, then T_v is an i-tree.*

Proof. By assumption, there is no back edge to a node less than i, hence there is no node with label less than i. Therefore, every leaf of T_v has the label i and this forces every node of T_v to have a label no less than i. Since none of the node in T_r can have a label > i, the lemma follows. z

Lemma 2.3. *If $b(v) > i$ but T_v is not an i^*-tree, then there exists a terminal node in T_v .*

Proof. There must exist a leaf v' of T_v with $b(v') = i$. Let w be the first node along the unique tree path from v' to v with $b(w) > i$. Then w satisfies conditions (1) and (2) above. Let w' be the smallest node satisfying conditions (1) and (2) in T_v. Then w' is a terminal node. z

Lemma 2.4. *Let u be a terminal node of T_r. Then, for each child v of u, T_v is an i-tree if $b(v) = i$; T_v is an i^*-tree if $b(v) > i$.*

Proof. If $b(v) = i$, then T_v is an i-tree by Lemma 2.2. Assume $b(v) > i$. If T_v is not an i^*-tree, then by Lemma 2.3, T_v would contain another terminal node, contradictory to the fact that u is a terminal node. z

Theorem 2.5. *Let G be a planar graph and T_r the subtree defined above. Then there are at most two terminal nodes in T_r .*

Proof. Assume to the contrary there are three terminal nodes, say i_1, i_2 and i_3 (none of them can be a descendent of the other). Then each of them has a descendant (could be themselves) with a neighbor larger than i. Note that such neighbors must lie on the unique path from i to root n. *Choose any three such neighbors among* $\{i+1,...,n\}$ for the descendants of i_1, i_2 and i_3 in their i^*-subtrees, respectively and let the medium (not necessarily distinct) of these three neighbors be t. Let the smallest of the three least common ancestors of $\{i_1,i_2\}$, $\{i_1,i_3\}$ and $\{i_2,i_3\}$ be w. Choose three descendants (could be themselves) of i_1, i_2 and i_3 in their i-subtrees, respectively that are adjacent to i. Then a subgraph homeomorphic to $K_{3,3}$ can be found as shown in Figure 2.1, where the dotted lines denote paths.

z

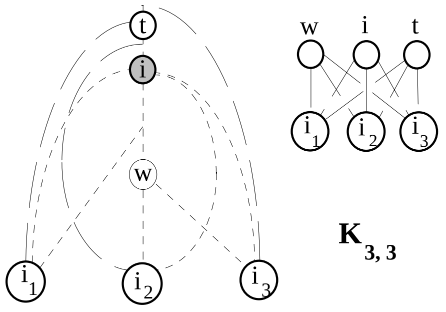


Figure 2.1. A forbidden structure

Lemma 2.6. *Suppose there are two terminal nodes u and u' in T_r. Let P be the unique path in T from u to u'. Let m be the least common ancestor of u and u' in T. Let P' be the unique path from m to r. Let S = { v | v is a child of a node in P, but v is not in P}. Let S' = { v | v is a child of a node in P'-{m}, but v is not in P'}* (*note that when m = r, S' is empty*). *Then, for each node v in S, T_v is either an i-tree or an i*-tree* (*see the example in Figure 2.2*) *and, for each node v in S', T_v is an i-tree.*

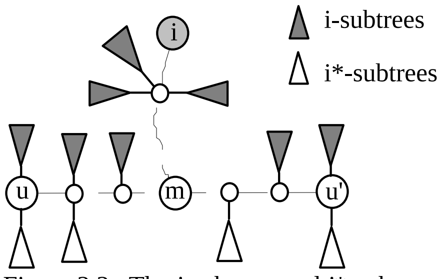


Figure 2.2. The i-subtrees and i*-subtrees

Proof. Let v be a node in S with $b(v') > i$, if T_v is not an i*-tree, then by Lemma 2.3, T_v would contain a third terminal node, a contradiction.

Now, assume $S'' \neq \varnothing$ and there is a node v' in S' with $b(v') > i$. Then node v' has a descendant w with $h(w) > i$. The same holds for both u and u'. Through these descendants one can find three paths from v', u and u', respectively, to neighbors greater than i. Let the median node of these three neighbors (not necessarily distinct) be t. Then a subgraph homeomorphic to $K_{3,3}$ can be found as shown in Figure 3.3. Þ

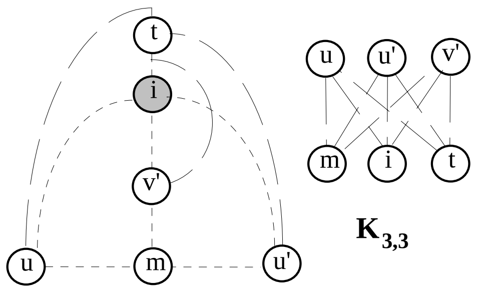


Figure 2.3. A forbidden structure in S'

Corollary 2.7. *Suppose there is only one terminal node u in T_r. Let P be the unique path in T_r from u to r. Let S = { v | v is a child of a node in P, but v is not in P}. Then, for every node v in S, T_v is either an i-tree or an i*-tree.*

Proof. By Lemma 2.3, every child v of u in S satisfies that T_v is either an i-tree or an i*-tree. Let u' be the first node along path P from u to r violating this condition. Then there is a child v of u' with $b(v) > i$ and T_v is not an i*-tree. By Lemma 2.3, there exists a terminal node in T_v, contradictory to the fact that u is the only terminal node. z

Lemma 2.8. *Suppose there are two terminal nodes u and u' in T_r. The unique path P between the two terminal nodes u and u' must be on the boundary of any 2-connected components formed by node i and nodes in the subtree T_r.*

Proof. Since i has exactly two neighbors in the boundary cycle of such a component, any such cycle can contain at most two back edges to i. The remaining path on the cycle must be a path Q in the tree whose two end nodes are adjacent to i. It suffices to show that the two terminal nodes must be in Q. Suppose u is not in Q. Let v, v' be children of u with $b(v) = i$, $b(v') > i$. Consider the following two cases:

(1) *u* is in the interior of the component, then *v'* must be in *Q* (since *v'* cannot be in the interior). But then path *Q* must contain only nodes in $T_{v'}$, which is impossible since no nodes in $T_{v'}$ is adjacent to *i*.

(2) *u* is in the exterior of the component, then *v* must be in *Q* (since *v* cannot be in the exterior). But then path *Q* must contain only nodes in T_v. In that case, a child *w* of *u'* with label *i* must be in the interior. Now the tree path from *u* through *u'* to *w* connecting an exterior node *u* to an interior node *w* must contain a node in *Q*, a contradiction. z

Corollary 2.9. *With the same assumptions as those in Corollary 2.7, let u' be the last node along the tree path P from u to r that has a child v in S with b(v) > i (u' could be the same as u). Then the tree path from u to u' must be on the boundary of any 2-connected components formed by node i and nodes in the subtree* T_r.

3. The Creation of C-Nodes and the General Algorithm

In the last section, we have described the labeling structure at the first iteration that some 2-connected component is created. We have discussed the boundary path of the component. For each node j in the unique path P, define its new children to be its children in S whose labels are greater than i. Revise *b*(*j*) to be the maximum over the following numbers: the labels of its new children and *h*(*j*). For any 2-connected component formed by *i* and nodes in T_r, define the *essential nodes* on the boundary cycle to be those with new labels greater than *i*. Since these nodes must be in *P*, they are independent of the selection of the exact boundary cycle of the component. The essential nodes are the only ones on the boundary cycle relevant to future embedding.

Define the *representative boundary cycle* (*RBC*) of this 2-connected component to be a cycle formed by node *i* and those essential nodes whose cyclic order follows their original order in the *RBC*. The *RBC* will be stored as a circular doubly linked list. To distinguish from the original edges of the graph, we shall refer to the connection on the *RBC* as *links*.

The internal embedding will be discussed in the next section. In the revised tree we shall represent the 2-connected component by a *C*-node whose parent is *i* and whose children are the essential nodes as shown in Figure 3.1. The label of the *C*-node is defined to be the maximum of the labels of its children. Since a *C*-node has at least one child it can never be a leaf, a label defined this way naturally satisfies (2.1).

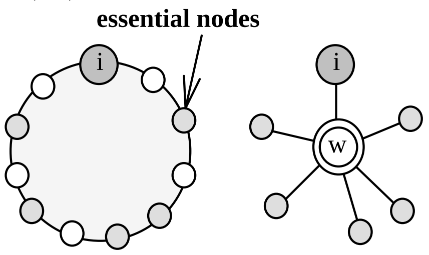
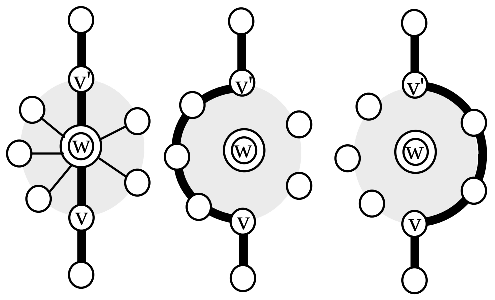


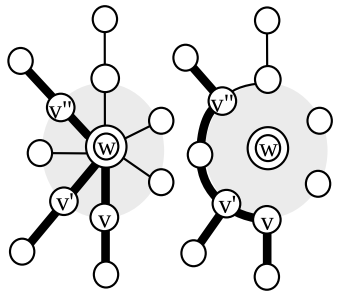
Figure 3.1. The representation of a *C*-node

This concludes our discussion of the first iteration that a back edge emerges. Hence, at the end of this iteration, we shall have *C*-nodes in the revised tree T^i.

In the following we shall discuss our revision of Lemmas and Theorems proved above in the general case when there are *C*-nodes in the revised tree. For ease of reading, some notations will be abused a little. We shall, again, denote the current iteration by *i*; denote a subtree with root *j* in the revised tree by T_j. Notation-wise, the *C*-nodes can be treated just as a regular *P*-node and most of the arguments in the lemmas and proofs will go through as long as the tree paths are interpreted correctly: a tree path of G through a *C*-node w in T^i should be interpreted as a path using a boundary path from *v* to *v'* in its original 2-connected component where *v* and *v'* are the neighbors of *w* in the path; three node disjoint paths emanating from a *C*-node *w* should be interpreted as three node disjoint paths emanating from the boundary path. These are illustrated in Figure 3.2. To depict the paths more clearly, we sometimes draw a *C*-node as a disk with its essential nodes on the boundary cycle.



(i) Interpreting a path through w from v to v'



(ii) Interpreting three paths through w

Figure 3.2. The interpretation of tree paths through a C-node w

The definition of terminal nodes are the same as before. Lemmas 2.2 through Corollary 2.7 and their proofs basically go through for the case of general trees without any changes provided that paths through a *C*-node are interpreted correctly as above. In the proof of Theorem 2.5, we could have three terminal nodes being neighbors of a *C*-node, in which case we would get a subgraph homeomorphic to K_5 as illustrated in Figure 3.3 (provided that two of the neighbors greater than *i* coincide with *t* and the third one is larger).

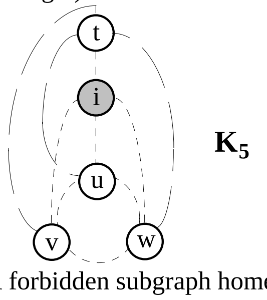


Figure 3.3. A forbidden subgraph homeomorphic to K_5

Let *v*, *v'* be any two nodes in the *RBC* of a *C*-node *w*. Then the children of *w* other than *v*, *v'* are divided into two subsets depending on whether they are on the same side of the cycle divided by *v* and *v'*. Denote these two subsets by $C_1(v,v')$ and $C_2(v,v')$.

Lemma 3.1. *Consider the same assumptions as those in Lemma 2.6. Let w be an intermediate node of P. Let v, v' be the two neighbors of w in P. Then either for each node v in $C_1(v,v')$, T_v is an i-tree; and for each node v in $C_2(v,v')$, T_v is an i*-tree or the other way around* (*as illustrated in Figure 3.4*).

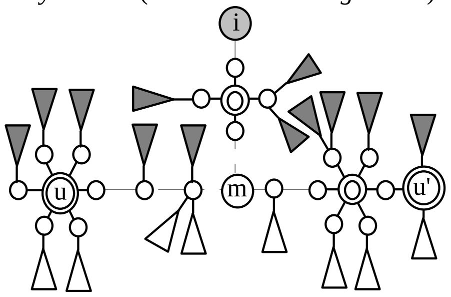


Figure 3.4. *i*-trees and *i**-trees around a *C*-node

Proof. Suppose there exists both *j* and *j'* such that T_j is an *i**-tree and $T_{j'}$ is an *i*-tree. Then we can find a graph homeomorphic to $K_{3,3}$ as shown in Figure 3.5. z

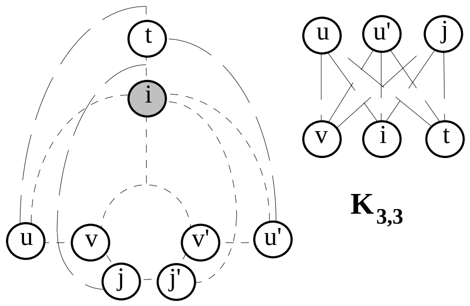


Figure 3.5. A forbidden structure in *C*-nodes

The following lemma shows that children of a terminal *C*-node *w* that give rise to *i*-subtrees must be consecutive on the *RBC* next to the parent of *w*.

Lemma 3.2. *Let a C-node w be a terminal node. Let the parent of w be j. Let j' be a child of w s.t. $T_{j'}$ is an i-tree. Then either for every v in $C_1(j,j')$, T_v is* *an i-tree or for every v in $C_2(j,j')$, T_v is an i-tree (as illustrated in Figure 3.6).*

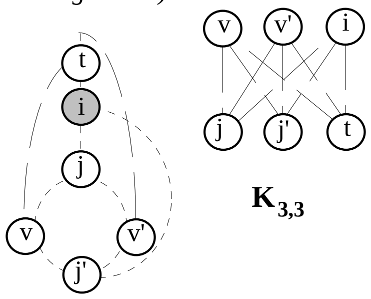


Figure 3.6. A forbidden structure in a terminal *C*-node

The unique outer boundary path corresponding to that in Lemma 2.8 can be extracted from the unique tree path *P* from *u* to *u'* (the proof is analogous). It consists of the following nodes (one should use the original path between two consecutive essential nodes):

1. all *P*-nodes in path *P*;
2. all children of an intermediate *C*-node w that give rise to *i**-subtrees of *w*;
3. all children (which are consecutive) of a terminal *C*-node *w* that give rise to *i**-subtrees of *w* plus the child *v* which is next to the consecutive subsequence s.t. T_v is an *i*-tree. Note also that *v* is the end vertex of the unique path.

Note that children of a *C*-node w that give rise to *i**-subtrees do not have to be traversed, it suffices to keep track of the end vertices of *w* that are in path *P*.

The outer boundary path related to Corollary 2.9 can be extracted similarly. When there is only one terminal node which happens to be a *C*-node *w*, then we could have the situation that the unique boundary path consists of all essential children in the *RBC* that give rise to *i*-subtrees plus the parent of *w* (as shown in Figure 3.7).

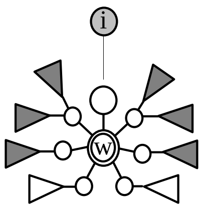


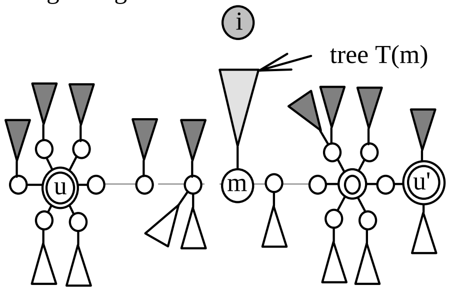
Figure 3.7. The unique path around a *C*-node

To complete the planarity test for this iteration, we only have to show that each 2-connected component can have a planar embedding.

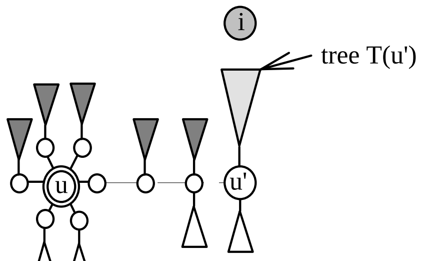
4. The Internal Embedding of a 2-Connected Component Formed in Section 2

In Lemma 2.6, Corollary 2.7, Lemma 3.1 and Lemma 3.2 we made the assumption that graph *G* is planar in deriving at those conclusions. We shall show that if these conclusions hold at each iteration, then *G* must be planar by showing that these conditions imply a feasible internal embedding for each 2-connected component. To ensure that the embedding is correct we need to construct a planar embedding in which node *i* and the terminal nodes are on the boundary cycle.

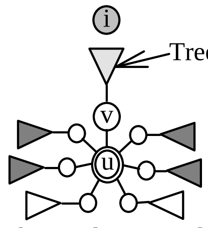
Our embedding algorithm will be based on the unique tree path (call it *P*) in Lemma 2.6 and Corollary 2.9. In both cases we shall make the following modifications to simplify our discussion: Delete the tree edge (*i*,*r*). Then the tree path from *r* to the parent of m in case (i) (respectively, *u'* in case (ii) and *v* in case (iii)) together with their children subtrees form a tree *T*(*m*) (respectively, *T*(*u'*) and *T*(*v*)). This tree satisfies the same property as any *i*-tree does: none of the nodes in the tree has a neighbor greater than *i*. Namely, they can only be adjacent to *i*. We illustrate the subtrees for internal imbedding in Figure 4.1.



(i) the two terminal nodes case



(ii) the single terminal node case
(u is not a C-node or $u \neq u'$)



(iii) the single terminal node case
(u is a C-node and $u = u'$)

Figure 4.1. The subtree for internal imbedding

We shall now imbed the following *PC*-tree $T[i]$ in the plane: $T[i]$ consists of the path P, the i-subtrees of nodes in P and the special tree $T(m)$ (respectively, $T(u')$, $T(v)$). In addition, the children of each C-node in path P will observe its original order and each P-node j in path P will be changed to a C-node with its neighbors oriented as follows. Let the two neighbors of j in P be denoted by j_1, j_2 where j_1 is closer to node u. Order the other neighbors of j in $T[i]$ arbitrarily into a sequence Q_j. Then arrange all neighbors of j into a clockwise cyclic order: j_1, Q_j, j_2.

It can be easily proved by induction that every *PC*-tree can be embedded in the plane. Now, embed the *PC*-tree $T[i]$ in the plane. Below, we describe a way to order the neighbors of i in the tree $T[i]$ which admits a planar embedding of edges incident to i.

For any C-node j with a fixed orientation, define the *clockwise next neighbor* (*CNN*) of a neighbor v for j to be the neighbor of j that follows v in the clockwise direction.

In cases (i) and (ii) of Figure 4.1, let u_1 be the neighbor of u in path P. Let w be the *CNN* of u_1 for u. In case (iii), let w, w' be the essential nodes in the *RBC* of the C-node u such that T_w is an i-tree, $T_{w'}$ is an i*-tree and w follows w' in the clockwise direction according to the current imbedding of $T[i]$ (recall that there are two ways to imbed a C-node). Start a DFS from u by first visiting w and breaking ties in favor of the *CNN* of the current node. The following can be easily proved by induction.

Theorem 4.1. *If we order the neighbors of i in tree $T[i]$ according to the above DFS ordering, then the final embedding is planar and, node i and the terminal nodes will be on the outer boundary of the 2-connected component .*

The containment relationships of the 2-connected components created during the algorithm can be recorded by a tree. Since the internal imbedding of each such component can be fixed (the final orientation of all components within this component can also be fixed), the final embedding can be backtracked by following down the containment tree and embed the children components recursively.

5. The Implementation and the Complexity Analysis

The algorithm consists of the initial labeling, the tree traversal for identifying the i-trees, and the embedding inside a 2-connected component. Since a planar graph can have most $3n$ - 6 edges, the number of edges of G can be assumed to be $O(n)$. It is clear that the initial labeling takes linear time.

Next we discuss the traversal algorithm for identifying the i-trees. Search the neighbors of i in ascending order. Traverse from the smallest unmarked neighbor up the tree until a marked node is encountered. Mark every node traversed. Repeat the above process until all neighbors of i in T_r are marked. This will give us the unique tree path P (for example, the first node encountered in the traversal with a label greater than i is a terminal node).

By Lemma 2.6 and Corollary 2.7, for a planar graph the nodes that are left unmarked at the end of the algorithm are exactly those in i*-trees. Each back edge will be used once in the traversal. The traversal on the marked nodes can be charged on the traversed edges. Each tree edge traversed will be placed either inside or on the outer boundary of some 2-connected subgraph. When a tree edge (u,v) (from u to its parent v) is place on the outer boundary cycle, we revise $b(v)$ by deleting u from its child list and update the maximum children label in constant time (since the children labels are already sorted). We could also assume that a tree edge can be placed on the outer boundary at most once (because after that, it is replaced by the links of the *RBC*). Since we assume the given graph is 2-connected, each edge will eventually be placed either inside or on the outer boundary of some 2-connected subgraph. The number of links created is never greater than the total number of edges.

Now, consider the traversal on the links of an *RBC*. In determining which side of an intermediate *C*-node w contains i-subtrees (let v, v' be two neighbors of w in P) we only have to check the two neighbors of v (or v') in the cyclic list to see which one has the label i. Thus, we could assume that each link of an *RBC* which stays on the outer boundary at the end of an iteration is never traversed again; and each link traversed will be placed inside a 2-connected component. Thus, the total number of times an edge or a link is traversed is constant and the running time of our tree traversal algorithm is linear.

Finally, the time for the embedding algorithm is proportional to the time for the tree traversal. Hence, the total running time of the algorithm is linear.

6. Acknowledgment

The authors would like to thank Professor R. H. Möhring for his valuable discussions during the rewriting of this paper.

References

1. K. S. Booth and G. S. Lueker [1976], *Testing the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms,* **J. Comput. Syst. Sci.** 13, 335-379.
2. J. Cai, X. Han and R. E. Tarjan [1993], *An O(mlogn)-time algorithm for the maximal planar subgraph problem*, **SIAM J. Comput.** 22, 1142-1162.
3. N. Chiba, T. Nishizeki and S. Abe and T. Ozawa [1985], *A linear algorithm for embedding planar graphs using PQ-trees*, **J. Comput. Syst. Sci.** 30, 54-76.
4. J. E. Hopcroft and R. E. Tarjan [1974], *Efficient planarity testing,* **J. Assoc. Comput. Mach.** 21, 549-568.
5. A. Lempel, S. Even and I. Cederbaum [1967], *An algorithm for planarity testing of graphs*, **Theory of Graphs**, ed., P. Rosenstiehl, Gordon and Breach, New York, 215-232.
6. K. Mehlhorn [1984], *Graph algorithms and NP-completeness*, **Data Structure and Algorithms** 2, 93-122.
7. K. Mehlhorn and P. Mutzel [1994], *On the embedding phase of the Hopcroft and Tarjan planarity testing algorithm,* to appear in **Algorithmica**, Technical Report MPI-I-94-117, Max-Planck-Institute f. Informatik, Saarbrucken.
8. J. Small [1993], *A unified approach of testing, embedding and drawing planar graphs*, Proc. ALCOM International Workshop on Graph Drawing, Sevre, France.
9. H. Stamm-Wilbrandt [1993], *A simple linear-time algorithm for embedding maximal planar graphs,* Proc, ALCOM International Workshop on Graph Drawing, Sere, France.
10. S. G. Williamson [1984], *Depth-first search and Kuratowski subgraphs,* **J. ACM** 31, 681-693.