Correcting and Implementing the PC-tree Planarity Algorithm

John M. Boyer¹, Cristina G. Fernandes^{2*}, Alexandre Noma^{2**}, and José Coelho de Pina Jr.^{2*}

¹ PureEdge Solutions Inc. Victoria, BC Canada jboyer@acm.org; jboyer@PureEdge.com ² University of São Paulo, Brazil {cris, noma, coelho}@ime.usp.br

Abstract. A graph is *planar* if it can be drawn on the plane with vertices at unique locations and no edge intersections except at the vertex endpoints. Recent research efforts have produced new algorithms for solving planarity-related problems. Shih and Hsu proposed a linear-time algorithm based on a data structure they named PC-tree, which is similar to but much simpler than a PQ-tree. However, their presentation does not explain in detail how to implement the routines that manipulate a PC-tree, and there are some nontrivial correctness and run-time issues that were not addressed in their paper. So it is far from trivial to derive a proper linear-time implementation from their description. This paper presents additions to the theoretical framework of the PC-tree algorithm that are necessary to achieve correctness and linear running time. A linear-time implementation that addresses the issues raised in this paper was developed in the LEDA platform and is available.

1 Introduction

The first linear-time planarity testing algorithm is due to Hopcroft and Tarjan [10]. The method first embeds a cycle of the graph, then it breaks the remainder of the graph into a sequence of paths that can be added either to the inside or outside of the starting cycle. Some corrections appear in [8], and significant additional details are presented by Williamson [18,20] as well as the text by Reingold, Nievergelt and Deo [15].

The second method of planarity testing proven to achieve linear time began with a quadratic algorithm due to Lempel, Even, and Cederbaum [13] (the LEC algorithm). The algorithm begins by creating an *st*numbering for a biconnected input graph. One property of an *st*-numbering is that there is a path of higher numbered vertices leading from every vertex to the vertex t, which has the highest number. Thus, if the input graph is planar, there must exist an embedding \tilde{G}_k of the first k vertices such that the remaining vertices (k + 1 to t) can be embedded in a single face of \tilde{G}_k . This planarity testing algorithm was optimized to linear time by a pair of contributions. Even and Tarjan [9] optimized *st*-numbering to linear time, while Booth and Lueker [1] developed the PQ-tree data structure, which allows the planarity test to efficiently maintain information about the portions of the graph that can be permuted or flipped before and after embedding each vertex. Chiba, Nishizeki, Abe, and Ozawa [6] augmented the PQ-tree operations so that a planar embedding is computed as the operations are performed, all in linear time.

These algorithms are widely regarded as being quite complex [6,11,16]. Recent research efforts have resulted in two simpler linear-time algorithms, proposed independently, one by Boyer and Myrvold [3, 4] and the other by Shih and Hsu [16]. Both algorithms present a number of similar and very interesting ideas. One of the common ideas consists of processing the vertices in a post-order traversal of the depth first search (DFS) tree of the graph, or simply the reversal of the DFS number order (instead of an *st*-numbering). This has the property that there is a path of unprocessed vertices from every vertex to the root of the DFS-tree. While processing vertex v, the edges from v to the already processed vertices are embedded (if possible).

In our opinion, the Boyer-Myrvold method is the simplest among all linear-time algorithms currently known for the planarity problem. Their data structure maintains a collection of planar biconnected components that are formed as edges are added. The cut vertices separating the biconnected components are represented by 'virtual' vertices. For each vertex v in reverse of the depth first search order, a preliminary bottom-up method is performed to identify the 'active' portion of the DFS subtree rooted at v based on which of its subtrees contain a proper descendant that, in the input graph, is adjacent to v by a back edge.

^{*} Research partially supported by PRONEX/CNPQ 664107/1997-4 (Brazil).

^{**} Supported by FAPESP, a Brazilian funding agency.

Then, a method called 'Walkdown' traverses the active DFS subtree in a top-down fashion, embedding back edges from v to its descendants and merging biconnected components as necessary while preserving planarity. The Walkdown traversal method obeys a few simple rules that guarantee that it will be able to embed all edges from v to its descendants except when a $K_{3,3}$ or K_5 minor can be identified.

The method of Shih and Hsu [16] also processes the vertices of the input graph from descendants to ancestors, and it also adds the back edges from i to its descendants unless a nonplanarity conditions is detected. To effect this strategy, Shih and Hsu created a data structure called a PC-tree, which is a simplified form of the Booth-Lueker PQ-tree. For each vertex i, the algorithm first searches for a number of defined nonplanarity patterns in the PC-tree, and if none are found, then a planarity reduction is applied to embed the edges from i to its descendants. However, Shih and Hsu's formulation lacks a description of how exactly to implement the routines that manipulate the PC-tree to solve the planarity problem. This description is essential for one to derive a linear-time implementation of their algorithm, and there is a series of nontrivial details involved. Moreover, there are some flaws in the proof that the nonplanarity patterns and the planarity reduction patterns together form an unavoidable set. This problem was first reported in Boyer's dissertation [2] (see also Boyer and Myrvold [4]), and this paper provides a solution.

This paper presents a corrected version of Shih and Hsu's algorithm (SH algorithm) that contains several new nonplanarity patterns. Section 2 first presents an overview of the PC-tree data structure and algorithm as presented in [16]. Then, Section 3 presents corrections to the SH algorithm, and Section 4 presents a proof that the corrected SH algorithm does indeed distinguish between planar and nonplanar graphs. As for performance, Section 5 describes two issues that arise when one tries to create a linear-time implementation of Shih and Hsu's ideas. The solutions for these two issues were inspired by Boyer and Myrvold [3]. A linear-time implementation that accounts for the correctness and speed issues described in this paper can be found at http://www.ime.usp.br/~coelho/sh. Section 6 presents an empirical comparison of this implementation to linear-time implementations of other well-known planarity algorithms.

2 Overview of Planarity by PC-trees

The Shih-Hsu algorithm begins by embedding the depth first search tree (a trivial task). The main processing model is therefore concerned with embedding the back edges for each vertex. The vertices are processed in a post-order traversal of the depth first search tree. For a vertex i, the back edges from i to its descendants are added. The back edges from i to its ancestors are embedded when those ancestors are processed.

If a graph G is planar, then it is always possible to produce a planar embedding G_i of the subgraph induced by the subtree rooted by i such that all descendants of i with back edge connections to ancestors of i are on the boundary of a single face of \tilde{G}_i . The rationale is the same as that given above for the LEC algorithm. Hence, when the SH algorithm is processing a planar graph, it creates successively larger partial embeddings of the form $\tilde{G}_0, \tilde{G}_1, \tilde{G}_2, \ldots, \tilde{G}_i, \ldots, \tilde{G}_n$, where the last result is an embedding of G. Naturally, the SH algorithm must also account for nonplanar graphs. Nonetheless, the processing model for vertex iremains quite simple: search the partial embedding \tilde{G}_{i-1} for nonplanarity conditions established by several lemmata, and if none are found, then apply a planarity reduction to produce \tilde{G}_i .

The SH algorithm represents the partial embedding with a data structure called a PC-tree. The starting PC-tree represents \tilde{G}_0 , which is the depth first search tree only, with no back edges. Each node of the tree is a P-node that represents a single vertex of the input graph G. The PC-tree remains a tree at all times even though it conceptually represents a subgraph that contains cycles as back edges from G are embedded. As back edges are added, they biconnect portions of the embedding that were previously separable. The separable components are represented by multiple nodes of the PC-tree, and these are consolidated into a single C-node representing the new biconnected component.

In general, the P-nodes of a PC-tree represent cut vertices in the partial embedding, and the C-nodes represent biconnected components. Before the back edges from a vertex *i* to its descendants can be embedded, the partial embedding \tilde{G}_{i-1} must be rearranged so that vertices with back edge connections to proper ancestors of *i* are in a single face. This rearrangement must follow certain rules. Specifically, the children of a P-node can be arbitrarily permuted, and the children of a C-node can only be flipped (reversed). The nonplanarity conditions detect when the required rearrangement is not possible. If the rearrangement is possible, then the planarity reductions perform the rearrangement, and they consolidate portions of the PC-tree into single C-nodes as necessary to effect the embedding of the new back edges and produce \tilde{G}_i .

Each C-node in a PC-tree has only P-node neighbors that represent vertices along the external face bounding cycle of the biconnected component represented by the C-node. For this reason, the P-node neighbors of a C-node are called its *representative bounding cycle* (RBC). Given a C-node c, the neighbor that is closer to i than c is the parent of c, and the other neighbors of c are its children. However, the children of a C-node cannot indicate the C-node as the parent (see Section 5.2), so in order to traverse from a child w to the parent p of c (or vice versa), one of two paths around the RBC is taken.

In general, T_v denotes the PC-subtree rooted by v, which represents the partial embedding of a subgraph of G induced by the vertices of the DFS subtree rooted by vertex v. For each DFS child r of i, the algorithm considers separately the embedding of back edges between i and vertices in T_r . This is permissible since, given a subgraph H containing the DFS tree of G plus all back edges between vertices in T_i , vertex i still separates any two of its DFS children r_1 and r_2 in H. Therefore, a Kuratowski subgraph cannot span the subgraphs induced by T_{r_1} and T_{r_2} because there are not enough paths connecting them.

Within T_r , a subtree T_s is an i^* -subtree if it has unembedded back edge connections only to proper DFS ancestors of *i*. Similarly, an *i*-subtree is a subtree T_s of T_r that has unembedded back edges only to *i*. The absence of the nonplanarity patterns is supposed to guarantee that one of the planarity reductions is applicable. The planarity reductions have the property that the children of P-nodes can be permuted and the children of C-nodes can be flipped (reversed) such that all *i*-subtrees are near *i* and all *i**-subtrees can be avoided while visiting the *i*-subtrees to embed the back edges to *i*. See Figure 1.



Fig. 1. Permute P-nodes and flip C-nodes to visit *i*-subtrees and to avoid i^* subtrees. a) Darkened triangles are *i*-subtrees, whitened triangles are i^* -subtrees, double circles are C-nodes and single circles are P-nodes. b) The children of u are permuted, node c is flipped on the path axis (u, \ldots, m) , and u' is not changed since it has the desired configuration.

A terminal node is a node t in the PC-tree with the following properties: 1) t has a child *i*-subtree or is adjacent to *i* by a back edge; 2) t has a child *i**-subtree or is adjacent to a proper ancestor of *i* by a back edge; 3) t has no proper descendants in the PC-tree with the same properties. Terminal nodes are so named because they are the endpoints of *critical paths* to *r* that must be searched for nonplanarity conditions. As an example, Figure 2 illustrates a nonplanarity condition that can arise if there are three or more terminal nodes. Then, Lemma 1 (Lemma 2.5 in [16]) describes a necessary condition for planarity, the absence of which yields the $K_{3,3}$ minor appearing in Figure 3.

Lemma 1 (Shih and Hsu). 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'. Let P' be the unique path from m to r. Let $S = \{v || v \text{ is a child of a node in } P, \text{ but } v \text{ is not in } P\}$. Let $S' = \{v || v \text{ is a child of a node in } P' - \{m\}, \text{ but } v \text{ is not in } P'\}$ (note that when m=r, S' is empty). Then, for each node v in S, T_v is either an *i*-subtree or an *i**-subtree, and for each node v in S', T_v is an *i*-subtree.

Proof. Since nonessential nodes are removed, if v in S or S' is not the root of an *i*-subtree or *i**-subtree, then T_v must contain another terminal node, contradicting the assumption of two terminal nodes. That $v \in S'$ cannot be the root of an *i**-subtree is proven by Figure 3, which depicts the resulting $K_{3,3}$ minor. \Box



Fig. 2. (a) A PC-tree with three terminal nodes. (b) The corresponding $K_{3,3}$ Minor. Note that there are many possible variations in connections of the critical paths and the i^* -subtree connections to proper ancestors of i, but edge contraction is used to eliminate unnecessary complexities.



Fig. 3. (a) An i^* -subtree attached to a proper ancestor v' of m, the closest common ancestor of two terminal nodes u and u'. (b) The resulting $K_{3,3}$ from [16]

Remark: The proof of Lemma 1 (Lemma 2.5 in [16]) is specific to PC-trees that contain only P-nodes. Section 3 discusses difficulties with its extension to general PC-trees that contain C-nodes.

3 Corrections for the PC-tree Planarity Algorithm

The PC-tree method in [16] requires some fixes to yield a correct planarity test. Aside from the three terminal node case, Shih and Hsu present four necessary conditions for maintaining planarity: "In Lemma 2.5, Corollary 2.6, [and] Lemmas 3.1 and 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 by showing that these conditions imply a feasible internal embedding for each 2-connected component." [16, p. 188]. The authors then proceed to demonstrate how to perform planarity reductions for the one and two terminal node cases, but the proof does not show that the presence of the four necessary planarity conditions yields only PC-trees that are reducible by the methods shown.

3.1 Patterns of child *i*-subtrees and i^* -subtrees around a terminal C-node

Perhaps the most critical problem for PC-tree planarity correctness pertains to Lemma 3.2 in [16]. The lemma seeks to characterize the allowable pattern of child *i*-subtrees and *i**-subtrees around a terminal C-node. Put simply, the lemma states that for the root j of any child *i*-subtree of a terminal C-node, one of the two RBC paths from j to the parent of the C-node must contain only *i*-subtrees.

While the lemma statement is certainly necessary to maintaining planarity, it is only sufficient in the one terminal node case when the terminal node has no proper ancestor with a child i^* -subtree. In the two terminal node case and the one terminal node case where the terminal node has a proper ancestor with a child i^* -subtree, it is possible to be compliant with the statement of the lemma yet still have a nonplanarity condition. Lemma 2 characterizes the additional restriction required on terminal C-nodes. Figure 4 depicts example PC-trees for the additional restriction, along with the resulting K_{3,3} minor.



Fig. 4. (a) A $K_{3,3}$ nonplanarity minor from [3], (b) A corresponding PC-tree with one terminal C-node having the forbidden child *i*-*i*^{*} subtree pattern, (c) Another example with two terminal C-nodes that have the forbidden child *i*-*i*^{*} subtree pattern. Only one of the terminal nodes must be a C-node with the forbidden subtree pattern.

Lemma 2. If a terminal C-node c has a proper ancestor a with either a direct back edge to a proper ancestor of i or a child v not an ancestor of c such that T_v contains an i^{*}-subtree, then c must have a child w for which an RBC path from w to the parent p of c contains all child i-subtrees of c.

Proof. The children of the terminal C-node in Figure 4(b) depict the minimal configuration of forbidden subtrees to which all forbidden subtree patterns can be reduced. The result is the $K_{3,3}$ minor in Figure 4(a). Figure 4(c) shows that with two terminal nodes, a terminal C-node also must not have the forbidden subtree configuration because the subtree containing the other terminal node attached to the least common ancestor m is analogous to a child *i**-subtree, so again the $K_{3,3}$ minor in Figure 4(a) results.

3.2 Patterns of child *i*-subtrees and i^* -subtrees around an intermediate C-node

Lemma 3.1 of [16] places a necessary condition on the intermediate C-nodes of the path P between two terminal nodes. Given an intermediate C-node c with neighbors v and v' in P, one of the two RBC paths strictly between v and v' must contain only *i*-subtrees and the opposing RBC path strictly between v and v' must contain only child *i**-subtrees.

There are three problems with this lemma in [16]. First of all, as a proof by contradiction, the proof must account for the negation of the condition in the theorem. The proof in [16] presents the case of having both a child *i*-subtree and *i**-subtree along a single RBC path. However, it is possible to avoid this case yet still have a nonplanarity condition according to the lemma statement if both RBC paths contain only a child *i**-subtree. Secondly, the stated condition is not quite strong enough if the intermediate C-node is m, the closest common ancestor of the terminal nodes. Lemma 3 provides the required modifications to the statement and proof of Lemma 3.1 in [16]. The third problem is simply that Lemma 3.1 of [16] applies only to the two terminal node case, but Corollary 1 demonstrates the need to extend the necessary condition of the lemma to the analogous scenario in the one terminal node case.

Lemma 3. Given the PC-tree path P between two terminal nodes u and u' in T_r , consider an intermediate C-node c in $P - \{u, u'\}$ with neighbors v and v' in P. Let m denote the closest common ancestor of u and u' in the PC-tree. Then, the children of c along one RBC path of c strictly between v and v' must be only child *i*-subtrees, and the opposing RBC path strictly between v and v' must contain only child *i**-subtrees. Further, if c = m, then the RBC path containing the child *i*-subtrees must also contain the parent p of c.



Fig. 5. (a) A PC-tree in which m is a C-node with a child *i*-subtree below path P between terminal nodes x and y. (b) A PC-tree with an intermediate C-node that has child *i*^{*}-subtrees along both RBC paths from parent p to the next node w in path P. (c) The corresponding K_{3,3} minor from [3]. Note: This minor also appears in the new three terminal node case of Figure 7 as well as the forbidden child *i*-*i*^{*} subtree pattern of Lemma 3.2 in [16].

Proof. When $c \neq m$, the proof of Lemma 3.1 in [16] demonstrates the nonplanarity condition that results if one RBC path contains both a child *i*-subtree and *i**-subtree. The nonplanarity condition also covers the case in which c = m and both a child *i**-subtree and *i*-subtree appear in the RBC path that excludes the parent p of c. The remaining points below were omitted from the proof of Lemma 3.1 in [16].

When c = m, the RBC path strictly between v and v' that excludes the parent p can still generate a nonplanarity condition even if it contains no child i^* -subtree as required by the partial proof of Lemma 3.1. If that RBC path contains a child *i*-subtree, then the PC-tree has the form depicted in Figure 5(a), which results in the K_{3,3} depicted in Figure 5(c).

When c = m, the RBC path strictly between v and v' that includes the parent p of c cannot contain a child i^* -subtree. Given the root j of such a child j i^* -subtree, one of the nonplanarity minors depicted for Lemma 5 can be obtained by edge contracting the RBC path to merge j with the parent p of c.

When $c \neq m$, then both RBC paths around c strictly between v and v' cannot contain a child *i**-subtree. If both RBC paths contain child *i**-subtrees as shown in Figure 5(b), then a K_{3,3} can be found according to the nonplanarity minor in Figure 5(c).

Remark: The nonplanarity condition of Figure 5(a) is similar to the one in Lemma 3.2 of [16], which requires the C-node to be a terminal node and x and y to be child i^* -subtrees that obstruct both RBC paths from w to the parent of the C-node.

Corollary 1. Given one terminal node u, let P denote the path from u to the farthest ancestor u' with a child i^* -subtree. Let c be an intermediate C-node in path $P - \{u\}$. For $c \neq u'$, let v and v' denote the neighbors of c in P. For c = u', let v denote the neighbor of c in P and let v' denote the closest child i^* -subtree along either RBC path from the parent p of c. The following conditions must hold:

- The children of c in one RBC path strictly between v and v' must contain only child i-subtrees.
- The opposing RBC path strictly between v and v' must contain only child i^* -subtrees.
- If c = u', then the RBC path containing the child i-subtrees must also contain p.

3.3 Direct back edges as degenerate i-subtrees and i^* -subtrees

There is an omission from the presentation of several results in [16], including Theorem 2.4, Lemma 2.5 and Corollary 2.6 of [16]. Lemma 4 demonstrates that the nonplanarity condition for Lemma 2.5 in [16] (Lemma 1 above) can still occur despite the absence of the condition stated by the lemma. Since a number of other results in [16] have the same problem, Corollary 2 makes a statement that fixes the underlying problem.

Lemma 4. Given the same assumptions as Lemma 1, nonplanarity can result if S' is empty or devoid of vertices that root child i^* -subtrees.

Proof. A node in $P' - \{m\}$ can have a direct back edge to a proper ancestor of *i*.

Corollary 2. A back edge (v, i) can be considered equivalent to a child *i*-subtree of v, and a back edge (v, t) where t is a proper ancestor of i can be considered equivalent to a child *i**-subtree of v.

Proof. Solely for the purpose of simplifying proof statements, such direct back edges can be considered to be subdivided by an implicit degree two vertex w, which would be an implicit child of v.

3.4 Additional cases of surrounding an i^* -subtree

Consider the extension of Lemma 2.5 in [16] (presented in Lemma 1 above) to the case of a PC-tree that contains C-nodes. Specifically, suppose that the closest common ancestor m of the two terminal nodes is in fact a C-node whose parent has the only child *i**-subtree along the path P'. Figure 6 depicts an example PC-tree and the corresponding K₅ minor pattern from [3]. This case is of critical importance because some graphs that it represents do not even contain a K_{3,3}, which demonstrates that the proof of Lemma 2.5 does not "go through for the case of general trees without any changes provided that the paths through a C-node are interpreted correctly" [16, p. 185]. Lemma 5 properly extends Lemma 2.5 of [16], including a proof that no other nonplanarity patterns result from its necessary condition.



Fig. 6. (a) A PC-tree in which the closest common ancestor of terminal nodes u and u' is a C-node with a proper ancestor that has a child i^* -subtree. (b) The corresponding K_5 minor from [3].

Lemma 5. Suppose there are two terminal nodes u and u' in T_r , and let m be their closest common ancestor. Let P' be the unique path from m to r. If m has a proper ancestor in T_r with a child i^{*}-subtree, then the input graph is not planar.

Proof. If m is a C-node, then the PC-tree has the form shown in Figure 6(a) and the input graph can be edge contracted to the K₅ in Figure 6(b) as follows. First, since r and its ancestors are P-nodes, edge contract the proper ancestor of i into one vertex t and do nothing to i and r. For each C-node c in P' - m, edge contract its RBC so that only the parent and child of c in P' remain. Then, edge contract P' - m into r. Similarly, edge contract the RBCs of C-nodes in P - m into a single edge per C-node. Then, edge contract the proper descendants of m leading to u into either u if u is a P-node or a child of u if u is a C-node. Likewise, edge contract the proper descendants of m leading to u' into either u' if it is a P-node or a child of u' if u' is a C-node.

On the other hand, if m is a P-node, then all C-nodes in T_r can be edge contracted as described above. Then, the $K_{3,3}$ given for Lemma 2.5 in [16] is applicable (see Figure 3).

The proof of Lemma 5 is also important because it demonstrates the actual method by which K_5 homeomorphs are found by the PC-tree algorithm, which also contradicts [16]: "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 Fig. 6." The K_5 in Figure 6 of [16] is equivalent to Figure 6(b). It does not result in three terminal nodes, but is instead discovered by the condition in Lemma 5. Moreover, the case of three terminal node neighbors of a C-node depicted in Figure 7 should be part of an extension of Theorem 2.4 in [16] to PC-trees containing C-nodes, but again the proof does not extend to general PC-trees because the $K_{3,3}$ identified in the proof cannot always be obtained. Lemma 6 provides the proper extension of the three terminal node case to general PC-trees that contain C-nodes.

Lemma 6. If T_r contains three terminal nodes, then the input graph is not planar.

Proof. The proof of Theorem 2.4 in [16] provides the proper $K_{3,3}$ in PC-trees containing only P-nodes (see Figure 2). For general PC-trees containing C-nodes, only proper descendants of r in T_r need to be considered since r and its ancestors are P-nodes.

For each of the three terminal nodes, denoted i_1 , i_2 and i_3 , let P_1 , P_2 and P_3 denote the critical paths from each terminal node to r. Without loss of generality, label the terminal nodes so that the join point j_1 of P_1 and P_2 is equal or descendant to the join point j_2 of the first two paths with P_3 . The endpoints of these critical paths are r and each of the terminal nodes. The endpoint r is a P-node. For each terminal C-node, edge contract the children of the RBC into a single vertex so that only the parent and one child of the C-node remain, then use the child of the C-node as an image vertex of a $K_{3,3}$, either from Figure 2 or Figure 7 depending on the conditions described below.

For each internal C-node of each critical path except j_1 and j_2 (if either is indeed a C-node), edge contract the RBC to a single edge containing the parent and child in the critical path. Since the endpoints of the critical paths have already been discussed, this leaves only j_1 and j_2 to consider.

If both j_1 and j_2 are P-nodes, then clearly j_1 can be used as the image vertex w in Figure 2, and the $K_{3,3}$ identified in [16] for the three terminal node case can still be obtained. Thus, suppose one or both of j_1 and j_2 are C-nodes.

Suppose $j_1 \neq j_2$. If j_1 is a P-node, then j_2 must be a C-node. Let c_3 denote the child of j_2 that leads to i_3 , and let $c_{1,2}$ denote the child of j_2 that leads to j_1 . In this case j_1 can again be used as the image vertex w. The path from w to r leads up to $c_{1,2}$. Then it follows the RBC path from $c_{1,2}$ through c_3 to the parent of j_2 then up to r. On the other hand, if j_1 is a C-node with parent p and children c_1 and c_2 leading to i_1 and i_2 , then the RBC paths from p to each of c_1 and c_2 can be contracted to a single edge. If j_2 is a P-node, then p is the desired vertex w and we are done. If j_2 is a C-node, then j_2 must be a proper ancestor of p. Again, we let p be the desired vertex w since the path from w to r can be obtained by going around the RBC of j_2 as described above.

Finally, suppose $j_1 = j_2$ is a C-node. Let c_1 , c_2 and c_3 denote the children of the C-node in RBC order that lead to each of the respective terminal nodes, and let p denote the parent of the C-node. Unless the biconnected component represented by the C-node happens to have an internally embedded path connecting c_2 and p, the desired vertex w in the $K_{3,3}$ of Figure 2 cannot be obtained. Figure 7(a) depicts the PC-tree for this case, which reduces to the $K_{3,3}$ in Figure 7(b).



Fig. 7. (a) A PC-tree with three terminal node proper descendants of a C-node. Note that the paths from the terminal nodes to the RBC vertices of the C-node have been contracted. (b) The corresponding $K_{3,3}$ minor from [3].

3.5 The case of zero terminal nodes

Lemma 7 presents an additional planarity reduction for zero terminal nodes, which occurs in the final step of embedding every graph. This case is easy to resolve, but it is worth mentioning since it is essentially a missing planarity reduction.

Lemma 7. If, during the embedding of a biconnected graph G, there is a step i for which zero terminal nodes are identified, then the PC-tree can be reduced to a single C-node plus P-node neighbors for the RBC of the C-node.

Proof. If there are no terminal nodes, then there are no i^* -subtrees within T_r . Having no i^* -subtrees prior to the last step contradicts the biconnectedness of G. Since G is biconnected, by definition its final embedding can be represented as described.

4 Proof of Correctness for Modified PC-tree Algorithm

This section presents a proof of correctness for the PC-tree algorithm as modified by the lemmas and corollaries of Section 3. First, the planarity reduction patterns are clearly characterized with property statements below. Then, violations of the properties are mapped to the lemmas and corollaries so that it is clear that the planarity reduction patterns are the only ones that do not result in a nonplanarity condition. Since it is clear how to maintain planarity for each of the planarity reductions, the correctness of the algorithm follows.

For the two terminal node case, let u and u' denote the two terminal nodes. Let m denote the closest common ancestor of u and u', and let P denote the PC-tree path $(u, \ldots, m, \ldots, u')$. Let P' denote the PC-tree path (r, \ldots, m) . For the one terminal node case, let u denote the terminal node and let u' denote the ancestor of u closest to the root of T_r that has a child i^* -subtree. Let m be a second label for u'. Let P denote the path (u, \ldots, u') , and let P' denote the path (r, \ldots, u') . To simplify the statement of properties, consider path P to be arranged horizontally in the plane, and consider P' as extending vertically upward from P. Let L denote an infinite horizontal line that contains P.

Property 1. Nodes in $P' - \{m\}$ have no child i^* -subtrees.

Property 2. The children of nonterminal nodes in P are arranged so that all child *i*-subtrees are above L and all child *i**-subtrees are below L.

Property 3. Except for the case of one terminal C-node and u = u', the children of terminal nodes in P are arranged so that all child *i*-subtrees are above L and all child *i**-subtrees are below L.

Property 4. For the case of one terminal C-node and u = u', let p be the parent of u and let w and w' be the first child *i**-subtrees in each of the two RBC paths extending from p. The children of u on the RBC path strictly between w and w' that contains p must be the roots of all child *i*-subtrees of u.

The proof of correctness of the modified PC-tree planarity algorithm in Theorem 1 will show that violations of the four properties above result in a nonplanarity condition. That the above properties characterize the planarity reductions in [16] and that the planarity reductions embed all back edges from i to descendants of r while maintaining planarity are taken to be straightforward. Moreover, the fact that maintaining planarity through all steps implies the planarity of the graph and that finding a nonplanarity condition in a step implies the nonplanarity of the graph are also taken to be evident.

Theorem 1. The modified PC-tree planarity algorithm applies a planarity reduction to T_r if and only if there are no terminal nodes or if there are at most two terminal nodes and Properties 1, 2, 3, and 4 hold.

Proof. Case no terminal nodes: The planarity reduction described in Lemma 7 is applied.

Case one terminal node: Property 1 holds by definition. Property 2 holds if u = u' because there are no nonterminal nodes in P. Property 2 holds if $u \neq u'$ except for nonplanarity conditions due to Corollary 1. If $u \neq u'$, then Property 3 holds except for nonplanarity conditions due to Lemma 2 and Property 4 holds degenerately (is not applicable). On the other hand, if u = u', then Property 3 holds degenerately, and Property 4 holds except for nonplanarity conditions due to Lemma 3.2 of [16]. Case two terminal nodes: Property 1 holds except for nonplanarity conditions due to Lemma 5. Property 2 holds except for nonplanarity conditions due to Lemma 3. Property 3 holds except for nonplanarity conditions due to Lemma 2 and Property 4 holds degenerately.

Case more than two terminal nodes: If there are at least three terminal nodes in T_r , then the input graph is not planar according to Lemma 6. Hence no planarity reduction is applied.

5 Issues Concerning Linear-Time Performance

Shih and Hsu [16] present the ideas necessary to achieve linear total work for the identification of terminal nodes, *i*-subtrees and *i*^{*}-subtrees in all steps of the PC-tree algorithm. However, there are two impediments to achieving linear time by the methods stated in [16]; both are complexities that arise when planarity reductions are applied to a PC-tree that contains C-nodes.

5.1 Maintaining the RBC when flipping C-nodes

The claim that the "RBC will be stored as a circular doubly linked list" [16, p. 184] cannot be supported. When the representative bounding cycles of C-nodes must be joined together, the direction of traversal of two successive C-nodes may be reversed at the intervening P-node depending on which path contains the child *i*-subtrees in each C-node. Joining the RBCs of two such C-nodes into a circular doubly linked list would require the inversion of links in the RBC nodes of one of the two C-nodes. It is easy to create planar graphs in which $\Theta(n^2)$ link inversions occur in total. To solve this problem, one can represent the RBC with a discordant list (defined in [3]). When merging the RBCs of two C-nodes separated by a P-node, only the neighbors of the P-node in the two RBCs are linked together. If the merge must be done such that one C-node is flipped relative to the other, then the resulting RBC pointers after the merge will be in discord. However, traversal of the RBC is still possible with only a little extra effort. When a traversal arrives at a node v from a predecessor p along a discordant RBC, the successor of v is indicated by one of its two RBC pointers. The pointer to use is the one that does not indicate p. Figure 8 illustrates this concept.



Fig. 8. A discordant list of size 3 or more can be traversed by taking whichever pointer does not lead back to the preceding node (from [3]).

5.2 On the infeasibility of C-nodes as parents

In the conceptual ideal, every C-node and P-node indicates its parent according to the PC-tree definition. However, the child P-nodes in the RBC of a C-node cannot indicate the C-node as their parent. Consider a C-node with the following properties: 1) The RBC of the C-node has a subset S of children that root i^* -subtrees which all have back edges only to the last vertex to be processed, and 2) the RBC of the C-node also has an O(n) sized succession of children that root i^* -subtrees which connect to proper ancestors of i such that successive steps of the algorithm merge the RBC of the original C-node into other ancestor C-nodes. At each merge, the members of S must be reparented to point to the new C-node that becomes their parent after the planarity reduction. This reparenting must be performed on a number of i^* -subtree roots that is a constant fraction of n, and the set of reparenting operations must be performed each time the parent C-node of the members of S must be changed to some other ancestor C-node with which the parent is merged. This results in $\Theta(n^2)$ performance in the worst case. The work could be substantially reduced by using the methods of the union-find data structure (also called a disjoint set data structure in [7]), but it would then have to be shown that the result is not super-linear, which is the case for generalized union-find operations. Either way, [16] presents neither this more sophisticated parenting strategy nor the required proof. Perhaps the simplest strategy to solve this problem is not to adopt a complex parenting strategy and present a complex proof, but rather to let the parent pointer of all children of a C-node simply be nil, indicating they are part of a C-node, and keeping a pointer from each C-node child to its entry in the RBC. To find the parent of any node whose parent pointer is nil, traverse both directions around the RBC in parallel. This will obtain the parent of the C-node by the shorter path, so that the work done will not exceed a constant factor of the length of RBC that will be eliminated during the planarity reduction in the same step. This is analogous to the method used in [3, 4] to traverse the external faces of biconnected components that are merged during the processing of a vertex.

6 Empirical Results and Future Work

This paper has reported and solved a number of additional theoretical complexities that arise in the published version of the Shih-Hsu PC-tree planarity algorithms [16]. A few years earlier, Thomas [17] provided an alternate formulation of the Shih-Hsu planarity algorithm that achieved linear-time performance for triconnected graphs. Thomas points out that significant additional technical complications would arise when accounting for graphs of lower connectivity and when one requires a planar embedding.

Our implementation efforts have been based on extending the formulation in Thomas' notes as a way of better understanding and correcting the problems with the PC-tree formulation in [16]. The resulting LEDA-based implementation contains code manifestations of the PC-tree problem solutions reported in this paper. We have achieved a linear-time implementation, both for producing a planar embedding and for isolating a Kuratowski subgraph in a nonplanar graph. We have performed the same empirical tests used in LEDA to compare the Hopcroft-Tarjan and Booth-Lueker implementations, and all results are consistent with the results for maximal planar graphs (MP) and their nonplanar counterparts created by adding one random edge (MP+e). Figure 9 presents the MP and MP+e empirical comparisons of our current implementation with the Hopcroft-Tarjan (HT) and Booth-Lueker (BL) implementations in LEDA, as well as a non-LEDA implementation of the Boyer-Myrvold (BM) algorithm in [4]. Note that there are no HT results for nonplanar graphs because LEDA does not implement Williamson's Kuratowski subgraph isolator (indeed Williamson [21] knows of no O(n) implementation).



Fig. 9. Empirical results comparing the SH, HT, BL and BM implementations on testing and justifying maximal planar graph (left) and their nonplanar counterparts (right) consisting of an extra random edge. The justification consists of creating an embedding for a planar graph or isolating a Kuratowski subgraph of a nonplanar graph. Results for HT on nonplanar graphs cannot be obtained from LEDA.

Although our implementation is not yet competitive with HT and BL, we believe that it can be made more competitive, in part through further application of some of the methods of the BM algorithm, which currently has the fastest implementation by about 2.5 times on planar graphs and about 8 times on nonplanar graphs. Our implementation efforts to date have been principally concerned with correctness and linear-time performance. The correctness concerns led us to extend Thomas' formulation based on an understanding of the original LEC algorithm (this formulation appears in [14]). We believe that the success of this approach in finding and solving problems with the PC-tree formulation substantiates the further investigation and exposition of the SH algorithm as an LEC-type algorithm. Indeed, future work shall consist of refining this alternate formulation with the ultimate goal of developing a unified LEC-type framework for describing the SH, BL and BM algorithms. As Williamson [19] notes, "it would be desirable to have not one but several basically different [linear time Kuratowski subgraph isolators]" because the condition of linearity "forces the emergence of a certain level of insight into the structure of nonplanar graphs and Kuratowski's theorem." The PC-tree formulation [16], augmented by the corrections in this paper, two variations of the BM algorithm [3, 4], and Karabeg's analysis [12] of the BL algorithm collectively demonstrate four different methods for the discovery of nonplanarity. Along with the correspondence drawn between HT and BL in [5], we believe that a more generalized LEC-type formulation could unify all of these methods and increase our graph theoretic understanding of planarity.

References

- 1. K. S. Booth and G. S. Lueker. Testing for the consecutive ones property, interval graphs, and graph planarity using PQ-tree algorithms. *Journal of Computer and Systems Sciences*, 13:335-379, 1976.
- 2. J. Boyer. Simplified O(n) algorithms for planar graph embedding, Kuratowski subgraph isolation and related problems. Ph.D. Thesis, University of Victoria, 2001.
- 3. J. Boyer and W. Myrvold. Stop minding your P's and Q's: A simplified O(n) planar embedding algorithm. Proceedings of the Tenth Annual ACM-SIAM Symposium on Discrete Algorithms, pages 140–146, 1999.
- 4. J. Boyer and W. Myrvold. Simplified O(n) planarity algorithms. Journal of Algorithms, pages 1-41, Submitted Dec. 3, 2001.
- 5. E. R. Canfield and S. G. Williamson. The two basic linear time planarity algorithms: Are they the same? *Linear* and Multilinear Algebra, 26:243-265, 1990.
- N. Chiba, T. Nishizeki, A. Abe, and T. Ozawa. A linear algorithm for embedding planar graphs using PQ-trees. Journal of Computer and Systems Sciences, 30:54-76, 1985.
- 7. T. H. Cormen, C. E. Leiserson, and R. L. Rivest. Introduction to Algorithms. The MIT Press, Cambridge, Massachusetts, 1990.
- 8. N. Deo. Note on Hopcroft and Tarjan planarity algorithm. Journal of the Association for Computing Machinery, 23:74-75, 1976.
- 9. S. Even and R. E. Tarjan. Computing an st-numbering. Theoretical Computer Science, 2:339-344, 1976.
- 10. J. Hopcroft and R. Tarjan. Efficient planarity testing. Journal of the Association for Computing Machinery, 21(4):549-568, 1974.
- M. Jünger, S. Leipert, and P. Mutzel. Pitfalls of using PQ-trees in automatic graph drawing. In G. Di Battista, editor, Proc. 5th International Symposium on Graph Drawing '97, volume 1353 of Lecture Notes in Computer Science, pages 193-204. Springer Verlag, Sept. 1997.
- A. Karabeg. Classification and detection of obstructions to planarity. Linear and Multilinear Algebra, 26:15–38, 1990.
- 13. A. Lempel, S. Even, and I. Cederbaum. An algorithm for planarity testing of graphs. In P. Rosenstiehl, editor, *Theory of Graphs*, pages 215–232, New York, 1967. (Proc. Int. Symp. Rome, July 1966), Gordon and Breach.
- A. Noma. Análise experimental de algoritmos de planaridade. Master's thesis, Universidade de São Paulo, May 2003. in Portuguese.
- 15. E. M. Reingold, J. Nievergelt, and N. Deo. Combinatorial Algorithms: Theory and Practice. Prentice-Hall, Inc., Englewood Cliffs, New Jersey, 1977.
- 16. W.-K. Shih and W.-L. Hsu. A new planarity test. Theoretical Computer Science, 223:179-191, 1999.
- 17. R. Thomas. Planarity in linear time. http://www.math.gatech.edu/~thomas/, June 1997.
- 18. S. G. Williamson. Embedding graphs in the plane- algorithmic aspects. Ann. Disc. Math., 6:349-384, 1980.
- S. G. Williamson. Depth-first search and Kuratowski subgraphs. Journal of the Association for Computing Machinery, 31(4):681-693, 1984.
- 20. S. G. Williamson. Combinatorics for Computer Science. Computer Science Press, Rockville, Maryland, 1985.
- 21. S. G. Williamson. Personal Communication during Boyer's Ph.D. Defense. August 24, 2001.