

# Planarity Algorithms via PQ-Trees

(Extended Abstract)

Bernhard Haeupler\*, Robert E. Tarjan\*<sup>†</sup>

In Memory of Shimon Even

March 26, 2008

## Abstract

We give an abstract vertex-addition method for planarity testing that encompasses the algorithms of Lempel, Even, and Cederbaum, Shih and Hsu, and Boyer and Myrvold. The main difference between the former and the latter two is the order of vertex addition; the latter two differ only in implementation details. For the general method we give a direct proof of correctness that avoids the use of Kuratowski's theorem. We give a linear-time implementation that simplifies and unifies the Shih-Hsu and Boyer-Myrvold methods. Our algorithm extends to generate embeddings uniformly at random, to count embeddings, to represent all embeddings, and to produce a Kuratowski subgraph of a non-planar graph. Our algorithm keeps track of all possible embeddings by reinterpreting Booth and Lueker's PQ-tree data structure to represent circular instead of linear orders. This interpretation of PQ-trees gives the PC-trees of Shih and Hsu and leads to a simpler, more-symmetric form of PQ-tree reduction.

## 1 Introduction

The problem of testing a graph for planarity has a long and rich history; we shall review a small but important part of it. Kuratowski [Kur30] famously characterized non-planar graphs, but his characterization does not lead directly to a polynomial-time planarity test. Hopcroft and Tarjan [HT74] gave the first linear-time algorithm, an implementation of the path-addition approach of Auslander and Parter [AP61] made efficient by the systematic use of depth-first search to find the paths and a "stack of pairs of stacks" data structure to keep track of possible embeddings.

Earlier, Lempel, Even, and Cederbaum [LEC67] gave a different planarity test that builds an embedding by adding vertices in a precomputed order, called an st-order. The algorithm keeps track of the set of possible embeddings by using a formula to represent relevant information about the embeddings. Booth and Lueker [BL76], acting on a suggestion of Tarjan, obtained a linear-time implementation of the Lempel-Even-Cederbaum algorithm by combining a linear-time st-ordering method ([ET76]; see [Tar85] for a related but simpler method) with their efficient PQ-tree reduction algorithm. They had independently [Boo75, Lue75] developed the PQ-tree as a data structure to solve several ordering problems on matrices and graphs, including consecutive ones, circular ones, and interval and chordal graph recognition. PQ-trees correspond exactly to the formulas of Lempel, Even, and Cederbaum; PQ-tree reduction is the same as the main part of their formula-manipulation process. But Lempel, Even, and Cederbaum did not have an efficient implementation of reduction. A straightforward implementation takes time linear in the size of the entire formula, giving a quadratic time bound for planarity testing [Tar69]. Booth and Lueker obtained an efficient reduction algorithm by combining several ideas, including a carefully chosen set of pointers to represent PQ-trees, an elegant method for finding the part of the tree that must be changed during a reduction, and a nice amortized analysis.

---

\*Department of Computer Science, Princeton University, Princeton NJ, {haeupler, ret}@cs.princeton.edu

<sup>†</sup>HP Laboratories, Palo Alto CA

More recently, Shih and Hsu [SH93] and later but apparently independently Boyer and Myrvold [BM99] developed closely-related planarity tests and embedding algorithms that add vertices in postorder with respect to a depth-first spanning tree. Shih and Hsu's algorithm uses what they call a PC-tree to represent cut vertices and biconnected components of a partial embedding, whereas Boyer and Myrvold's algorithm, at least in its later form [BCPDB03, BM04], uses a graph representation directly but maintains a tree structure implicitly. Shih and Hsu's first paper [SH93] did not address important implementation issues, and subsequent papers [SH99, Hsu03, HM03, HM04] gave incorrect algorithms; Boyer et al. ([BFNdP04]; see also [BFNdPJ03, Boy05]) finally gave a correct version of the Shih-Hsu algorithm.

## 2 Planarity Testing via Vertex Addition

Let us systematically explore the vertex-addition strategy for planarity-testing. Consider testing the planarity of a connected graph by starting with an empty embedding and adding one vertex at a time. At any step, each edge will be of one of three types: *embedded*, meaning both ends are embedded, *half-embedded*, meaning exactly one end is embedded, or *unembedded*, meaning neither end is embedded. Addition of a vertex converts some edges from half-embedded to embedded, and some other edges from unembedded to half-embedded. The half-embedded edges form the boundary between the embedded vertices and the unembedded vertices.

We would like to avoid backtracking. This requires some way of maintaining all possible embeddings of the half-embedded edges, as constrained by the embedded ones. In general the half-embedded edges can lie in many different faces of the partial embedding; keeping track of the possibilities seems difficult. As a first simplification, we restrict the order of vertex addition so that the subgraph induced by the unembedded vertices is connected. That is, the vertex addition order is a leaf-to-root order for some spanning tree. Then, if the partial embedding can be extended to a complete embedding, the half-embedded edges lie in a common face, which we can take to be the outside face. The partial embedding in general consists of one or more connected components. These components partition the half-embedded edges. The half-embedded edges incident to each component are circularly ordered around the outside of the component. The set of all possible partial embeddings corresponds to a set of circular orders of half-embedded edges for each component.

Now consider the effect of adding a vertex. The half-embedded edges that become embedded as a result of this vertex addition are partitioned among the connected components existing before the vertex addition. These components are combined into a single component by the vertex addition. The effect of the vertex addition on the possible circular orders of the half-embedded edges is two-fold:

**Reduce:** For each connected component existing before the vertex addition, retain only those circular orders in which all the half-embedded edges that become embedded occur consecutively; if for some component there are no such orders, the graph is non-planar.

**Combine:** Form the set of circular orders of the new half-embedded edges as follows. For each connected component before the vertex addition, choose one of the circular orders remaining after reduction and delete from it all half-embedded edges that become embedded, to give a linear order. Catenate these linear orders, one from each connected component, with the new half-embedded edges incident to the newly added vertex (each of which forms a singleton linear order) to form a circular order. Doing this in all possible ways gives the set of circular orders for the component formed by the vertex addition.

We can prove the correctness of this abstract method by using an appropriate combinatorial corollary of the Jordan curve theorem. This is simpler than the alternative (used by Shih and Hsu and Boyer and Myrvold) of using Kuratowski's theorem (if the algorithm halts and declares the graph non-planar, then it contains a Kuratowski subgraph), which requires a tedious case analysis.

To implement the abstract method, we need a data structure to keep track of sets of circular orders in a way that allows efficient reduction and combination. The PQ-tree of Booth and Lueker provides such a data structure, although we need to reinterpret the data structure as representing circular orders instead of linear orders. We discuss this data structure further in the next section.

Using PQ-trees to obtain a linear time bound for planarity testing further restricts the planarity test, because PQ-trees can be combined efficiently only by attaching the root of one to an arbitrary node of the other. This imposes a second restriction on the order of vertex addition. Each connected component has a *special vertex* corresponding to the root of its PQ-tree. This special vertex is incident to a half-embedded edge. A vertex can be added only if it is adjacent to the special vertices of all but one of the existing connected components. (It can be adjacent to all of them.) If it is adjacent to all but one of them, the special vertex to which it is not adjacent becomes the special vertex of the single new component. If it is adjacent to all of them, any of the special vertices of the old components, or the newly added vertex, can be chosen as the special vertex of the new component. In either case the special vertex must have an incident half-embedded edge, unless the new vertex is the last one and the planarity test is finished.

If we restrict the method in this way, two natural extremes give the vertex addition strategies of Lempel et al. and of Shih and Hsu and Boyer and Myrvold. The first is to assume that throughout the process there is only one connected component, and the first vertex added remains the special vertex throughout the algorithm. This corresponds to adding the vertices in an order such that there is an edge between the first and last vertex, and every intermediate vertex is adjacent to an earlier vertex and to a later vertex. If the edge joining the first and last vertex is ignored, this is exactly the definition of an st-order, giving the Lempel et al. strategy. The second is to assume that throughout the process the newly added vertex is adjacent to the special vertices of all the components, and the newly added vertex is always the special vertex of the new component. This corresponds to adding the vertices in a leaf-to-root order of a depth-first spanning tree, giving the Shih and Hsu and Boyer and Myrvold strategy.

### 3 Circular Orders via PQ-trees

The *PQ-trees* of Booth and Lueker represent sets of permutations of a set, as follows. An *ordered tree* is a tree with a root, such that the children of every node are totally ordered. The *leaf order* of an ordered tree is the order in which the leaves are visited by a preorder traversal of the tree that visits the children of each node in the given total order. A PQ-tree is an ordered tree each of whose internal nodes is either a P-node or a Q-node. Two PQ-trees are *equivalent* if they are isomorphic up to arbitrary reordering of the children of P-nodes and reversal of the order of the children of Q-nodes. A PQ-tree represents the set of permutations of its leaves that are the leaf order of some equivalent tree. Given a PQ-tree  $T$  and a subset  $S$  of its leaves, a *reduction* of  $T$  on  $S$  modifies  $T$  to form a tree  $T'$  that represents the subset of the permutations represented by  $T$  such that the elements of  $S$  occur consecutively (but in arbitrary order).  $T$  is the null tree if there are no such permutations. Booth and Lueker gave an on-line algorithm for reduction that takes  $O(n + m)$  time for a sequence of reductions of an  $n$ -node tree on sets whose sizes sum to  $m$ . This algorithm extends to handle the combining of trees needed in the Lempel et al. planarity test, giving the linear-time implementation of this test by Booth and Lueker.

We give a simple way to reinterpret PQ-trees to represent sets of circular orders instead of linear orders: Given a PQ-tree, we add a new root, whose only child is the original root. We treat this degree one root as a special leaf. The *leaf order* of the augmented tree is the circular order of its leaves including the special leaf that begins with the special leaf, lists the other leaves in leaf order, and returns to the special leaf. The augmented tree represents the set of circular leaf orders of its equivalent trees, with equivalence defined as above. There is a one-one correspondence between leaf orders of trees equivalent to the original tree and leaf orders of trees equivalent to the augmented

tree; all the properties of PQ-trees apply to the new interpretation. We also get a more symmetric view of reduction: given a subset  $S$  of the set of leaves, a reduction of the augmented tree modifies it to produce a new tree that represents the subset of circular orders in which all the elements of  $S$  occur consecutively. If  $S$  does not contain the special leaf, this corresponds to reduction of the original tree on  $S$ . If  $S$  does contain the special leaf, this corresponds to reduction of the original tree with respect to the set of leaves that are not in  $S$ , which we call *complement reduction*. The Booth-Lueker reduction algorithm extends to the new interpretation of PQ-trees, and in particular gives an efficient implementation of complement reduction, for which some slight simplification is possible. A reduction can be done purely bottom-up. A global view of reduction given in the Master's thesis of Young avoids the template-based description of Booth and Lueker, which has many cases.

Augmented PQ-trees representing sets of circular orders correspond to the unrooted PC-trees of Shih and Hsu, if we ignore the root. A root seems to be required for efficient implementation of reduction, however. We view the implementation of PC-trees by Hsu and McConnell as just a reinvention of PQ-trees and PQ-tree reduction, with the more symmetric view of reduction given by considering circular orders instead of linear orders and the global view of reduction given previously by Young.

The Shih-Hsu and Boyer-Myrvold planarity-testing methods in effect use complement reduction. Boyer and Myrvold describe their reduction process as operating on a representation of the partially embedded graph rather than on a separate data structure, but the computation is still an implementation of complement reduction: their pointer structure can be interpreted as representing a PQ-tree. In our view, adding the PQ-tree abstraction makes the algorithm easier to understand. Also, their reduction method walks down the tree and reduces it node-by-node. This seems to limit the method to handling complement reduction, and it needs extra pointers, to support both bottom-up and top-down walks on the tree.

## 4 A Simple Linear-Time Implementation

In this section we describe a linear-time planarity test that simplifies the Shih-Hsu and Boyer-Myrvold methods. The test adds vertices in a postorder with respect to a depth-first spanning tree [Tar72]. This has the advantage that we can break the computation described in Section 2 into simpler steps, and we can do the computation on the fly during a single depth-first search of the graph. Each time the search retreats along a tree arc, backing up from  $w$  to  $v$ , we do a reduction on the component containing  $w$  with respect to the edges joining  $v$  and this component, and we do the part of the combining step corresponding to this component. We collect the edges on which the reduction is done as the search proceeds. This approach avoids some preprocessing done by Shih and Hsu and Boyer and Myrvold. We maintain one PQ-tree for each ancestor in the depth-first spanning tree of the current vertex of the search, not including the root, and we maintain for each ancestor  $v$  of the current vertex a set  $S(v)$  of PQ-tree leaves; eventually a reduction is done on this set. Here are the details of the algorithm:

When advancing along a tree arc  $(v, w)$ , construct a PQ-tree for  $w$  consisting of a root that is the special leaf and represents  $(v, w)$ , with a single child that is a P-node and represents  $w$ . Initialize  $S(v)$  to contain just the special leaf.

When advancing along a back arc  $(v, w)$ , add a leaf child representing  $(v, w)$  to the P-node for  $v$ , and add this leaf to  $S(w)$ .

When retreating along a tree arc  $(v, w)$ , reduce the PQ-tree for  $w$  to represent the circular orders such that the leaves in  $S(v)$  are consecutive. If this produces the null tree, stop and declare the graph non-planar. Otherwise, delete from the reduced tree all the leaves in  $S(v)$ , simplify the tree by deleting P-nodes and Q-nodes with no children and repeating until all P-nodes and Q-nodes have

children, and do the appropriate part of the combining step by making the root of the simplified tree a child of the P-node for  $v$ , unless  $v$  is start vertex of the depth-first search.

A single search suffices to test the planarity of a connected graph; if the graph is disconnected, we repeat the test on each connected component. The algorithm automatically handles multiple edges, loops, and graphs that are not biconnected. We can avoid storing the leaves and roots of the PQ-trees explicitly, at the cost of making the implementation of PQ-tree reduction a little more complicated.

## 5 Remarks

Our algorithm extends in various ways. If instead of deleting nodes during reduction and simplification, we remove and save the subtrees containing these nodes, we get a linear-time algorithm that produces a representation of all possible embeddings. We can also generate a single embedding, either arbitrary or uniformly randomly chosen, or count embeddings. An alternative way to view the vertex-embedding process is as the time reversal of a process of contracting a graph to a single vertex by contracting edges, which might offer some conceptual advantages.

Our interpretation of PQ-trees as representing sets of either linear or circular orders offers the possibility of efficiently solving ordering problems with mixed constraints. In particular, suppose we are given a zero-one matrix and asked whether there is a permutation of the rows such that each column satisfies either a consecutive-ones constraint or a circular-ones constraint. (See [BL76].) We can test for the existence of such a permutation in time linear in the number of ones in the matrix (assuming a sparse matrix representation).

Our work suggests at least two possible directions for further research, one concerning planarity-testing, the other concerning PQ-trees. Is there a way to unify and simplify path-addition planarity tests such as those of Hopcroft and Tarjan [HT74] and Fraysseix, Mendez and Rosenstiehl [dFdMR06]? Is there a way to obtain the path-addition and vertex-addition methods as different versions of a more-general approach? Is there an efficient way to re-root a PQ-tree and so allow more general ways to combine such trees efficiently? This would allow a more-general strategy for planarity-testing to run in linear time.

### Acknowledgment

The second author thanks Janet S. Yoon for contributions to an early phase of this work.

## References

- [AP61] L. Auslander and S. V. Parter. On imbedding graphs in the plane. *Journal of Mathematics and Mechanics*, 10(3):517–523, 1961.
- [BCPDB03] J.M. Boyer, P.F. Cortese, M. Patrignani, and G. Di Battista. Stop minding your P’s and Q’s: Implementing a fast and simple DFS-based planarity testing and embedding algorithm. *Proceedings of the 11th International Symposium on Graph Drawing, Lecture Notes Computer Science*, 2912:25–36, 2003.
- [BFNdP04] J.M. Boyer, C.G. Fernandes, A. Noma, and J.C. de Pina. Lempel, Even, and Cederbaum Planarity Method. *Proceedings of the 3rd Workshop on Experimental Algorithms, Lecture Notes in Computer Science*, 3059:129–144, 2004.
- [BFNdPJ03] J.M. Boyer, C.G. Fernandes, A. Noma, and J.C. de Pina Jr. Correcting and Implementing the PC-tree Planarity Algorithm. <http://citeseer.ist.psu.edu/626629.html>, 2003.

- [BL76] 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 System Sciences*, 13(3):335–379, 1976.
- [BM99] J. Boyer and W. Myrvold. Stop minding your P’s and Q’s: a simplified O (n) planar embedding algorithm. *Proceedings of the 10th annual ACM-SIAM symposium on Discrete algorithms*, pages 140–146, 1999.
- [BM04] J.M. Boyer and W.J. Myrvold. On the Cutting Edge: Simplified O (n) Planarity by Edge Addition. *Journal of Graph Algorithms and Applications*, 8(3):241–273, 2004.
- [Boo75] K.S. Booth. *PQ-Tree Algorithms*. PhD thesis, University of California, Berkeley, 1975.
- [Boy05] J.M. Boyer. Additional PC-Tree Planarity Conditions. *Proceedings of the 12th International Conference on Graph Drawing 2004, Lecture Notes Computer Science*, 3383:82–88, 2005.
- [dFdMR06] H. de Fraysseix, P.O. de Mendez, and P. Rosenstiehl. Trémaux trees and planarity. *International Journal of Foundations of Computer Science*, 17:1017, 2006.
- [ET76] S. Even and R.E. Tarjan. Computing an st-numbering. *Theoretical Computer Science*, 2(3):339–344, 1976.
- [HM03] W.L. Hsu and R.M. McConnell. PC trees and circular-ones arrangements. *Theoretical Computer Science*, 296(1):99–116, 2003.
- [HM04] W.L. Hsu and R. McConnell. PQ Trees, PC Trees and Planar Graphs, a book chapter in *Handbook of Data Structures and Applications*, edited by Dinesh P. Mehta and Sartaj Sahni, 2004.
- [Hsu03] W.L. Hsu. An Efficient Implementation of the PC-tree Algorithm of Shih and Hsu’s Planarity Test. Technical report, Technical Report TR-IIS-03-015, Institute of Information Science, Academia Sinica, July 2003, 2003.
- [HT74] J. Hopcroft and R. Tarjan. Efficient planarity testing. *Journal of the ACM*, 21:549–568, 1974.
- [Kur30] K. Kuratowski. Sur les problemes des courbes gauches en Topologie. *Fund. Math*, 15:271–283, 1930.
- [LEC67] A. Lempel, S. Even, and I. Cederbaum. An algorithm for planarity testing of graphs. In Rosenstiehl, P., editor, *Theory of Graphs: International Symposium*, pages 215–232, New York, 1967. Gordon and Breach.
- [Lue75] G.S. Lueker. *Efficient Algorithms for Chordal Graphs and Interval Graphs*. PhD thesis, Princeton University, 1975.
- [SH93] W.K. Shih and W.L. Hsu. A simple test for planar graphs. *Proceedings of the International Workshop on Discrete Mathematics and Algorithms*, pages 110–122, 1993.
- [SH99] W.K. Shih and W.L. Hsu. A new planarity test. *Theoretical Computer Science*, 223(1-2):179–191, 1999.
- [Tar69] R.E. Tarjan. Implementation of an Efficient Algorithm for Planarity Testing of Graphs. unpublished, 1969.
- [Tar72] R. Tarjan. Depth-First Search and Linear Graph Algorithms. *SIAM Journal on Computing*, 1:146, 1972.
- [Tar85] R.E. Tarjan. Two Streamlined Depth-first Search Algorithms, 1985.