MAXPLANAR : a graphical software package for testing maximal planar subgraph algorithms
by Kedan Zhao

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Computer Science
Montana State University


Abstract:
We present an efficient implementation of a software package, MAXPLA-NAR, with a user-friendly interface for several algorithms for finding maximal planar subgraphs of nonplanar graphs. The algorithms include the methods of path addition, edge addition, vertex addition, and cycle packing.

MAXP LANAR is designed to facilitate graph input and output and algorithm efficiency analysis. The result is an easy-to-use software package for researchers to test the various planarization algorithms.

Extensive empirical results are given for the heuristics on several families of nonplanar graphs. Type I are random nonplanar graphs with unknown maximum planar subgraph size. Type II are sparse, planar-like graphs which are graphs that are almost planar. Type III instances are dense graphs. Results of empirical testing show that the cycle-packing algorithm found the best solution in random nonplanar graphs, but it required much more CPU time than the other heuristics. The results also show that for planar-like graphs the vertex addition method is better than the edge addition method. However, for dense random graphs the edge addition method is better than others.

# MAXPLANAR: A Graphical Software Package For Testing Maximal Planar Subgraph Algorithms

by

Kedan Zhao

A thesis submitted in partial fulfillment
of the requirements for the degree

of

**Master of Science**

in

**Computer Science**

Montana State University
Bozeman, Montana

January 1996

## APPROVAL

of a thesis submitted by

Kedan Zhao

This thesis has been read by each member of the thesis committee and has been found to be satisfactory regarding content, English usage, format, citations, bibliographic style, and consistency, and is ready for submission to the College of Graduate Studies.

2-7-96
Date

Robert Cimbrowski
Chairperson, Graduate Committee

Approved for the Major Department

2/8/96
Date

J Dunbigh Stanley
Head, Major Department

Approved for the College of Graduate Studies

4/21/96
Date

R L Brown
Graduate Dean

## STATEMENT OF PERMISSION TO USE

In presenting this thesis in partial fulfillment of the requirements for a master's degree at Montana State University, I agree that the Library shall make it available to borrowers under rules of the Library.

If I have indicated my intention to copyright this thesis by including a copyright notice page, copying is allowable only for scholarly purposes, consistent with "fair use" as prescribed in the U.S. Copyright Law. Requests for permission for extended quotation from or reproduction of this thesis in whole or in parts may be granted only by the copyright holder.

Signature Kedar Khand

Date 2-8-96

# ACKNOWLEDGMENTS

I would like to take this opportunity to thank my graduate committee members, Dr. Robert Cimikowski, Dr. Gary Harkin, and Prof. Ray Babcock, and the rest of the faculty members from the Department of Computer Science for their help and guidance during my graduate program. I would also like to thank my thesis advisor, Dr. Robert Cimikowski, for his encouragement and support while supervising this thesis.

Much of the credit for completion of this project goes to my wife Qian Cai and my son Andy. Their love, support and sacrifice gave me the opportunity to spend two years in graduate school.

Special thanks go to my father Jimin Zhao and my mother Xiouzhen Zhang, for their encouragement and support.

# Contents

# List of Tables

# List of Figures

## Abstract

We present an efficient implementation of a software package, MAXPLANAR, with a user-friendly interface for several algorithms for finding maximal planar subgraphs of nonplanar graphs. The algorithms include the methods of path addition, edge addition, vertex addition, and cycle packing.

MAXPLANAR is designed to facilitate graph input and output and algorithm efficiency analysis. The result is an easy-to-use software package for researchers to test the various planarization algorithms.

Extensive empirical results are given for the heuristics on several families of nonplanar graphs. Type I are random nonplanar graphs with unknown maximum planar subgraph size. Type II are sparse, planar-like graphs which are graphs that are almost planar. Type III instances are dense graphs. Results of empirical testing show that the cycle-packing algorithm found the best solution in random nonplanar graphs, but it required much more CPU time than the other heuristics. The results also show that for planar-like graphs the vertex addition method is better than the edge addition method. However, for dense random graphs the edge addition method is better than others.

# Chapter 1

# Introduction

## 1.1 Planarity and Graph Planarization

Given an undirected graph, the planarity testing problem is to determine whether the graph can be drawn in the plane without any crossing edges. It has many applications, e.g. in the design of VLSI circuits, in printed circuit board layout, automated graphical display systems, determining the isomorphism of chemical structures, and in various problems dealing with the display and readability of diagrams. A few planarity testing algorithms of different types are known. They all have linear time complexity. Of the two major algorithms, one is called a "path addition" algorithm and the other is called a "vertex addition" algorithm. These terms refer to the approaches used in the algorithms. The path addition algorithm is originally due to Auslander and Parter [1] and a linear time implementation was developed by Hopcroft and Tarjan [2]. The vertex addition algorithm, which was presented first by Lempel, Even and Cederbaum[3], and improved later to a linear algorithm by Booth and Lueker[4], uses a novel data structure called the PQ-tree.

If a graph is not planar, then we may want to delete some edges to obtain a planar subgraph. This process of removing a set of edges from a nonplanar graph $G$ to obtain a planar subgraph is known as *planarization* of the nonplanar

graph $G$. On the other hand, *maximal planarization* of a nonplanar graph $G$ refers to the process of deleting a minimal set of edges of $G$ in order to obtain a planar subgraph.

## 1.2 Applications of Planarization

Maximal planarization of a nonplanar graph is an important problem encountered in the automated design of printed circuit boards. If an electronic circuit cannot be wired on a single layer of a printed circuit board, then we would like to determine the minimum number of layers necessary to wire the circuit. Since only a planar circuit can be wired on a single layer board, we would like to decompose the nonplanar circuit into a minimum number of planar circuits. In general, for a nonplanar graph, neither the set of edges to be removed to maximally planarize it nor the number of these edges is unique.

Finding the *minimum* number of edges whose deletion from a nonplanar graph gives a planar subgraph was shown to be NP-complete[5]; hence research has focused on computing a *maximal planar subgraph* $G'$ of $G$, that is, a subgraph $G'$ such that for all edges $e \in G - G'$, the addition of $e$ to $G'$ destroys the planarity. The first algorithm for this problem runs with $O(mn)$ worst-case time bound[6]. Here $m$ is the number of edges and $n$ is the number of vertices. This algorithm starts with one edge and check for every subsequent edge, whether its addition to the graph preserves the planarity (by employing a linear-time planarity tester). Recently Cai, Han and Tarjan described an $O(m \log n)$ maximal planarization algorithm[8], based on the Hopcroft-Tarjan planarity testing algorithm. Di Battista and Tamassia described an incremental algorithm to check in $O(\log n)$ amortized time whether adding an edge to the graph preserves planarity, which yields an $O(m \log n)$ time maximal planarization algorithm as well. Ozawa and Takahashi[9] proposed another

$O(mn)$ time and $O(m+n)$ space algorithm to planarize a nonplanar graph using the PQ-tree implementation[4] of Lempel, Even, and Cederbaum's planarity testing algorithm[3]. However, for a general graph this algorithm may not determine a maximal planar subgraph[10]. Moreover, in certain cases, this algorithm may terminate without considering all the vertices; in other words, it may not produce a *spanning* planar subgraph[11]. Jayakumar, Thulasiraman and Swamy[11] presented an $O(n^2)$ planarization algorithm for a special class of graphs based on PQ-trees. However, this algorithm contains some errors. Kant give a correct version that can be implemented to run in $O(n^2)$ time[12], which is better than the time bound $O(m \ log \ n)$ for dense graphs. Moreover, instead of testing for every edge whether or not it can be added without destroying the planarity, it calculates for every vertex the minimum number of edges which must be deleted to preserve planarity. The maximal planarization algorithms were developed independently and in their original form by various reseachers, and some of them were implemented in PASCAL and some in C. Until now there was no unified software package that includes all of them. We present an efficient C language implementation of a software package with a user-friendly interface for several algorithms for finding maximal planar subgraphs of nonplanar graphs. The algorithms include the methods of path-embedding, edge-embedding, vertex-embedding, and cycle packing. The algorithms are well known but have not previously been organized and implemented as a single, cohesive, integrated software package in an efficient manner. A user-friendly interface was also developed to facilitate graph input and output and algorithm efficiency analysis. The result is an easy-to-use software package for researchers to use in applications requiring the maximal planarization of graphs.

## 1.3 Thesis Outline

In this thesis our objective was to develop a unified software package for all of the major graph planarization algorithms in order that empirical testing and a comparative analysis of the methods would be more easily performed. To facilitate the process, we developed a front-end graphical interface to the algorithms using Motif and X-windows.

The paper is organized as follows. In Chapter 2 we present definitions and other basic concepts. In Chapter 3 we discuss the vertex-addition algorithm. In Chapter 4 and Chapter 5 we describe the path- and edge- addition algorithms and in Chapter 6 we present with the cycle-packing algorithm. In Chapter 7 we discuss the graphical user interface, and in Chapter 8 we discuss experimental testing and offer concluding remarks.

# Chapter 2

# Definitions and Preliminaries

## 2.1 Definitions

Let $G = (V, E)$ be a graph with vertex set $V(G)$ and edge set $E(G)$. We assume that $G$ is *simple*, that is, has no multiple edges or loops. Throughout this paper $n$ denotes the number of vertices of $G$, that is, $n = |G|$ and $m$ denotes the number of edges of $G$, that is, $m = |E|$. If there is a graph $H$ with vertex set $V(H)$ and edge set $E(H)$ then we say that $H$ is a subgraph of $G$ if $V(H) \subseteq V(G)$ and $E(G) \subseteq E(G)$.

A graph $G$ is *connected* if every pair of points are joined by a path. In this paper we always assume the graph is connected. If a graph is disconnected then all of the algorithms discussed in this thesis can be applied to the individual components one by one.

A graph $G$ is *planar* if it is embeddable in the plane without any crossing edges.

A *biconnected component* of $G$ is a maximal set of edges such that any two edges in the set lie on a common simple cycle. Figure 2.1 illustrates this definition.

A graph $G$ is called *k-connected*, $k \geq 3$, if $G$ is simple, $|V(G)| \geq k+1$, and the graph obtained from $G$ by deleting any $k-1$ vertices is connected.

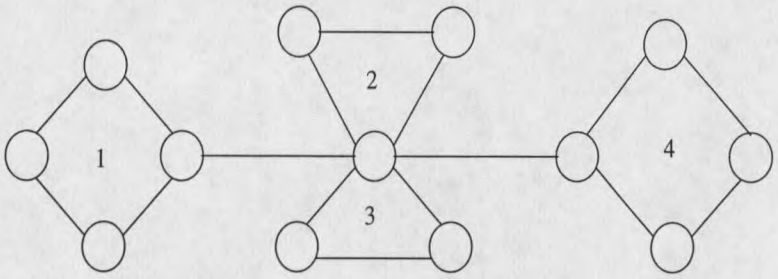


Figure 2.1: Four biconnected components

A graph is planar if and only if all of its biconnected components are planar.

We call an algorithm *randomized* if its behavior is determined not only by the input but also by value produced by a random number generator.

## 2.2 Representations of Graphs

There are two standard ways to represent a graph $G = (V, E)$, as a collection of adjacency lists or as an adjacency matrix. The adjacency list provides a compact way to represent *sparse* graphs, i.e., $|E| = O(n)$. All of the algorithms presented in this paper assume that an input graph is represented in adjacency list form. An adjacency matrix representation may be preferred, however, when the graph is dense, or when one needs to be able to tell quickly if there is an edge joining two vertices.

The adjacency list representation of a graph $G = (V, E)$ consists of an array $Adj[]$ of $n$ lists, one for each vertex in $V$. For each $u \in V$, the adjacency list $Adj[u]$ contains (pointers to) all the vertices $v$ such that there is an edge $(u, v) \in E$. That is, $Adj[u]$ consists of all the vertices adjacent to $u$ in $G$. The vertices in each adjacency list are typically stored in an arbitrary order. Figure 2.2 is an adjacency list representation of an undirected graph.

A potential disadvantage of the adjacency list representation is that there

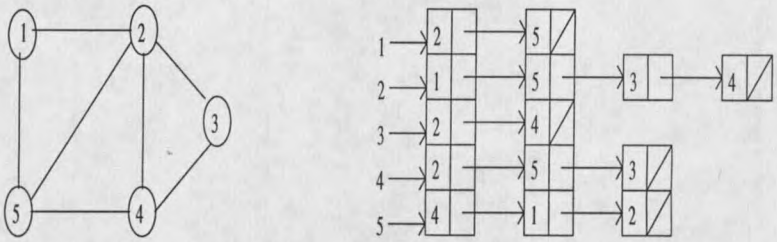


Figure 2.2: Adjacency list representation

is no quicker way to determine if a given edge $(u, v)$ is present in the graph than to search for $v$ in the adjacency list $Adj[u]$. This disadvantage can be remedied by an adjacency matrix representation of the graph, at the cost of using a factor of $n$ more memory.

For the adjacency matrix representation of a graph $G = (V, E)$, we assume that the vertices are numbered 1, 2, ..., $n$ in some arbitrary manner. The adjacency matrix representation of a graph $G$ then consists of a $n \times n$ matrix $A = (a_{ij})$ such that $a_{ij} = 1$ if $\{i, j\} \in E$ and $a_{ij} = 0$ otherwise.

Figure 2.3 is the adjacency matrix representation of an undirected graph. The adjacency matrix representation of a graph requires $O(n^2)$ memory, independent of the number of edges in the graph.

## 2.3 Some Useful Theorems

A *Jordan curve* in the plane is a continous non-self-intersecting curve whose origin and terminus coincide. For example, in Figure 2.4, the curve $C_1$ is not a Jordan curve because it intersects itself, $C_2$ is not a Jordan curve since its origin and terminus do not coincide, i.e., its two end points do not meet but $C_3$ is a Jordan curve.

If $J$ is a Jordan curve in the plane then the part of the plane enclosed by

| | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| 1 | 0 | 1 | 0 | 0 | 1 |
| 2 | 1 | 0 | 1 | 1 | 1 |
| 3 | 0 | 1 | 0 | 1 | 0 |
| 4 | 0 | 1 | 1 | 0 | 1 |
| 5 | 1 | 1 | 0 | 1 | 0 |

Figure 2.3: Adjacency maxtrix representation

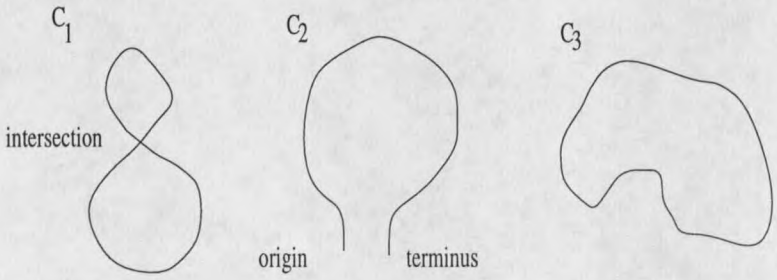


Figure 2.4: $C_1$ and $C_2$ are not Jordan curves but $C_3$ is.

$J$ is called the *interior* of $J$ and denoted by *int* $J$. We exclude from *int* $J$ the points actually lying on $J$. Similarly the part of the plane lying outside $J$ is called the *exterior* of $J$ and denoted by *ext* $J$.

**Jordan Curve Theorem.** If $J$ is a Jordan curve, $x$ is a point in *int* $J$ and $y$ is a point in *ext* $J$ then any line joining $x$ to $y$ must meet $J$ at some point, i.e., must cross $J$.

A *hamiltonian path* in a graph $G$ is a simple path which contains every vertex of $G$. No vertex of a path is repeated. A *hamiltonian cycle* in G with initial vertex $v$ contains every vertex of $G$ precisely once and then ends up back at the initial vertex.

A graph is called *hamiltonian* if it has a hamiltonian cycle.

By simply deleting the last edge of a hamiltonian cycle we get a hamiltonian path. However a nonhamiltonian graph may possess a hamiltonian path, i.e., hamiltonian paths cannot always be used to form hamiltonian cycles. For example, in Figure 2.5, $C_1$ has no hamiltonian path or hamiltonian cycle; $C_2$ has the hamiltonian path $b, a, c, d$ but no hamiltonian cycle, while $C_3$ has the hamiltonian cycle $a, b, d, c, a$.

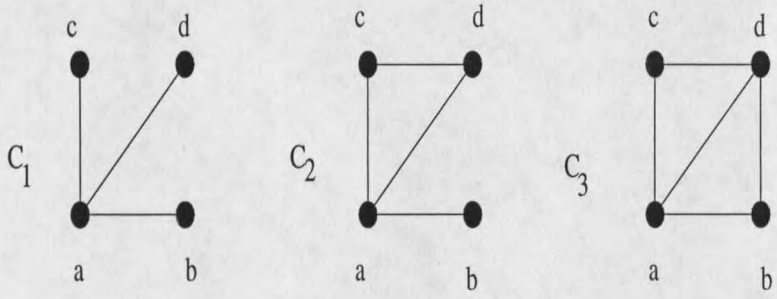


Figure 2.5: $C_1$ has no hamiltonian path, $C_2$ has a hamiltonian path but no hamiltonian cycle, while $C_3$ has a hamiltonian cycle.

An *independent set* of a graph $G = (V, E)$ is a subset $V' \in V$ of vertices