A branch-and-bound algorithm for the crossing number of a graph

by Zheng Tan

Abstract:
Determining the crossing number of a graph is a well-known NP-Hard problem. This thesis will present a branch-and-bound algorithm for finding the crossing number of a graph and the details required to implement it. To the author's knowledge, this is the only implemented algorithm to find the crossing number of a graph.

The algorithm begins with the vertex set and adds edges by selecting every legal option for creating a crossing or not and checks if the resulting partial graph is planar. At the point at which all edges have been added or at the point where the graph cannot be drawn without a crossing, the algorithm backtracks to see whether the graph can be drawn with fewer crossings by trying other options. Nicholson's heuristic for the linear crossing number of a graph is used as an initial upper bound for the algorithm.

The algorithm is shown to be effective providing that the size of the input graph is relatively small, i.e., no more than approximately 20 edges.

A BRANCH-AND-BOUND ALGORITHM FOR THE CROSSING NUMBER OF A GRAPH

by

Zheng Tan

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

of

Master of Science

in

Computer Science

MONTANA STATE UNIVERSITY
Bozeman, Montana

June 2000

N378
T15

APPROVAL

of a thesis submitted by

Zheng Tan

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.

Robert Cimikowski	Robert Cimikowski (Signature)	4-5-2000 Date

Approved for the Department of Computer Science

J. Denbigh Starkey	[illegible] (Signature)	4/5/2000 Date

Approved for the College of Graduate Studies

Bruce McLeod	Bruce R. McLeod (Signature)	6-26-00 Date

STATEMENT OF PERMISSION TO USE

Signature [illegible]

Date 6/10/2000

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Determining the crossing number of a graph is a well-known NP-Hard problem. This thesis will present a branch-and-bound algorithm for finding the crossing number of a graph and the details required to implement it. To the author's knowledge, this is the only implemented algorithm to find the crossing number of a graph.

The algorithm begins with the vertex set and adds edges by selecting every legal option for creating a crossing or not and checks if the resulting partial graph is planar. At the point at which all edges have been added or at the point where the graph cannot be drawn without a crossing, the algorithm backtracks to see whether the graph can be drawn with fewer crossings by trying other options. Nicholson's heuristic for the linear crossing number of a graph is used as an initial upper bound for the algorithm.

The algorithm is shown to be effective providing that the size of the input graph is relatively small, i.e., no more than approximately 20 edges.

CHAPTER 1

INTRODUCTION

Topological graph theory is a beautiful and profound subject in which most problems are conceptually simple yet computationally intractable. Among the problems in this field, determining the planar crossing number of a graph is an important problem with applications in areas such as circuit design and network configuration. Producing drawings of graphs with a small number of crossings can influence the area of the layout as well as the number of wire-contact cuts necessary. Little is known in general about planar crossing number, and exact values of planar crossing numbers are known only for very restricted classes of graphs. Once the mathematician Zarankiewicz believed he had solved the crossing number problem for the complete graph [15], but a gap was found in his proof, and the problem became the notorious unsolved question that it remains today. It is this importance that has driven the work in finding algorithms for computing the crossing number of a graph.

Definitions

In this work, we consider only simple, undirected, unweighted graphs G = (V, E) whose V is the set of *vertices* (nodes) and E is the set of *edges* between pairs of vertices.

Throughout the thesis, we let $n = |V|$ and $m = |E|$. For a good basic reference on general graph theoretic terminology, see [3].

We now define some terms from topological graph theory which will be used throughout the rest of this thesis.

Crossing Number

The *crossing number problem* is that of determining, for a given integer K, whether a graph G can be embedded in the plane with K or fewer pairwise crossings of the edges (not including the intersections of the edges at their common end points). The *planar crossing number*, $\nu(G)$, of a graph G is the minimum such K. Henceforth, we refer to the *planar crossing number* of a graph as simply the *crossing number*, for convenience. The *crossing minimization problem* is that of embedding a graph in the plane with the minimum number of edge-crossings among all good drawings, where a good drawing has the following properties:

(a) No edge crosses itself

(b) No pair of adjacent edges cross

(c) Two edges cross at most once

(d) No more than two edges cross at one point

Applications of the Crossing Number

There are several applications of the crossing number problem, of which we mention two here.

In graph drawing visualization problems, minimizing the number of crossings is an aesthetic criterion used to measure the quality of a graphical display. Too many crossings in a graphical diagram hinder its comprehensibility [9]. Also, in optimal circuit layout, given a graph or network, the goal is to embed it in the plane so as to minimize the number of wire crossings, since crossings may cause interference between wires.

Graph-Theoretic Terminology

For our purposes, a compact-orientable *2-manifold*, or simply a *surface*, may be thought of as a sphere or a sphere with handles. The *genus* of the surface is the number of handles.

An *embedding* of a graph G on a surface S is a drawing of G on S in such a manner that edges intersect only at a vertex to which they are both incident.

A *face* (region) in an embedding is called a *2-cell* if any simple closed curve in that face (region) can be continuously deformed or contracted in that face (region) to a single point. An embedding is called a *2-cell embedding* if all the regions in the embedding are 2-cell.

The algebraic description of a 2-cell embedding if referred to as a Rotational Embedding Scheme and will be covered in the next section.

Theorems

Euler's Formula

The relationship between the number of regions of a graph and the surface on which it is embedded is described by the well-known generalized *Euler's Formula*:

Let G be a connected graph with n vertices and m edges with a 2-cell embedding on the surface of genus g having r regions. Then $n - m + r = 2 - 2g$.

Rotational Embedding Scheme

With these definitions as background, we can now look at the Rotational Embedding Scheme, first formally introduced by Edmonds [10] and then discussed in detail by Youngs [38] a few years later. The following is the formal statement of the Rotational Embedding Scheme as given by Chartrand and Lesniąk [4].

Let G be a nontrivial connected graph with $V(G) = \{ v_1, v_2, \ldots, v_n \}$. For each 2-cell embedding of G on a surface there exists a unique n-tuple $(\pi_1, \pi_2, \ldots, \pi_n)$, where for i = 1, 2, ..., n, π_i: $V(i) \rightarrow V(i)$ is a cyclic permutation that lists the subscripts of the vertices adjacent to v_i. Conversely, for each such n-tuple $(\pi_1, \pi_2, \ldots, \pi_n)$, there exists a 2-cell embedding of G on some surface such that for i = 1,2, ..., n the subscripts of the vertices adjacent to v_i and in the counterclockwise order about v_i, are given by π_i.

For example, consider Figure 1 which gives a planar embedding of a graph. From this graph we obtain the following counterclockwise permutations associated with each vertex:

$\pi_1 = (6, 4, 2)$ $\pi_2 = (1, 4, 3)$

$\pi_3 = (2, 4)$ $\pi_4 = (3, 2, 1, 5)$

$\pi_5 = (4, 6)$ $\pi_6 = (5, 1)$

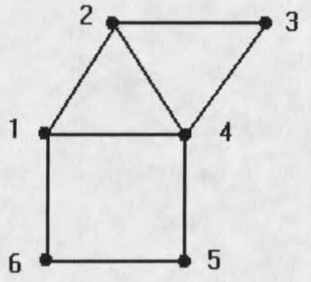

Figure 1. A planar embedding of a graph.

From these permutations we can obtain the edges of the graph and the number of regions of the graph. For instance, this graph has 4 regions. The edges for one of these regions can be traced as follows:

1) Start with edge (1, 2).
2) From permutation π_2 determine which vertex follows 1; it is 4. Therefore the second edge is (2, 4).
3) From permutation π_4 determine which vertex follows 2; it is 1. Therefore the third edge is (4, 1).
4) From permutation π_1 determine which vertex follows 4; it is 2. This yields edge (1, 2) which was the original edge, so we are finished.

The region we considered is bounded by the edges (1, 2), (2, 4), and (4, 1). The other regions and associated edges can be found in a similar manner.

The important thing to note is the converse portion of the Rotational Embedding Scheme—that every collection of vertex permutations corresponds to an embedding on some surface. Given a set of permutations, we can trace the edges of the regions and determine the genus of the surface on which the graph is embedded.

Upper bounds for the Crossing Number

The exact value of the crossing number has not yet been determined for all complete graphs or complete bigraphs; except for the first few instances, only upper bounds are known. The prevailing conjecture is that the bounds in Theorem 1.1 and Theorem 1.2 are exact.

Theorem 1.1 The crossing number of the complete graph satisfies the inequality

$$\nu(K_n) \leq \tfrac{1}{4} \lfloor n/2 \rfloor \lfloor (n-1)/2 \rfloor \lfloor (n-2)/2 \rfloor \lfloor (n-3)/2 \rfloor.$$

Equality has been shown for $n \leq 10$ by Saaty.

Theorem 1.2 The crossing number of the complete bigraph satisfies the inequality

$$\nu(K_{m,n}) \leq \tfrac{1}{4} \lfloor m/2 \rfloor \lfloor (m-1)/2 \rfloor \lfloor n/2 \rfloor \lfloor (n-1)/2 \rfloor.$$

Equality has been shown for $m \leq 6$ by Kleitman, [22].

In Table 1 are listed the conjectured crossing numbers of the complete graph K_n.

Table 1. Conjectured Values of Crossing Numbers of K_n

n	13	18	21	24	27	9n + 7
$\nu(K_n)$	7	15	21	28	36	$(9n^2 + 13n + 2) / 2$

Discussion of Previous Results

Although the crossing number problem is easily stated and has been well studied, not much is known about its algorithmic solution. Unfortunately, computing the crossing number of a graph is NP-complete [13]. This implies that the crossing minimization problem is very likely to be intractable. Because of the difficulty of the problem, attention has turned away from finding the minimum crossing number of a graph to subproblems and other related problems. With regard to the subproblems, there has been work on product graphs ranging from the product of C_n and graphs of order four to C_3 x C_n, C_4 x C_4, C_5 x C_5 and C_5 x C_n. In 1991 Beinstock [3] published work relating the crossing number of a graph to the arrangement of pseudolines, a topic well studied by combinatorialists.

The crossing number problem includes several variations which have been studied extensively. First, Nicholson developed a heuristic algorithm for the crossing minimization problem [29]. His algorithm finds a special type of embedding of a given graph, namely, (i) the vertices are placed on a horizontal line l, and (ii) the edges are drawn by semicircles (see Figure 2(b)). We call this type of embedding a *linear embedding* and the minimum number of crossings, the linear crossing number. Despite the restrictions, the problem is still NP-Complete [28]. However Nicholson's algorithm finds embeddings for small complete graphs and complete bipartite graphs that are nearly optimal for any style of embedding. But, it does not always produce good results for general graphs. One way to improve his solution for the general case is to reassign the semicircles to either side of *l* after the positions of the vertices on *l* are determined. This motivates researchers in the graph theory community to consider the crossing minimization problem with an additional constraint, (iii) the positions of the vertices on *l* are predetermined and fixed.

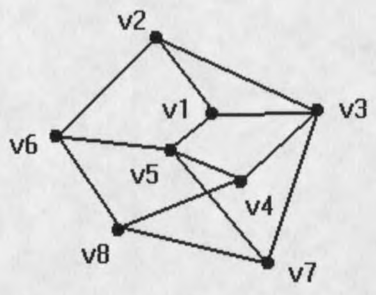

Figure 2. (a) A simple graph G.

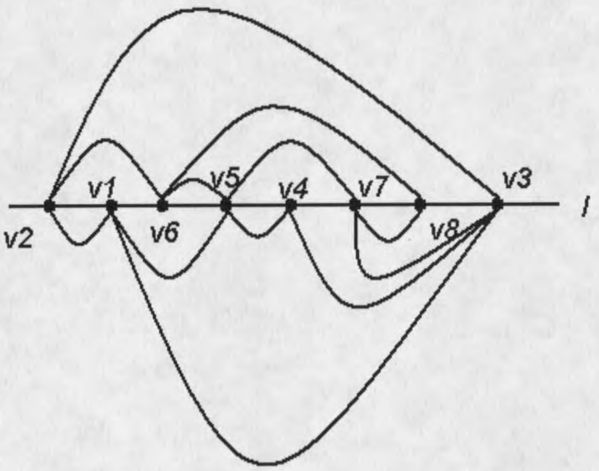

Figure 2. (b) A linear embedding of G.

The *fixed linear crossing minimization problem* is that of finding a linear embedding of a graph with the minimum number of edge-crossings under a specified vertex ordering. We call the problem of finding such an embedding with no vertex ordering specified the *free linear crossing minimization problem.* [6] presents several heuristics and an exact branch-and-bound algorithm for the *fixed linear crossing mnimization problem.*

The linear crossing number problems are related to the *book embedding problems* which have recently attracted considerable attention. A *book embedding* of a graph is an embedding in a book with the vertices placed on the spine and the edges on the pages such that no two edges drawn on the same page cross each other. The objective is to minimize the number of pages necessary to embed the graph without crossings. One such algorithm of minimizing edge crossings in drawings of nonplanar graphs was presented in [5].

Another problem which is related to the fixed linear crossing number problem is the so-called *topological via minimization problem.* Slightly modifying the formulation [27], we can formulate the via minimization problem as that of finding a maximum subgraph having a fixed linear embedding with no edge-crossings under a specified vertex ordering.

The last variation of the crossing number problem we consider is a layout problem of computer communication networks. It is formulated graph-theoretically as follows: Given a graph $G = (V, E)$, find an embedding of G in the plane with the least number of edge-crossings such that (i) the vertices in V are placed on the circumference of a circle

C, and (ii) the edges in *E* are drawn only inside *C*. An important goal of computer communication network management tools is to provide users with information on the global configuration of a network in an appropriate form. A natural and reasonable method of presentation is to draw the network on a graphics screen so that the users can easily grasp its entire logical structure. As an example, an automatic network layout algorithm is provided in IBM's CNMgraf: Communications Network Management Graphics Facility [14]. Based on a classification of the links, the algorithm first partitions the set of the computing facilities into subsets, called 'sites'. It then draws the sites as points located on the circumference of a "primary" circle and the links between different sites as straight lines inside the circle. Similarly, for each site, the algorithm places its computing facilities on a "regional" circle and draws the links between the different facilities of the site as straight lines inside the regional circle. The final drawing of the entire network is completed by placing the regional circles around the primary circle.

In this approach, all circles must be arranged in such a way as to make the resultant layout of the network suitable for the users' inspection. For this purpose, the algorithm used in CNMgraf tries to find a drawing which results in as few crossings of links as possible for each of the circles. There exists a linear-time algorithm for embedding an outerplanar graph G in the particular form mentioned above in such a way that no edge-crossings occur. But for general graphs, the problem is NP-hard, which implies that the network layout problem is, in general, very likely to be intractable. The following figure is an example of a circle-confined drawing of a graph. When constraints (i) and (ii) are not imposed, the problem becomes an optimization version of the crossing number problem.

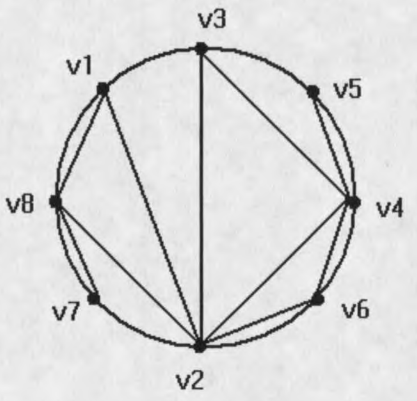

Figure 3. An example of a circle-confined drawing of a graph.

The purpose of this study is to bring the planar crossing number problem to the attention of the algorithmic graph theory community by providing a description of a branch-and-bound algorithm, and the details of its implementation. This thesis should be viewed as a point of departure for further research in this area.

CHAPTER 2

DESCRIPTION OF THE BRANCH-AND-BOUND ALGORITHM

DFS, Branch-and-Bound, and Backtracking

Branch and bound algorithms belong to the class of partial enumeration algorithms, which are exact algorithms for solving hard combinatorial optimization problems. This method turns out to be quite useful when solving many NP-hard optimization problems, e.g., Travelling Salesman Problem, graph coloring, maximum clique, minimum crossing number, etc.. Basically, partial enumeration does an exhaustive search of the solution space tree for an optimization problem, pruning infeasible or suboptimal branches of the tree as it proceeds. As a result of the pruning, only a small subset of the search tree is actually explored.

Finding the crossing number of a graph can be solved by exhaustively searching the solution space of all embeddings and saving the best solution. But the number of different plane embeddings of a graph is exponentially large in the number of edges of a graph. Hence, even for a branch-and-bound algorithm, the task is formidable. For many problems, notably NP-hard problems, the size of the search tree is exponential in the number of nodes. For example, to find a maximum clique in a graph given an n-vertex graph, we may have to check all 2^n subsets of vertices. By using partial enumeration, only a small number of the subsets will be typically explored by pruning most of the partial solution branches of the search tree.

Backtracking is another type of partial enumeration algorithm. Backtracking is actually a modified depth-first search (DFS) of a tree. DFS makes a preorder tree traversal. A path in the tree is followed as deeply as possible until a dead end is reached. At a dead end we back up until reaching a node with an unvisited child, then again proceed to go as deeply as possible. In the next section, we propose an algorithm to compute the crossing number of a graph. The first step of the algorithm finds each component of the graph. Depth-first search is used in this step. The recursive algorithm for finding components is shown below:

```
DFS(v)
{
    /* Graph G = (V,E) and array VISITED(1:n) initialized to 0*/
    /*  This algorithm will visit all the nodes reachable from v  */
    /*              G and VISITED are both global                 */
   VISITED(v) = 1;
   for ( Every node w connected to node v ) do
        if VISITED (w) = 0 then DFS (w) endif
   repeat
}
```

Backtracking, on the other hand, is the procedure where, after determining a node can lead to nothing but dead ends, " backtracks" to the node's parent and proceeds with the search on the next child. Backtracking is a recursive process, and therefore the implicit data structure is a stack. A "nonpromising" node is encountered if, when visiting the node, it is determined that it cannot possibly lead to an optimum solution; otherwise it is "promising".. If a node is found to be nonpromising, this node and all branches emanating downward from that node are cut off (pruned)from the tree and the search backs up to the next child of the parent node.

Branch-and-bound is another type of partial enumeration. At each node of the tree a numeric bound is computed for the best complete solution obtainable if the path from that node were to be extended to a leaf node. If that bound is no better than the value of the best solution found so far, the node is nonpromising; otherwise, it is promising. Thus, branches emanating from nonpromising nodes are pruned from the tree. At each level of the tree, the algorithm will compare the bounds of all promising nodes and visit next the children of the one with the best bound. In this way, in the end we will arrive at an optimum solution much faster than visiting every branch and node of the tree.

A Proposed Algorithm for Crossing Number

In the proposed algorithm we will map the solution space from the crossing number problem into a tree and search for the minimum crossing number using a DFS search with branch-and-bound. The root of the tree corresponds to the vertex set. The root has m (the number of edges) branches. Each branch corresponds to the first edge which is added to the graph. The next level of the tree has m - 1 branches from each node. The tree is m levels deep, and the path to each leaf corresponds to a unique permutation ordering for the edges.

The search process begins by selecting edge {i, j}. Euler's Formula is used to determine if the edge can be added from vertex i to vertex j without any crossings. We can do this by keeping track of the number of regions in the graph as it is constructed. If the edge can be added without crossings, we do so and pick the next edge. If not, we select an edge already added to cross with. This edge may be the only edge crossed, or the first edge in a long list of edges to be crossed. Once we decide that edge {i, j} is to cross with edge {k, l}, a new vertex m is created, we remove edge {k, l}, add partial edges {k, m}, {m, l}, and {i, m} and then draw edge {m, j} in the same way.

When drawing an edge, we know that an edge cannot cross itself, that no pair of adjacent edges cross and that two edges cross at most once. We need to keep track of all partial edges and remember that they are actually part of an original edge. In the case that the previous paragraph described, we must remember that partial edge {i, m} is part of edge {i, j}.

Next, the algorithm applied to K_5 will be demonstrated as an example. K_5 has 10 edges. The first 9 are added directly without crossing another edge. The partial graph is shown in Figure 4. When the last edge {i, j} is to be added, Euler's Formula indicates that the resulting graph cannot be embedded in the plane without crossings. Therefore, we must add the edge {i, j} with at least one crossing.

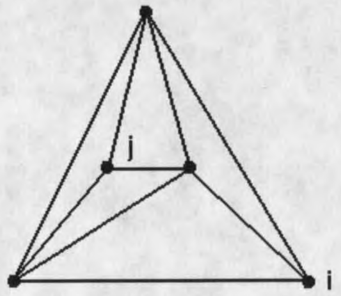

Figure 4. The first 9 edges of K_5 embedded with no crossings.