



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
© Copyright by Zheng Tan (2000)

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

© COPYRIGHT

by

Zheng Tan

2000

All Rights Reserved

N378
T15

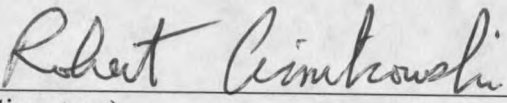
APPROVAL

of a thesis submitted by

Zheng Tan

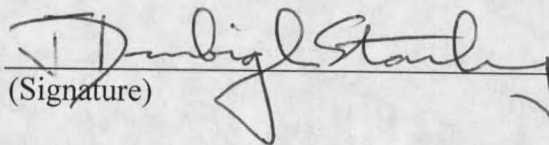
This thesis has been read by each member of the thesis committee and has been found to be satisfactory regarding content, English usage, format, citations, bibliographic style, and consistency, and is ready for submission to the College of Graduate Studies.

Robert Cimikowski


(Signature)4-5-2000
Date

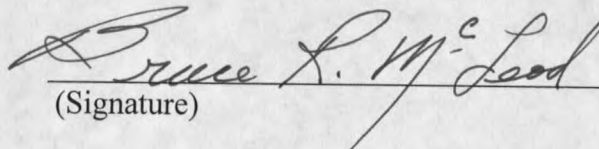
Approved for the Department of Computer Science

J. Denbigh Starkey


(Signature)4/5/2000
Date

Approved for the College of Graduate Studies

Bruce McLeod

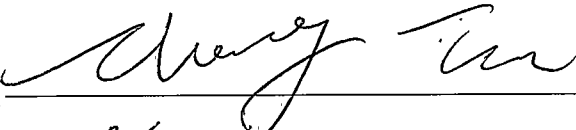

(Signature)6-26-00
Date

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 this 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



Date

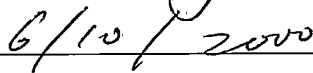


TABLE OF CONTENTS

1. INTRODUCTION.....	1
Definitions.....	1
Crossing Number.....	2
Applications of the Crossing Number.....	2
Graph-Theoretic Terminology.....	3
Theorems.....	3
Euler's Formula.....	3
Rotational Embedding Scheme.....	4
Upper bounds for the Crossing Number.....	6
Discussion of Previous Results.....	7
2. DESCRIPTION OF THE BRANCH-AND-BOUND ALGORITHM.....	12
DFS, Branch-and-Bound, and Backtracking.....	12
A Proposed Algorithm for Crossing Number.....	14
Time Complexity Analysis.....	19
Bounding Strategies.....	20
Heuristic Bounds.....	21
3. EXPERIMENTAL RESULTS.....	25
Description of the Test Graphs.....	25
Table of Results.....	27
4. CONCLUSIONS.....	28
Limitations of the Algorithm.....	28
Further Research.....	28
BIBLIOGRAPHY.....	30

LIST OF TABLES

Table	Page
1. Conjectured Values of Crossing Numbers of K_n	6
2. Results for Tested Graphs.	27

LIST OF FIGURES

Figure	Page
1. A planar embedding of a graph	5
2. (a) A simple graph G	8
(b) A linear embedding of G	8
3. An example of a circle-confined drawing of a graph.	11
4. The first 9 edges of K_5 embedded with no crossings	15
5. The first part of the last edge is embedded for K_5	16
6. A drawing of K_5 with 1 crossing at point m	16
7. (a) Hypercube Q_3	25
(b) Twisted cube TQ_3	25
(c) Crossed cube CQ_3	26
(d) Folded cube FLQ_3	26
(e) Pertersen graph	26
(f) Undirected de Bruijn graph DB_4	26
(g) Random hamiltonian graph $C10.2$	27

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*, $vc(G)$, 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 is 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 Lesniak [4].

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

$$\begin{array}{ll} \pi_1 = (6, 4, 2) & \pi_2 = (1, 4, 3) \\ \pi_3 = (2, 4) & \pi_4 = (3, 2, 1, 5) \\ \pi_5 = (4, 6) & \pi_6 = (5, 1) \end{array}$$

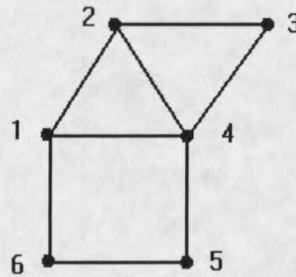


Figure 1. A planar embedding of a graph.

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

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

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

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

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

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	$9n + 7$
$v(K_n)$	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 $C_3 \times C_n$, $C_4 \times C_4$, $C_5 \times C_5$ and $C_5 \times C_n$. 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.

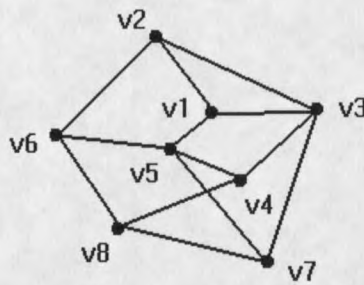


Figure 2. (a) A simple graph G .

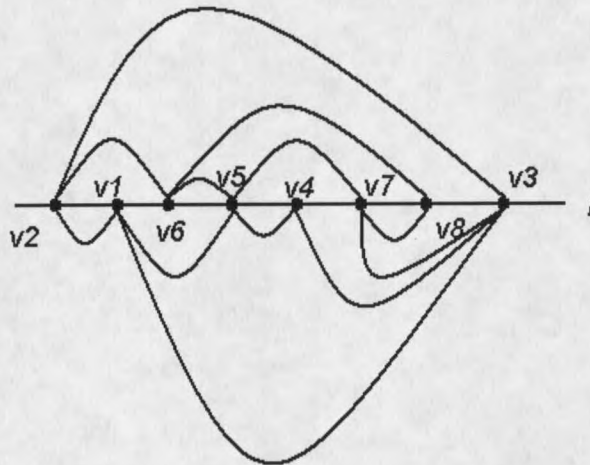


Figure 2. (b) A linear embedding of G .

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 minimization 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.

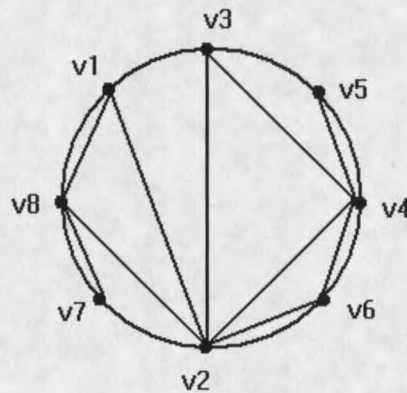


Figure 3. An example of a circle-confined drawing of a graph.

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.

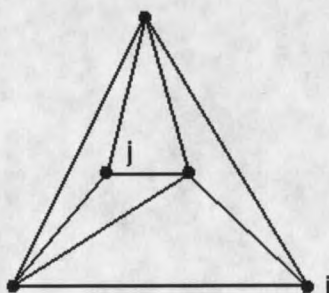


Figure 4. The first 9 edges of K_5 embedded with no crossings.

At this stage we pick edge $\{k, l\}$ to cross, insert vertex m , remove edge $\{k, l\}$, and add partial edges $\{k, m\}$, $\{m, l\}$ and $\{i, m\}$. The resulting graph is shown in Figure 5.

Once these partial edges have been drawn, the algorithm tries to draw the rest of edge $\{i, j\}$ without any crossings and succeeds. Figure 6 shows the drawing of K_5 with one crossing.

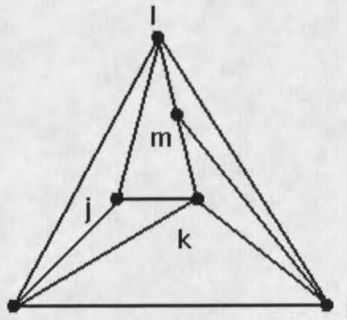


Figure 5. The first part of the last edge is embedded for K_5 .

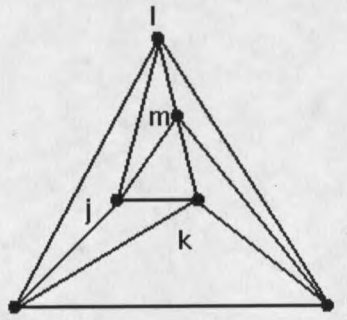


Figure 6: A drawing of K_5 with 1 crossing at point m .

The algorithm then backtracks with a new bound of one crossing. It will back up through the rest of the tree and try all other branches. But if the algorithm has

determined that it can't add an edge without any crossings, it will not go further down that branch because so far the best drawing has no crossing. Hence, that branch is pruned. The algorithm will stop when it has exhausted all possibilities. In the example of K_5 , it is impossible to get a drawing with no crossings.

Backtracking algorithm for the minimum crossing number of a graph:

Let $\nu(G)$ = crossing number of G .

Let $G = (V, E)$; $n = |V|$; $m = |E|$.

UB := *heuristic*(G); /* obtain initial upper bound for $\nu(G)$ */

for $i := 1$ to m do $E[i] := i$; /* fill edge vector */

backtrack(1,0);

backtrack ($i, ncross$)

{

if ($i > m$)

save drawing of G ; /* new upper bound */

UB := $ncross$; /* update UB */

else

for $j := i$ to m do

$k := E[i]$;

$E[i] := E[j]$;

$E[j] := k$;

$ncomp := component(G)$; /* find components of G */

for $k := 1$ to $ncomp$ do

if not *planar*(G_k)

compute candidate crossing set, $cset_i$, of edges for e_i ;

find all edges $e = uv$ such that e has no common

endpoints with $e_i = rs$;

replace edges e, e_i with new edges uw, wv, sw, wr and

new vertex w as crosspoint;

let the resulting graph be G_k' ;

let $c = |cset_i|$;

For $b := 1$ to c do

cross e_i with each of b edges in the b -subsets of $cset_i$,

adding b new vertices and $4b$ new edges and

deleting $2b$ edges;

add the new 'partial' edges E_b to the edge vector $E[]$;

```

        if the resulting graph is planar then break;
    if (  $b + ncross$  )  $\geq UB$  then
        delete  $E_b$  from  $E[]$ ;
        break;      /* backtrack */
    backtrack(  $i + 1, b + ncross$  );
     $k := E[i]$ ;
     $E[i] := E[j]$ ;
     $E[j] := k$ ;

    return;
}

planar(H)
{
    if  $|E(H)| < 9$  then
        return true;      /* trivially planar */
    else
        if  $|E(H)| > 3|V(H)| - 6$  then
            return false;  /* nonplanar - too many edges */
        else
            Find_Faces(H);  /* find all faces of H */
            apply Euler's formula to determine if H is planar;
            if planar(H) then
                return( true );
            else
                return( false );
}

component(G)
{
    find the connected components  $G_1, G_2, \dots, G_z$  of  $G$ ;
    return( $G_1, \dots, G_z$ );
}

Find_Faces(H)
{
    for(  $w = 1; w < \text{number of current edges}; w++$  )
        if ( the number of times accessed for this edge  $W_i < 2$  )
            /* Fact: Each edge is always part of two faces */
            {
                start with edge  $W_i = \{i, j\}$ ;
            }
}

```

from permutation π_j determine which vertex follows i ; let's say it is k , therefore the second edge is $\{j, k\}$;

from permutation π_k determine which vertex follows j ; let's say it is g , therefore the third edge is $\{k, g\}$.

...

continue these steps until the process yields the original edge $\{i, j\}$ and then increase the number of faces by one and increase the number of times accessed for each edge in this face by one.

otherwise, until all the edges have been accessed twice, start with another edge from the for-loop.

}

}

Time Complexity Analysis

From the way the rotational embedding scheme works, it is not difficult to see that if in one permutation the relative position among the vertices is not changed, the result, or the faces to be found, will stay the same. For example, $\pi_4 = (3, 2, 1, 5)$ is facially the same as $\pi_4 = (1, 5, 3, 2)$. So we can just ignore all cycles of the same permutation, e.g. $\pi_i = (v_1, v_2 \dots v_n)$, there will be $(n!-n)$ permutations for vertex i . if We define symbol $deg[i]$ to represent the *degree* of vertex i , that is, the number of vertices incident to i . For K_5 , the number of different vertex permutation schemes is:

$$\begin{aligned} & \prod_{i=1, n} (deg[i]! - deg[i]) \\ &= \prod_{i=1, 5} (4! - 4) = \prod_{i=1, 5} 20 = 2^5 * 10^5 = 3,200,000 \end{aligned}$$

Then the number of permutations that need to be computed and accessed in the solution space will be as follows:

$$m! * (\prod_{i=1, 1} (deg[i]! - deg[i]) + \prod_{i=1, 2} (deg[i]! - deg[i]) + \dots \prod_{i=1, n} (deg[i]! - deg[i]))$$

Although we will prune a large number of branches during the execution of the

algorithm, the worst case time complexity is still exponential in the number of edges.

Hence this algorithm has exponential time complexity in the worst case.

Bounding Strategies

As the previous section described, there are two basic components in partial enumeration. One is a *search* algorithm to traverse the search tree space. The choices can be depth-first search or breadth-first search. The second component is *optimality* and *feasibility bounding* functions to prune the search tree. Optimality bounds evaluate the potential of a partial solution of leading to an optimum solution, while feasibility bounds determine if a partial solution satisfies all of the required properties of a solution. In the case of this algorithm of finding the minimum crossing number, we use a heuristic bound as the initial global bound, and if we get a better result at the end of one branch, we will use the new result as the new global bound to compare with nodes at each level of the search tree. Generally an ideal bounding function should be easily computable, but still smart enough to detect partial solutions which have no chance of leading to an optimum. So there is a tradeoff made between intelligence and speed. As a preliminary step to optimality bounding, a good initial global bound must be obtained before beginning the search process. This is usually acquired through some fast heuristic (e.g. greedy algorithm) for the problem. Hence, if a maximization problem is to be solved, the initial bound represents a lower bound to an optimum solution, and if a minimization problem is to be solved, the initial bound represents an upper bound. Naturally, in finding the

crossing number, what we need will be a good initial upper bound. In the next section we will discuss some heuristics for obtaining an initial upper bound.

Heuristic Bounds

A heuristic is a suboptimal yet fast strategy which yields solutions we are willing to accept when an optimal strategy is not feasible due to time limitations. In the implementation, we used Nicholson's heuristic [29] to come up with an approximate solution as the initial upper bound. We also tried a spanning tree heuristic, but it produced inferior initial bounds than Nicholson's. Nevertheless, we describe the spanning tree heuristic since it has the potential for obtaining a better heuristic solution.

A *subgraph* of a graph $G=(V(G), E(G))$ is a graph $H=(V(H), E(H))$ such that $V(H)$ is a subset of $V(G)$ and $E(H)$ is a subset of $E(G)$. If $V(H) = V(G)$, then H is a *spanning subgraph* of G . We next give a useful theorem.

Theorem 1.3 The following assertions are equivalent for a graph T :

- (i) T is a tree;
- (ii) Any two vertices of T are linked by a unique path in T ;
- (iii) T is minimally connected, i.e. T is connected but $T-e$ is disconnected for every edge e belonging to T ;
- (iv) T is maximally acyclic, i.e. T contains no cycle but $T + xy$ does, for any two non-adjacent vertices x, y belonging to T .

That every connected graph contains a spanning tree is a frequently used application of this theorem. The reason is very simple. By the equivalence of (i) and

(iii), any minimal connected spanning subgraph is a tree. Then we find a spanning tree by doing a DFS search. Using the same method described previously, we add edges one by one and compute how many cross points result. We may use this number as an initial upper bound for the branch-and-bound algorithm. Since a heuristic normally yields a suboptimal solution, a good heuristic is one that generally yields a solution as near to the optimal value as possible and which is relatively fast. In finding the minimum crossing number, the spanning tree heuristic turns out to be a poor heuristic, in general, although it has the potential for improvement.

It is a fact that the planar crossing number of a graph is less than or equal to the fixed linear crossing number of the graph. The next bound used is given by a heuristic presented by Nicholson [29]. In this method the network is expressed in the form of a permutation by deforming the network so that the node points lie on a straight line with the connections drawn as semicircles above and below the node line. Since the formulas used to calculate the number of crossings consist primarily of summations, the procedure is quickly performed on a computer, actually in polynomial time. The goal of this heuristic is to compute the positions of the vertices and the routing of the connections in the plane to avoid crossings as much as possible.

To lay out the graph by a computational process, the graph needs to be expressed in a suitable numerical form. In this algorithm the positions of the vertices and the routes of the connections constitute a permutation. So in the end the graph is drawn so that the vertices and edges possess the following characteristics:

- (a) The vertices all lie on a straight line called the *node line*.

(b) Each connection consists of a series of semicircles which lie on alternate sides of the node line. For convenience, the node line is horizontal and each semicircle lies completely above or below the node line.

An important theorem and its proof is given in [29] to support the algorithm.

Theorem 3.1 Any general network drawn in a plane with a minimum number of crossings can be redrawn as a second network with an equivalent crossings structure so that the second network possesses the two features mentioned above as (a) and (b).

The positions of the vertices can be simply denoted by their place in the sequence of vertices along the node line, and the routes of the connections can be recorded as a list of signs (+ or -) denoting whether the semicircle lies above or below the node line. In the algorithm, $p(j)$ denotes the vertex number occupying the j th position along the node line, so that the vertex arrangement is expressed by a permutation $[p]$ of size n , where n is the number of vertices. Also let there be m connections (edges) and denote by $q(i)$ the route of the i th connection, and $q(i) = 1$ for the semicircle above the node line and $q(i) = -1$ for the semicircle below the node line. Then the routing of the connections can be expressed by a permutation $[Q]$ of size m in which all the elements are +1 or -1. The graph layout can be represented by a joint permutation as follows:

$$[P, Q]: [p(1), p(2) \dots p(n); q(1), q(2) \dots q(m)]$$

Also, there is a way to specify how to evaluate the number of crossings which will occur for any given permutation $[P, Q]$. Let $\{c(i), c'(i)\}$ for $i = 1, m$ be the list of m connections where the connection $\{c(i), c'(i)\}$ joins the two node numbers $c(i)$ and $c'(i)$. It is also computationally useful to record the connection information in a connections matrix $C(i,$

j) of dimensions $n \times n$ which is determined as

$$C\{c(i), c'(i)\} = C\{c'(i), c(i)\} \equiv 1 \text{ for } i = 1, m$$

$$C(i, j) = 0 \text{ otherwise}$$

Then define upper and lower connections matrices separately as

$$\begin{aligned} A(i, j) &= 1, \text{ if } \{c(i), c'(i)\} \equiv \{p(i), p(j)\} \text{ and } q(i) = 1 \text{ for some } i \\ &= 0 \text{ otherwise} \end{aligned}$$

$$\begin{aligned} B(i, j) &= 1, \text{ if } \{c(i), c'(i)\} \equiv \{p(i), p(j)\} \text{ and } q(i) = -1 \text{ for some } i \\ &= 0 \text{ otherwise} \end{aligned}$$

Hence, the number of crossings associated with any permutation $[P, Q]$ is expressed as

$$\begin{aligned} F[P, Q] &= \sum_{i=1, n-3} \sum_{j=i+2, n-1} \{ A(i, j) \sum_{k=i+1, j-1} \sum_{l=j+1, n} A(k, l) + B(i, j) \\ &\quad \sum_{k=i+1, j-1} \sum_{l=j+1, n} B(k, l) \} \end{aligned}$$

Nicholson tested this heuristic on some complete graphs and bigraphs, and also on some randomly generated networks. The results show that the permutation procedure works well. The permutation procedure is computationally fast on graphs of up to approximately 20 nodes. We used this heuristic result as the initial upper bound, and it turned out to be very close to optimal for some simple graphs.

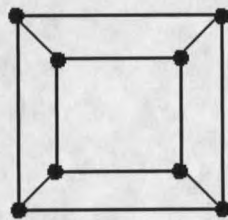
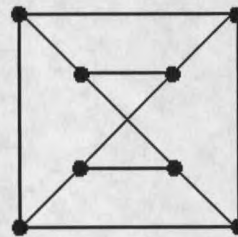
CHAPTER 3

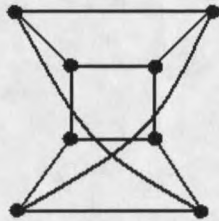
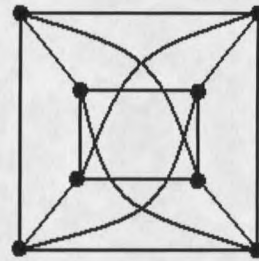
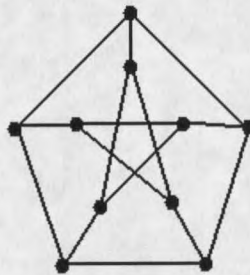
EXPERIMENTAL RESULTS

Description of the Test Graphs

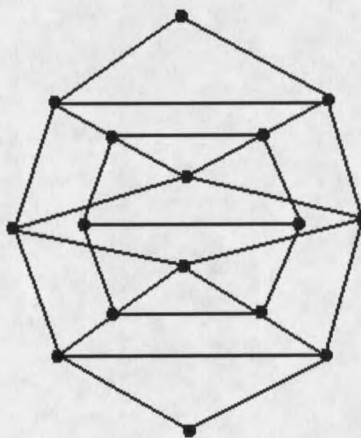
The algorithm was implemented in the C language and tested on three types of computers. One is a DEC AlphaServer 2100A 5/300 workstation with 300 MHz cpu speed and 512 megabytes of RAM. The other computer is a Silicon Graphics Origin2000 located at National Computational Science Alliance (NCSA). The Origin2000 is a cache coherent, non-uniform memory access supercomputer based on the MIPS R10000 processor, with 56 195MHz R10000 processors, 14Gb memory and 533Gb scratch disk space. And we also used a DELL PC.

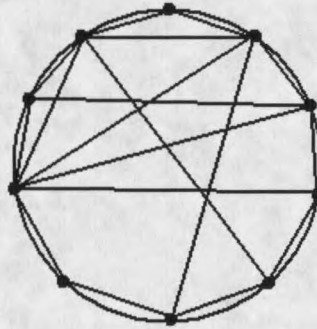
Some of the tested graphs are shown in Figure 7.

(a) Hypercube Q_3 (b) Twisted cube TQ_3

(c) Crossed cube CQ_3 (d) Folded cube FLQ_3 

(e) Petersen graph

(f) Undirected de Bruijn graph DB_4



(g) Random hamiltonian graph C10.2

Figure 7. Some tested graphs: (a) Hypercube Q_3 ; (b) Twisted cube TQ_3 ;
 (c) Crossed cube CQ_3 ; (d) Folded cube FLQ_3 ; (e) Petersen graph;
 (f) Undirected de Bruijn graph DB_4 ; (g) Random hamiltonian graph C10.2

Table of Results

Table 2. Results for Tested Graphs

Graph	n	m	# Crossings found	CPU time (Hours: Minutes: Seconds)	Computer
$K_{3,3} - e$	6	8	0	0:00:00	DEC Alpha
$K_5 - e$	5	9	0	0:00:00	DEC Alpha
$K_{3,3}$	6	9	1	0:00:10	DEC Alpha
K_5	5	10	1	0:00:15	DEC Alpha
$K_{4,3}$	7	12	2	1:30:00	DELL PC
Q_3	8	12	0	0:22:00	Origin2000
CQ_3	8	12	1	1:40:00	DEC Alpha
TQ_3	8	12	1	1:50:00	DEC Alpha
Petersen	10	15	2	3:30:00	Origin2000
K_6	6	15	3	35:00:00	DEC Alpha
$K_{3,5}$	8	15	4	40:00:00	DEC Alpha
FLQ_3	8	16	4	72:00:00	Origin2000
C10.2	10	18	2	120:00:00	Origin2000

CHAPTER 4

CONCLUSIONS

Limitations of the Algorithm

As mentioned in the section on time complexity analysis, the algorithm runs in exponential time, and can handle only relatively small graphs. Since the solution space is arranged according to edges, each time a new edge is added, the computation (permutation) requirements increase dramatically. That is why it takes almost no time in testing a graph with less than 12 edges on an ordinary PC, but it takes many CPU hours to run on a supercomputer for a graph with more than 20 edges.

This algorithm is not feasible for practical usage, such as for designing VLSI circuits including millions of nodes, but it serves as the basis for a refined algorithm for the crossing number problem. On a supercomputer, it may be used as a practical proof of crossing number for some small graphs with unknown crossing numbers, and also for verifying exact results for K_n and $K_{m,n}$ for smaller instances.

Further Research

We have presented a proposed algorithm for calculating the minimum crossing number of a graph. Several points of departure are suggested by this thesis for future

research. This could also be extended to finding the *maximum* crossing number by making minor modifications to the search algorithm. We could also find all possible drawings with the minimum crossing number and develop a software package to draw them on the screen in some systematic way, or we could remove the bound and enumerate all possible drawings of a graph.

Another approach would be to parallelize this process. This would greatly speed up the calculations since parallel DFS branch-and-bound algorithms can achieve super-linear speedup. For this sequential algorithm, a parallel version would be fairly straightforward. In order to obtain some initial results we can make a basic static partitioning of the search tree among the p processors on a parallel machine. This would range from looking at a best-first search with a dynamic queue as in [31], to looking at various graph-theoretic methods to prune the tree more effectively.

Finally, better bounding strategies and functions would undoubtedly improve the performance of the algorithm and enable the processing of larger graphs. Earlier pruning can lead to dramatic reductions in the number of search tree nodes explored by the algorithm. However, the scarcity of theoretical bounds for the crossing number of an arbitrary graph makes it difficult to come up with better bounding functions.

BIBLIOGRAPHY

- [1] Beineke, L.W. and R.D. Ringeisen. On the crossing numbers of products of cycles and graphs of order 4. *J. Graph Theory* 4 (1980) 145-155.
- [2] Bhatt, S.N., Leighton, F.T., A framework for solving VLSI, *Mathematical Systems Theory* 17, 1984, 47-70.
- [3] Bienstock, D., Some provably hard crossing number problems. *Disc. Comput. Geom.*, 6:443-459, 1991
- [4] Chartrand, G., and L. Lesniak. *Graphs and Digraphs*. Wadsworth & Brooks / Cole Advance Books & Software, Monterey, CA, 2nd. Edition, 1986.
- [5] Cimikowski, R. and P. Shope. A neural network algorithm for a graph layout problem. *IEEE Trans. Neural Networks* 7 (2) (1996) 341-345.
- [6] Cimikowski, R. Algorithms for the fixed linear crossing number problem, submitted for publication.
- [7] Damiani, E. and O. D'Antonia. An upper bound to the crossing number of the complete graph drawn on the pages of a book. *J. Comb., Info. & Sys. Sci.* 19 (1-2) (1994) 75-84.
- [8] Dean, A.M. and R.B Richter. The crossing number of $C_4 \times C_4$. *J. Graph Theory* 19 (1) (1995) 125-129.
- [9] Di Battista, G. and R. Tamassia. Automatic graph drawing and readability of diagrams. *IEEE Trans. Sys. Man Cybern.* SMC-18 (1) (1988) 61-79
- [10] Edmonds, J., A combinatorial representation for polyhedral surfaces. *Notices Amer. Math. Soc.*, 7:646, 1960
- [11] Eggleton, R.B. and R.K. Guy. The crossing number of the n-cube. *Amer. Math. Soc. Notices* 17 (1970) 757.

- [12] Erdos, P. and R.K. Guy. Crossing number problems. *Amer. Math. Monthly*, Jan., 1973, 52-58.
- [13] Garey, M. R., and Johnson, D. S., Crossing number is NP-complete, *SIAM J. Algebraic and Discrete Methods* 4 (1983), 312-316.
- [14] Gilbert R. S. and W. B. Kleinorder, "CNMgraf - Graphic presentation services for network management", *Proc. 9th Data Communications Symposium*, Whistler, B.C., Canada, September 1985, pp. 199-206.
- [15] Guy, R.K., The decline and fall of Zarankiewicz's theorem, in: *Proc. Proof Techniques in Graph Theory*, (F. Harary ed.), Academic Press, N.Y., 1969, 63-69.
- [16] Guy, R.K.. The decline and fall of Zarankiewicz's theorem, in F. Harary ed., *Proof Techniques in Graph Theory*, Academic Press, New York (1969), pp. 63-69.
- [17] Guy, R.K. Crossing numbers of graphs, in *Graph Theory and Applications*, Lecture Notes in Math. #303, Springer-Verlag, New York (1972).
- [18] Guy, R.P., Latest results on crossing numbers, in: *Proc. Recent Trends in Graph Theory*, Lecture Notes in Mathematics 186, Springer Verlag, Berlin, 1971, 143-156.
- [19] Guy, R.P., Crossing numbers of graphs, in: *Proc. Graph Theory and Applications*, Lecture Notes in mathematics 303, Springer Verlag, Berlin, 1972, 111-124.
- [20] Harary, F., *Graph Theory*, Addison-Wesley, Reading, 1969.
- [21] Harris, F.C., Jr., Harris, C.R., A proposed algorithm for calculating the minimum crossing number of a graph, in: *Proc. 8th Quadrennial Intl. Conference on Graph Theory, Combinatorics, Algorithms and Computing*, (Alavi, Y., Schwenk, A.J., Graham, R.I., eds.), 1996.
- [22] Kleitman D., The crossing number of $K_{5,n}$. *J. Comb. Theory* 9 (1970) 315-323
- [23] Klesc, M., R.B. Richter, and I. Stobert. The crossing number of $C_5 \times C_n$. *J. Graph Theory* 22 (3) (1996) 239-243.
- [24] Kumar, V., A.Grama, A.Gupta, and G.Karypis, *Introduction to Parallel Computing: Design and Analysis of Algorithms*, The Benjamin/Cummings Publishing Company, Inc., Redwood City, CA, 1994.
- [25] Lovegrove, C. and R.D. Ringeisen. *Crossing numbers of permutation graphs*. *Congressus Numerantium* 67 (1988) 125-135.

- [26] Madej, T. Bounds for the crossing number of the n -cube. *J. Graph Theory* **15** (1991) 81-97.
- [27] Marek-Sadowska, M., An Unconstrained Topological Via Minimization Problem for Two-Layer Routing, *IEEE Trans. Computer-Aided Design*, vol. CAD-3, pp. 184-190, 1984
- [28] Masuda, S., Nakajima, K., Kashiwabara, T., and T. Fujisawa. Crossing minimization in linear embeddings of graphs. *IEEE Trans. Comput.* 39 (1) (1990) 124-127.
- [29] Nicholson, T.A.J., Permutation procedure for minimizing the number of crossings in a network, *Proc. Inst. Elec. Engrs.* 115 (1968), 21-26.
- [30] Pach, J., Shahrokhi, F., and M. Szegedy. Applications of the crossing number. *Algorithmica* 16 (1) (1996) 111-117.
- [31] Quinn, M., and N. Deo, *An upper bound for the speedup of parallel best-bound branch-and-bound algorithms*, BIT 26 (1986), pp. 35-43.
- [32] Ringelsen, R.D. and L.W Beineke. The crossing number of $C_3 \times C_n$. *J. Comb. Theory B* 24 (1978) 134-136.
- [33] Shahrokhi, F., and Székely, L. A., Effective lower bounds for crossing number, bisection, width and balanced vertex separators in terms of symmetry, *Combinatorics, Probability, & Computing* 3 (1994), 523-543.
- [34] Shahrokhi, F., Sýkora, O., Székely, L.A., Vrto, I, Crossing numbers: bounds and applications, in: *Intuitive Geometry, Bolyai Studies*, Akadémia kiadó, Budapest, 31 p., 1997, 179-206.
- [35] Shahrokhi, F., Sykora, O., and L.A. Szekely. Bounds for the crossing number of a graph drawn on a compact 2-manifold. *Proc. WG'93 (19th. Int. Workshop on Graph-Theoretic Concepts in CS)*, June 1993, Utrecht, Netherlands.
- [36] Shahrokhi, F., Szekely, L.A., Sykora, O. and I. Vrto. Drawings of graphs on surfaces with few crossings. *Algorithmica* 16 (1) (1996) 118-131.
- [37] Sykora, O. and I. Vrto. On crossing numbers of hypercubes and cube-connected cycles. *BIT* 33 (1993) 232-237.
- [38] Youngs. J.W.T. Minimal embeddings and the genus of a graph. *J. Math. Mech.*, 12:303-315, 1963.

MONTANA STATE UNIVERSITY - BOZEMAN



3 1762 10338421 8