A Proposed Algorithm for Calculating the Minimum Crossing Number of a Graph

Frederick C. Harris, Jr.
Department of Computer Science

Cynthia R. Harris
Department of Mathematics

University of Nevada
Reno, NV 89557
fredh@cs.unr.edu

Abstract

In this paper we present a branch and bound algorithm for finding the crossing number of a graph. We begin with the vertex set and add edges. After adding each edge we use Edmonds' Rotational Embedding Scheme to determine if the resulting partial graph is planar. We continue until no more edges can be added and the partial graph remain planar. At this point we lay down the rest of the edges one at a time, adding vertices of degree 4 where crossings occur, thereby keeping the partial graph planar. Once all the edges are down, we begin backtracking to see if the graph can be drawn with fewer crossings by selecting other options when laying down edges.

1 Introduction

Determining the crossing number of a graph is an important problem with applications in areas such as circuit design and network configuration. It is this importance that has driven our work in finding the minimum crossing number of a graph.

Informally, the *crossing number* of a graph G, denoted $\nu(G)$, is the minimum number of crossings among all good drawings of G in the plane, 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

Although the problem is easily stated and has been well studied, not much is known about it. Erdös and Guy [5] put together a survey of what was known in 1973, and Turan discussed it via his Brick Factory Problem [13]. However, it was not until 1983 that Garey and Johnson [6] proved that finding the minimum crossing number of a graph was an NP-Complete problem. When this happened, work turned away from finding the minimum crossing number of a graph to other related problems. These have ranged from a return to the rectilinear crossing problem [8, 12] to the maximum crossing number of a graph or subgraph [7], to the thrackle conjecture [2].

We now wish to turn the attention of this paper back to the minimal crossing number of a graph. In Section 2 we review some notation and definitions we build on throughout the paper. In Section 3 we outline Edmonds' Rotational Embedding Scheme which is a vital part of our proposed algorithm. In Section 4 we discuss depth first search and branch and bound algorithms and in Section 5 present our proposed algorithm. Conclusions and future work are presented in Section 6.

2 Notation and Definitions

We now define some terms from topological graph theory which will be used through the rest of this paper. In general these definitions are as in [1].

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

An algebraic description of a 2-cell embedding was observed by Dyck [3] and Heffter[9]. This description is referred to as a Rotational Embedding Scheme which will be covered in Section 3

And finally, 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* [1]:

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

3 Rotational Embedding Scheme

With these definitions as a background, we now look at the Rotational Embedding Scheme, first formally introduced by Edmonds [4] in 1960 and then discussed in detail by Youngs [14] a few years later. The following is the formal statement of the Rotational Embedding Scheme as given in [1] on pages 130–131.

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

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

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:

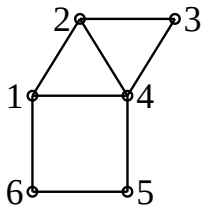

Figure 1: A Planar embedding of a graph

1) Start with edge (1,2)
2) Go to permutation π_2 and find out who follows 1, and it is 4. Therefore the second edge is (2,4).
3) Go to permutation π_4 and find out who follows 2, and it is 1. Therefore the third edge is (4,1).
4) Go to permutation π_1 and find out who follows 4 and it is 2. This yields edge (1,2) which was the original edge so we are finished.

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

The important thing to note at this point 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 and determine the genus of the surface.

One of the authors developed a computer program to generate all vertex permutation schemes for a graph [11]. The program then counts the regions of the resulting embedding and using Euler's formula determines if a given graph has a planar embedding. The code for this program can be found in [10], and we will make extensive use of it in Section 5.

4 Depth First Search Branch-and-Bound

If the solution space of a can be mapped to a tree, where each interior vertex is a partial solution, edges toward the leaves are options that refine the partial solution, and the leaves are complete solutions, then

there are various algorithms that can search the tree to find the optimal solution. A Depth First Search (DFS) algorithm is one such algorithm which, as its name implies, searches more deeply into the tree for a solution whenever possible. Once a path is found from the root to a leaf representing a solution, the search backtracks to explore the nearest unsearched portion of the tree. This continues until the entire tree has been traversed.

The Branch and Bound portion allows us to change one simple part of the DFS algorithm. When the cost to get to a vertex v exceeds the current optimal solution, we then tell the DFS algorithm not to traverse the subtree having v as its root.

This method exhaustively covers the entire search space even after finding an initial solution. However, it does not cover those sections of the search space that lead to solutions that are guaranteed to cost more than the current optimal solution. When the entire tree is covered the current optimal solution is the globally optimal solution.

5 Proposed Algorithm

For our proposed algorithm we are going to map the solution space from the crossing number problem onto a tree and search for the minimum crossing number with a branch and bound DFS. First we begin with the vertex set for the graph in question and begin to add edges. After each edge is added we determine whether the partial graph is still planar. This is done by using the Rotational Embedding Scheme code described in Section 3. Once we cannot add any edges and keep the partial graph planar we are ready to begin our mapping to the tree.

At this stage our partial graph is the root of our tree. The first option we have is the many different ways to draw this graph. The code described in Section 3 will return all possible planar embeddings of the partial graph. Therefore, the root of the tree has a branch for each possible embedding.

Now we select the first embedding and begin to build the rest of its tree. We do this by considering laying down the next edge (which will go from v_i to v_j). The first option we have is which one of the k regions that v_i is adjacent to should this edge to leave through. These regions represent the next layer of our tree. Once the region is selected, the next option is which of the l edges of that region the edge will cross.

When we have made this decision, we will create a cross vertex (degree 4) and place an edge from v_i to the cross vertex and then try to lay the edge from the cross vertex to v_j. This may be possible directly, or it may require more cross vertices.

For all the rest of the edges we lay them down in a similar manner, and when we have finally laid them all down we have reached a leaf in the tree. At this point we have a cost for the current solution which is the number of cross vertices we have added. This number of crossings becomes the new bound. The DFS continues by backing up and trying other decisions in the tree using the bound as a stopping criteria. We proceed until the entire solution space is traversed.

In order to understand this, let us walk through the algorithm with K_5 as our example. In Figure 2 we have the vertex set for the graph with all the edges added to the graph while it can still remain planar.

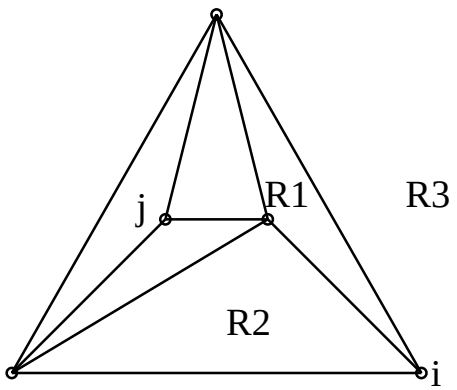

Figure 2: Planar portion of K_5

At this stage the algorithm states that we are to take all remaining edges and try to lay them down one at a time. This is fairly simple in this case since there is only one edge left to be added and that is from vertex i to vertex j. Now we have three choices, and these are the three regions that vertex i is adjacent to (R1, R2, and R3). We select R1 which has 3 edges and find that we cannot legally cross 2 of the edges (since they are incident with i), so there is only one choice. We then place a cross vertex on this edge and connect an edge from i to the cross vertex as shown in Figure 3. We then find out if we can draw the edge from the cross vertex to j and keep the graph planar. In this case

we can and we are finished with this edge and have one crossing. This solution is shown in Figure 4. The algorithm then backtracks and tries the other regions i is adjacent to and finds out that there are multiple ways to draw K_5 with one crossing, but none with zero crossings.

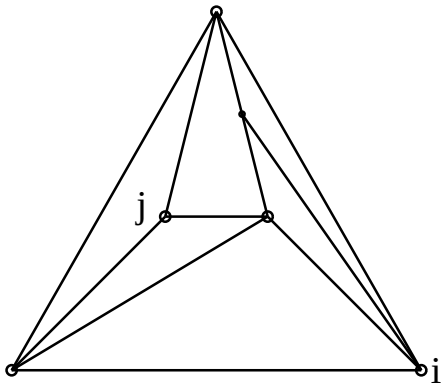

Figure 3: Beginning to lay down the alst edge for K_5

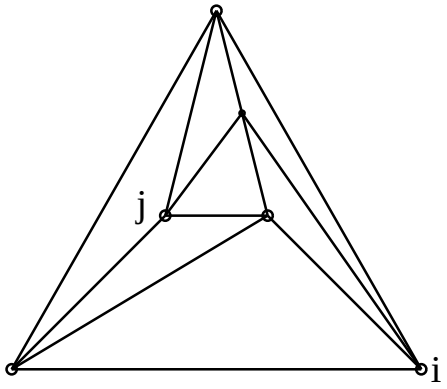

Figure 4: One drawing of K_5 with 1 crossing

6 Conclusions and Future Work

We have presented a proposed algorithm for calculating the minimal crossing number of a graph. This could also be extended to finding

the maximal crossing number by making minor modifications to the search algorithm.

The first step in the future work is obviously to implement this algorithm and test it on some known graphs. Once that is accomplished we would be able to determine if the order of the edges matter in reaching the correct answer. The order may be important for graphs in general, but it may not for certain classifications. Once that has been determined, we would like to calculate the crossing number for several families which are very important when looking at circuit design, such as K_n, $K_{(m,n)}$, and various others. The last item on the list (which would not be done last) would be to parallelize this process. This would greatly speed up the calculations since parallel branch and bound DFS algorithms can achieve super-linear speedup.

References

[1] G. Chartrand and L. Lesniak. *Graphs and Digraphs*. Wadsworth & Brooks/Cole Advanced Books & Software, Monterey, CA, 2nd. edition, 1986.

[2] J.E. Cottingham. *Thrackles, Surfaces, and Maximum Drawing of Graphs*. PhD thesis, Clemson, University, Clemson, SC 29634, August 1993.

[3] W. Dyck. Beiträge zur analysis situs. *Math. Ann.*, 32:457–512, 1888.

[4] J. Edmonds. A combinatorial representation for polyhedral surfaces. *Notices Amer. Math. Soc.*, 7:646, 1960.

[5] P. Erdös and R. Guy. Crossing numbers of graphs. In Y. Alavi, et al., editor, *Graph Theory and Applications*, pages 111–124. Springer-Verlag, New York, 1973.

[6] M.R. Garey and D.S. Johnson. Crossing number is NP-Complete. *SIAM J. of Alg. Disc. Meth.*, 4:312–316, 1983.

[7] Richard Guy, Heiko Harborth, Barry Piazza, Richard Ringeisen, and Sam Stueckle. Crossings in the complete graph. preprint.

[8] R.K. Guy. A minimal problem concerning complete plane graphs. In Y. Alavi, D.R. Lick, and A.T. White, editors, *Graph Theory and Applications*, pages 111–124, Berlin, 1972. Springer-Verlag.

[9] L. Heffter. Über das problem der nachbargebiete. *Math. Ann.*, 38:477–508, 1891.

[10] C. Lovegrove. Crossing numbers of permutation graphs. Master's thesis, Clemson University, Clemson, SC 29634, May 1988.

[11] C. Lovegrove and R.D. Ringeisen. Crossing numbers of permutation graphs. *Congr. Numer.*, 67:125–135, 1988.

[12] J.T. Thorpe and F.C. Harris, Jr. A parallel stochastic optimization algorithm for finding mappings of the rectilinear minimal crossing problem. *Ars Comb., to appear.*

[13] P. Turan. A note of welcome. *Journal of Graph Theory*, 1:7–9, 1977.

[14] J.W.T. Youngs. Minimal imbeddings and the genus of a graph. *J. Math. Mech.*, 12:303–315, 1963.