Parallel Computation of the Minimum Crossing Number of a Graph

Umid Tadjiev [*] Frederick C. Harris, Jr. [*]

Abstract

Determining the crossing number of a graph is an important problem with applications in areas such as circuit design and network configuration. In this paper we present the first parallel algorithm for solving this combinatorial optimization problem. This branch-and-bound algorithm, which adds and deletes crossings in an organized fashion, presents us with the opportunity to verify many conjectures which are decades old, as well as pursuing future work in efficient circuit design.

1 Introduction

Optimal circuit layout and network design are two problems that are becoming more important as the number of computers grows rapidly and their capabilities increase. Unfortunately the applications are obvious but the theory available to help solve the problem is quite deficient. Determining the crossing number of a graph is a problem whose solution could improve the state of the theory in this important application area. 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 [17]. 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, 16] to the maximum crossing number of a graph or subgraph [9], to the thrackle conjecture [2].

We now wish to turn the attention of this paper back to the minimum 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 we present the first sequential algorithm for the

[*]Department of Computer Science, University of Nevada, Reno NV 89557, tadjiev@cs.unr.edu, fredh@cs.unr.edu

problem. The parallel algorithm is discussed in Section 6 and Conclusions and Future Work are presented in Section 7.

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[11]. 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]:

Definition 2.1. *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 [18] a few years later. The following is the formal statement of the Rotational Embedding Scheme as given in [1] on pages 130–131.

Definition 3.1. *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:

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.

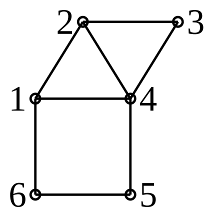

FIG. 1. *A Planar embedding of a graph*

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.

A computer program to generate all vertex permutation schemes for a graph was developed in [14]. This program 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 [13], and we will make extensive use of it in Sections 5 and 6.

4 Depth First Search with Branch-and-Bound

If the solution space of a problem 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 The First Sequential Algorithm

When we began studying this problem we were amazed that we could not find a single citation in the literature that discussed how to solve this problem algorithmically. Several heuristics and conjectures were found [5, 6, 8, 17], but there were no algorithms. Therefore, we began to work on a sequential algorithm for computing the minimum crossing number of a graph. This past year one of us presented a proposed algorithm for sequentially calculating the minimum crossing number of a graph [10].

In this algorithm we mapped the solution space of the crossing number problem onto a tree and then searched 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 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.

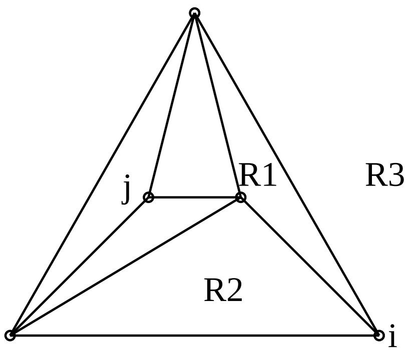

FIG. 2. *Planar portion of* K_5

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

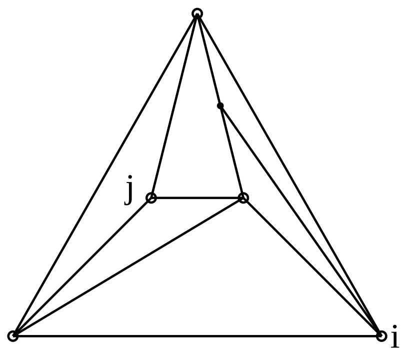

FIG. 3. *Beginning to lay down the alst edge for K_5*

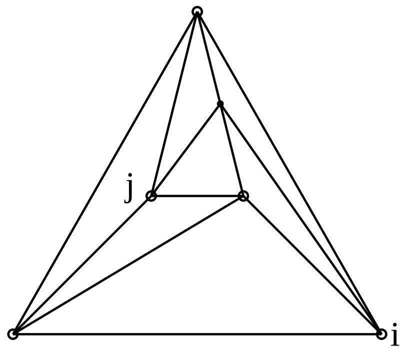

FIG. 4. *One drawing of K_5 with 1 crossing*

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.

6 A Parallel Algorithm and Implementation

The parallel algorithm for this preliminary work was very straightforward. In order to obtain some initial results we did a basic static partitioning of the search tree among the p processors in our parallel machine. This method, along with its benefits and drawbacks, is discussed in detail in [12]. We figured we would attempt this first and see if it was reasonable, and then modify it later to more dynamic methods after the feasibility was determined.

The implementation of this algorithm was developed on and run on a network of Pentium 133 machines running Linux. This network of machines was linked together into a parallel cluster with PVM [7] using a host-node programming style. This configuration was chosen for its ease of use and availability. In the future we wish to modify the implementation to have dynamic partitioning of the search space and target a new multiprocessor shared memory machine that has just become available.

For evaluation of this algorithm and its implementation we decided to use the family of complete graphs, denoted K_n, as our test cases. This family was selected for a few reasons. First it is one of the few families of graphs where there are some known answers since it is well-studied family. Secondly, it is one of the few families with a conjectured formula for the minimum crossing number. For complete graphs of size 10 and less, a simple formula provides the minimum crossing number [8]:

$$\nu(K_n) = \frac{\lfloor \frac{n}{2} \rfloor \lfloor \frac{n-1}{2} \rfloor \lfloor \frac{n-2}{2} \rfloor \lfloor \frac{n-3}{2} \rfloor}{4}$$

This formula, which is conjectured by Richard Guy to be the exact answer for all n [8], provided a ball park for an initial bounds when doing our branch-and-bound search. The results of the sequential and parallel versions are shown in Table 1. This basically shows that the computation times for K_6 and K_7 are so small that we do not have to worry about the parallel implementation; however, K_8 has a computation time that makes it very desirable to compute its minimum crossing number, and that of larger members of the family, in parallel.

TABLE 1
Computation time (in seconds) for K_6 - K_8 with p processors

	$p = 1$	$p = 2$	$p = 4$
K_6	0	0	1
K_7	8	4	3
K_8	12230	6251	3794

From this point on we dealt with K_8 almost exclusively. We will attack K_9 and others in this family when we have finished some optimization that we will discuss in Section 7. The numbers just presented for K_8 yield the speedup given in Figure 5 and an efficiency as given in Table 2.

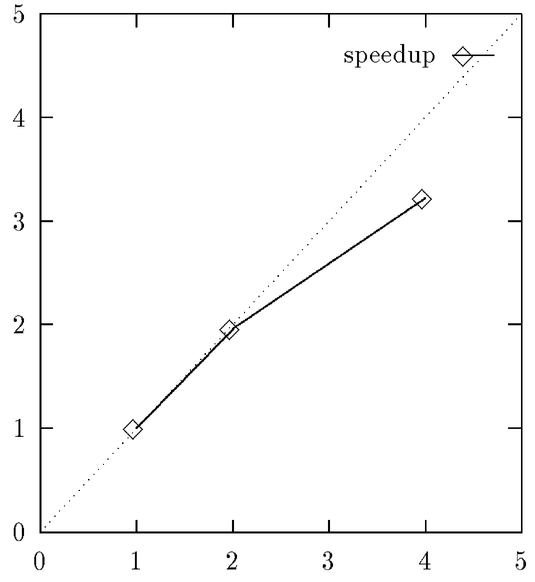

FIG. 5. *Speedup for K_8 as a function of p processors*

The interesting thing to note is the time for K_8 with four processors. For this computation two of the slaves finished in almost identical times of about 3050 seconds, while the other two were both around the final result of 3794 seconds. This detail points out one of the major problems with static partitioning in tree search algorithms, and that is lack of balanced workloads. For this reason we must consider dynamic partitioning of the workload.

TABLE 2
Efficiency for K_8 as a function of p processors

	$p = 1$	$p = 2$	$p = 4$
K_8	1.0	0.9782	0.8058

7 Conclusions and Future Work

We have presented a parallel algorithm for calculating the minimum crossing number of a graph. This has been built upon the proposed sequential algorithm we presented in [10]. We have implemented this algorithm, and shown its capabilites by comparing the sequential and the parallel versions on some of the initial members of the family of Complete Graphs (K_7 and K_8) that are not trivial.

We see this work continuing in various different ways. First we would like to construct a package that will be able to automatically draw the resulting graphs which we can compute. Secondly we would like to examine the computational method used in the sequential method and see if there are portions that can be improved, since this could improve the parallel version dramatically. This would range from looking at a best first search with a dynamic queue of work as Quinn and Deo presented in [15] to looking at various graph theoretic ways to legally prune the tree more effectively.

Then we would like to go back to calculate the crossing number for several families which are very important when looking at circuit design, such as the rest of K_n, $K_{(m,n)}$, and various others. We are hoping that at around K_{12} we will be able to show a counter-example (as we did with the rectilinear crossing problem [16]) to the conjecture proposed by Richard Guy [8] many years ago as the exact solution of this problem.

References

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

[2] J. 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** (1888), pp. 457–512.

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

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

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

[7] A. Geist, A. Beguelin, J. Dongarra, W. Jiang, R. Manchek, and V. Sunderam, *PVM: Parallel Virtual Machine – A User's guide and tutorial for networked parallel computing*, MIT Press, Cambridge, MA, 1994.

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

[9] R. Guy, H. Harborth, B. Piazza, R. Ringeisen, and S. Stueckle, *Crossings in the complete graph.* preprint.

[10] F. C. Harris, Jr. and C. R. Harris, *A proposed algorithm for calculating the minimum crossing number of a graph*, in Proceedings of the Eighth Quadrennial International Conference on Graph Theory, Combinatorics, Algorithms, and Applications, Y. Alavi, A. J. Schwenk, and R. L. Graham, eds., Kalamazoo, Michigan, June 1996, Western Michigan University.

[11] L. Heffter, *Über das problem der nachbargebiete*, Math. Ann., **38** (1891), pp. 477–508.

[12] V. Kumar, 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.

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

[14] C. Lovegrove and R. Ringeisen, *Crossing numbers of permutation graphs*, Congr. Numer., **67** (1988), pp. 125–135.

[15] M. Quinn and N. Deo, *An upper bound for the speedup of parallel best-bound branch-and-bound algorithms*, BIT, **26** (1986), pp. 35–43.

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

[17] P. Turan, *A note of welcome*, Journal of Graph Theory, **1** (1977), pp. 7–9.

[18] J. Youngs, *Minimal imbeddings and the genus of a graph*, J. Math. Mech., **12** (1963), pp. 303–315.