University of Nevada
Reno

Parallel Computation and Graphical Visualization of the Minimum Crossing Number of a Graph

A thesis submitted in partial fulfillment of the
requirements for the degree of Master of Science
with a major in Computer Science.

by

Umid Tadjiev

Dr. Frederick C. Harris, Jr., Thesis advisor

August 1998

The thesis of Umid Tadjiev is approved:

Thesis Advisor

Department Chair

Dean, Graduate School

University of Nevada

Reno

August 1998

Dedication

To my grandfather, Khasan Rasulev.

Acknowledgements

I would like to express deepest and sincerest gratitude to my advisor, Professor Harris, for all his help, guidance and encouragement throughout my academic career at University of Nevada, Reno in general and Master's program in particular.

I also wish to thank Professor Louis and Professor Pinsky for being on my committee and their valuable time.

A very special thank goes to my mother and brother, for their continuing support of my endeavors in the New World.

I would like to thank my high school Mathematics teacher, Felix Spektor, for showing that learning science can be fun.

I would also like to say thank you, to Marat Zhaksilikov, for his help on graph drawing part of the project.

Abstract

Finding the minimum crossing number of a graph is an interesting and challenging problem in graph theory and applied mathematics. Real world applications of this problem, such as circuit layout and network design, are becoming more and more important.

This thesis presents a parallel algorithm for finding the minimum crossing number of a graph, based on the first sequential algorithm presented in [20]. This parallel algorithm was tested on various architectures and a comparison of the corresponding results is given, including running time, efficiency, and speedup. Implementation of the algorithm gives us an ability to verify conjectures proposed for various families of graphs, as well as apply the algorithm to real world applications.

Another important aspect of the problem is ability to draw the solution on the $2-D$ plane. This thesis gives an overview of graph drawing algorithms, starting with the famous result by Fáry, and presents a new algorithm for drawing complete graphs, that uses the output of the previous algorithm.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Finding the minimum crossing number of a graph is one of the many interesting unsolved problems in graph theory. The problem is easily stated and the minimum crossing number has been found for some graphs, but the solution for a general case has not been found. It has been shown by Garey and Johnson that the problem is NP-complete and conjectures have been made concerning the minimum crossing number of some families of graphs. Despite being extensively studied by many mathematicians in the field of graph theory, no algorithm existed for solving the problem until 1996, when the first sequential algorithm was proposed in [20].

This thesis presents the results of the implementation of that algorithm, as well as the results of parallel implementation on different architectures and a comparison of those results.

Graph drawing is another equally important problem in graph theory. In 1948 Fáry proved in [12] that every planar graph can be drawn on the $2-D$ plane using only straight lines. A variety of graph drawing algorithms have been developed based on that famous result by Fáry. We present a new algorithm for drawing graphs that uses the output of our parallel algorithm.

The rest of this thesis is outlined as follows: Chapter 2 provides the background

information on the problem, as well as the problem statement. It also reports on the current state of affairs in this field of graph theory. In Chapter 3 we describe the first algorithm proposed to find the minimum crossing number of a graph. Chapter 4 presents the results of the sequential and various parallel implementations of this algorithm as well as their comparison and analysis.

Graph drawing algorithms based on the work by Fáry are presented in Chapter 5. Chapter 6 describes a new algorithm for drawing graphs. Conclusions and directions for Future Work are provided in Chapter 7.

Chapter 2

Current Results

Question about the crossing number of a graph represents one of the many important problems in graph theory and applied mathematics with numerous applications in areas of network design and circuit layout. Here is how Paul Turan described this problem when he first encountered it [37]:

> In July 1944 the danger of deportation was real in Budapest, and a reality outside Budapest. We worked near Budapest, in a brick factory. There were some kilns where the bricks were made and some open storage yards where the bricks were stored. All the kilns were connected by rail with all the storage yards. The bricks were carried on small wheeled trucks to the storage yards. All we had to do was to put the bricks on the trucks at the kilns, push the trucks to the storage yards, and unload them there. We had a reasonable piece rate for the trucks, and the work itself was not difficult; the trouble was only at the crossings. The trucks generally jumped the rails there, and the bricks fell out of them; in short this caused a lot of trouble and loss of time which was rather precious to all of us (for reasons not to be discussed here). We were all sweating and cursing at such occasions, I too; but *nolens-volens* the idea occurred to me that this

loss of time could have been minimized if the number of crossings of the rails had been minimized. But what is the minimum number of crossings?

2.1 Problem Statement

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

Garey and Johnson proved in [14], that this problem is NP-Complete. Erdös and Guy [11] compiled a survey of the problem and minimum crossing numbers have been found for some families of graphs and conjectures have been proposed for others.

2.2 Results for Selected Families of Graphs

Families of graphs for which the minimum crossing number has been found are products of C_n, such as C_m x C_l, and bipartite graphs. First, we will describe what those graphs are, but a few background definitions are needed in order to proceed.

Definition 2.2.1 *A walk of length k is a sequence $v_0, e_1, v_1, e_2, \ldots, e_k, v_k$ of vertices and edges such that $e_i = v_{i-1}v_i$ for all i.*

Definition 2.2.2 *A trail is a walk with no repeated edge.*

Definition 2.2.3 *A path is a walk with no repeated vertex.*

Definition 2.2.4 *A $u, v-walk$ is a walk with first vertex u and last vertex v; these are its endpoints, and it is closed if $u = v$.*

Definition 2.2.5 *A cycle is a closed trail of length at least one in which "first = last" is the only vertex repetition. Cycle of length n is denoted by C_n.*

Definition 2.2.6 *The Cartesian product of graphs G and H, written $G \times H$, is the graph with vertex set $V(G) \times V(H)$ specified by putting (u, v) adjacent to (u', v') if and only if*

(1) $u = u'$, and $vv' \in E(H)$

or

(2) $v = v'$, and $uu' \in E(G)$

For example, Fig 2.1 shows a product of C_3 and C_2.

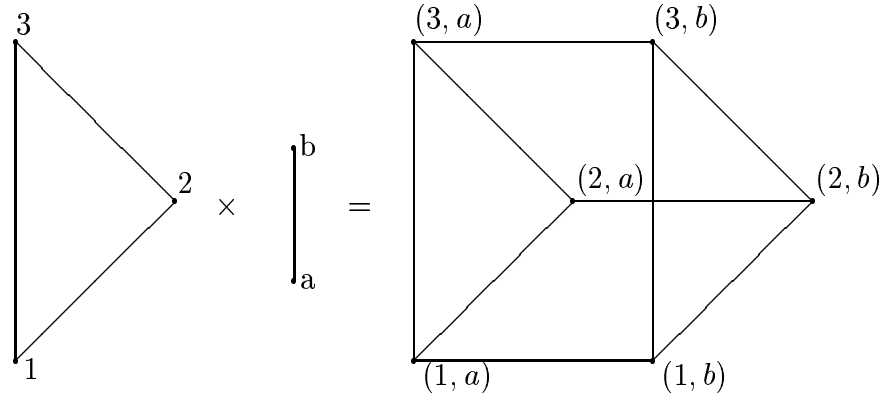

Figure 2.1: $C_3 \times C_2$

Crossing numbers have been found or conjectured for some families of graphs, namely complete graphs, bipartite complete graphs, and products of cycles. Guy and Jenkyns have found the bounds for the toroidal crossing number of $K_{m,n}$ in [17].

There they have shown that the crossing number of $K_{m,n}$ lies between

$$\frac{1}{15}\binom{m}{2}\binom{n}{2} \quad and \quad \frac{1}{6}\binom{m-1}{2}\binom{n-1}{2}$$

In 1971 Kleitman found the crossing number for $K_{5,n}$ on the plane in [23], and he also showed that for $1 \leq min(m,n) \leq 6$

$$\nu(K_{m,n}) = \lfloor\frac{m}{2}\rfloor\lfloor\frac{m-1}{2}\rfloor\lfloor\frac{n}{2}\rfloor\lfloor\frac{n-1}{2}\rfloor$$

Damiani et al. have determined an upper bound for the crossing number of the complete graph drawn on the pages of a book. A table giving this bound based upon the number of pages and the number of vertices is given on page 80 of [7]. The crossing number for $K_{3,n}$ has been found by Richter and Siráň in 1996 [30]. Ringeisen and Beineke have determined the crossing number for $C_3 \times C_n$ in [32] and they have also showed in [2] that

$\nu(C_4 \times C_n) = 2n$, for $n \geq 4$ and

$\nu(K_4 \times C_n) = 3n$, for $n \geq 3$.

Dean and Richter proved that crossing number of $C_4 \times C_4$ is 8 [8], and crossing number for $C_5 \times C_5$ has been shown to be 15 by Richter and Thomassen in [31]. Finally, Klešč et al. proved in [24] that crossing number for $C_5 \times C_n$ is $3n$.

2.3 Conjectures for Crossing Number Problem

Although results have been obtained for some graphs in families like K_n, $K_{m,n}$ and $C_n \times C_m$, a general solution for graphs in those families is still unknown. However, there are conjectured solutions for K_n, $K_{m,n}$ and $C_n \times C_m$.

Guy has conjectured in [16] that the minimum crossing number of K_n is equal

to:

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

It has been proven that the above mentioned formula is true for all K_n, where $n \leq 10$.

Zarankiewicz has conjectured in [41] that

$$\nu(K_{m,n}) = \lfloor \frac{m}{2} \rfloor \lfloor \frac{m-1}{2} \rfloor \lfloor \frac{n}{2} \rfloor \lfloor \frac{n-1}{2} \rfloor$$

Another conjecture was made by Harary et al. in [18] and it deals with products of cycles $C_n \times C_m$. It was conjectured that

$$\nu(C_n \times C_m) = n(m-2) \ \ for \ \ m \leq n$$

Guy has also conjectured the minimum crossing number of complete graphs in the case of rectilinear graphs [16], where all edges are assumed to be straight lines. However, counter examples were given by Thorpe and Harris in [36] which showed that the actual minimum crossing number for rectilinear complete graphs was lower than conjectured for graphs as small as K_{12} and K_{13}.

Chapter 3

Proposed Algorithm

In this chapter we describe an algorithm for finding the minimum crossing number of a graph. The algorithm was originally proposed in [20].

3.1 Definitions and Notation

In order to present this algorithm we must first define some terms from topological graph theory which will be used through the rest of this Chapter and Chapter 4. In general these definitions are as in [3].

Definition 3.1.1 *For our purposes a compact-orientable 2-manifold, or simply a surface, may be thought of as a sphere, or a sphere with handles.*

Definition 3.1.2 *The genus of the surface is the number of handles.*

Definition 3.1.3 *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.*

Definition 3.1.4 *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.*

Definition 3.1.5 *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 [9] and Heffter [21]. This description is referred to as a Rotational Embedding Scheme which will be covered in Section 3.2

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* [3]:

> 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 $p - q + r = 2 - 2n$.

3.2 Rotational Embedding Scheme

With these definitions as a background, we now look at the Rotational Embedding Scheme, first formally introduced by Edmonds [10] in 1960 and then discussed in detail by Youngs [40] a few years later. The following is the formal statement of the Rotational Embedding Scheme as given in [3] 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 $i = 1, 2, \ldots, p$, $\pi_i : V(i) \to V(i)$ 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 3.1 which gives a planar embedding of a graph. From this graph we obtain the following counterclockwise permutations associated with each vertex:

$$\pi_1 = (6, 4, 2) \quad \pi_2 = (1, 4, 3)$$
$$\pi_3 = (2, 4) \quad \pi_4 = (3, 2, 1, 5)$$
$$\pi_5 = (4, 6) \quad \pi_6 = (5, 1)$$

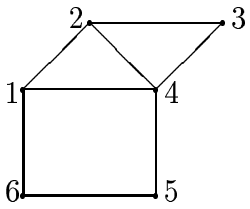

Figure 3.1: A Planar embedding of a graph

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) 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.

A computer program to generate all vertex permutation schemes for a graph was developed in [27]. 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 [26], and we will make extensive use of it in Sections 3.4 and 4.1.

3.3 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.

3.4 Description of the Algorithm

An amazing fact about this problem is that there was not a single citation in the literature that discussed how to solve this problem algorithmically. Several heuristics and conjectures were found [11, 14, 16, 37], but there were no algorithms. The first sequential algorithm for calculating the minimum crossing number of a graph was presented in [20].

In this algorithm the solution space from the crossing number problem is mapped onto a tree and then a DFS with branch-and-bound is employed to search for the minimum crossing number. The vertex set makes up the root of the tree and $|E|$ (the number of edges) branches are coming out of root. Each of the branches corresponds to the first edge that is added to the graph. On the next level of the tree there are $|E| - 1$ branches coming out of each node. There are $|E|$ levels in the tree, from top to bottom, and the path to each leaf corresponds to a unique permutation ordering of the edges.

First, edge (i, j) is selected and then Euler's formula is used to determine if the edge from vertex i to vertex j can be added without crossings (which is done by keeping track of the number of regions in the graph during construction). If the answer is positive, then the edge is added and next one is selected. Otherwise, an edge is selected (among the already added edges) which should be crossed (it may be the only edge crossed, or it may turn out to be the first one in a list of crossed edges). After choosing edge (k, l) to be crossed by edge (i, j), a new vertex m is created and edge (k, l) is removed. Next, partial edges (k, m), (m, l) and (i, m) are added and

edge (m, j) is drawn in the same way.

When drawing and edge, one should remember that no pair of adjacent edges cross, that two edges cross at most once and that an edge cannot cross itself. Another important point is to keep track of all partial edges and keep in mind that partial edges are actually *part* of an edge. In the case above, edge (i, m) is part of edge (i, j).

In order to understand the algorithm, let us walk through the procedure with K_5 as our example. There are 10 edges in K_5 and 9 of them can be added without creating any crossings (see Figure 3.2).

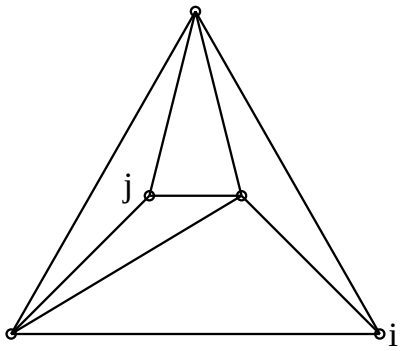

Figure 3.2: Planar portion of K_5 (first 9 edges)

When we try to add the last edge, Euler's formula shows that the resulting graph is not planar, which means that when edge (i, j) is added then at least one crossing is created.

The next step is to select edge (k, l) as the edge to cross, insert vertex m, remove edge (k, l) and, finally, to add partial edges (k, m), (m, l), and (i, m). The result is shown in Figure 3.3.

After drawing these partial edges, the algorithm tries to lay down the rest of edge (i, j) and it does so without any more crossings, as shown in Figure 3.4

The algorithm then backtracks, knowing that the new bound for number of crossings is one. It will then back up through the rest of the tree and try other branches

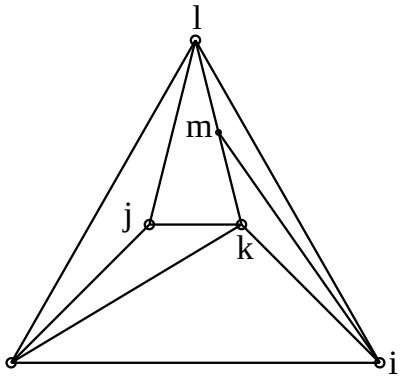

Figure 3.3: Beginning to lay down the last edge for K_5

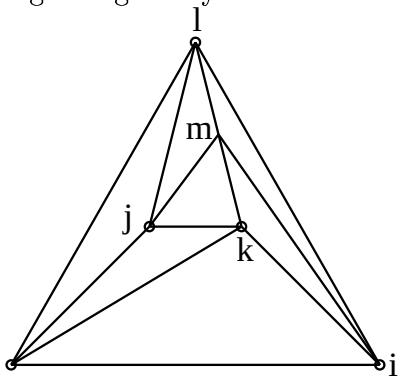

Figure 3.4: Drawing of K_5 with 1 crossing

only to find out that there are multiple ways to draw K_5 with one crossing, but none with zero crossings.

Chapter 4

Sequential and Parallel Computation Results

4.1 Parallelization of the Original Algorithm

The parallel algorithm for this work was very straightforward. In order to obtain some initial results we performed 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 [25].

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 [15] using a host-node programming style. This configuration was chosen for its ease of use and availability, but in the future we wish to modify the implementation to have dynamic partitioning of the search space.

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 [16]:

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

This formula, which is conjectured by Richard Guy to be the exact answer for all n, provided a ball park for an initial bounds when doing our branch-and-bound search.

4.2 Results

The results of the sequential and parallel versions are as in [34] and they are are shown in Table 4.1. 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 4.1: Computation time (in seconds) for K_6 - K_8 with p processors (PVM)

	$p = 1$	$p = 2$	$p = 4$
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 issues that we will discuss in Chapter 7. The numbers just presented for K_8 yield the speedup given in Figure 4.1 and an efficiency as given in Table 4.2.

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

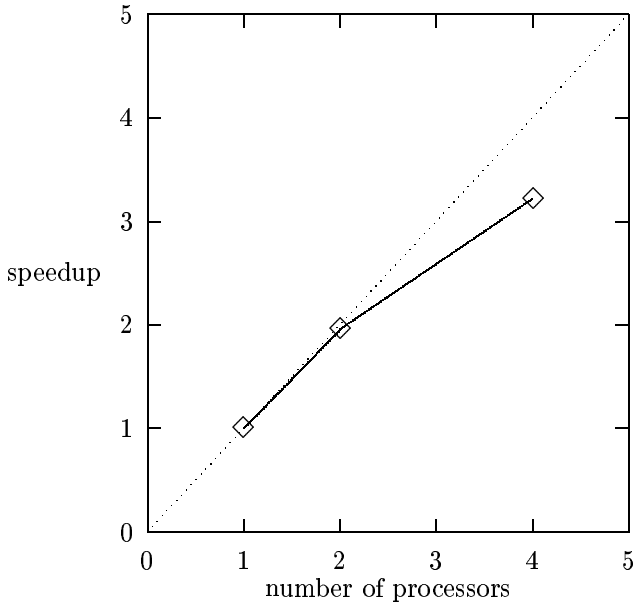

Figure 4.1: Speedup for K_8 as a function of p processors (PVM)

Table 4.2: Efficiency for K_8 as a function of p processors (PVM)

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

points out one of the major problems with static partitioning in tree search algorithms, and that is lack of a balanced workload. For this reason we must consider dynamic partitioning of the workload in the future.

It was interesting for us to see how two different message-passing interfaces, PVM and MPI will compare to each other. Therefore, the code was ported to LAM/MPI on the same network of workstations, with the only difference being that the operating system had been changed from LINUX to SOLARIS. Results, analogous to those given in Table 4.1, are shown in Table 4.3. Efficiency achieved for K_8 with MPI

implementation is shown in Table 4.4.

Table 4.3: Computation time (in seconds) for K_6 - K_8 with p processors (MPI)

	$p=1$	$p=2$	$p=4$
K_6	0	3	3
K_7	8	5	3
K_8	12230	6553	3897

Table 4.4: Efficiency for K_8 as a function of p processors (MPI)

	$p=1$	$p=2$	$p=4$
K_8	1.0	0.9332	0.7846

In the message-passing environment, like MPI or PVM, the major drawback is communication overhead. Therefore, it was interesting to rewrite the implementation, which was intended for network of workstations, for a shared memory multiprocessor. The performance increase was expected because communication overhead was eliminated and more powerful computational resources were used (SGI Power Challenge vs. network of P5 133). The structure of the original algorithm was left unchanged. The message passing was replaced by the introduction of the shared memory. The completely asynchronous nature of the original algorithm eliminated the need for barrier synchronization. Results for shared memory implementation are shown in Table 4.5 and corresponding efficiency is shown in Table 4.6.

Comparison of the speedup for different implementations is shown in Figure 4.2.

Table 4.5: Computation time (in seconds) for K_6 - K_8 with p processors (Shared Memory)

	$p = 1$	$p = 2$	$p = 4$
K_6	0	0	0
K_7	4	2	1
K_8	5823	2977	1771

Table 4.6: Efficiency for K_8 as a function of p processors (Shared Memory)

	$p = 1$	$p = 2$	$p = 4$
K_8	1.0	0.9779	0.8220

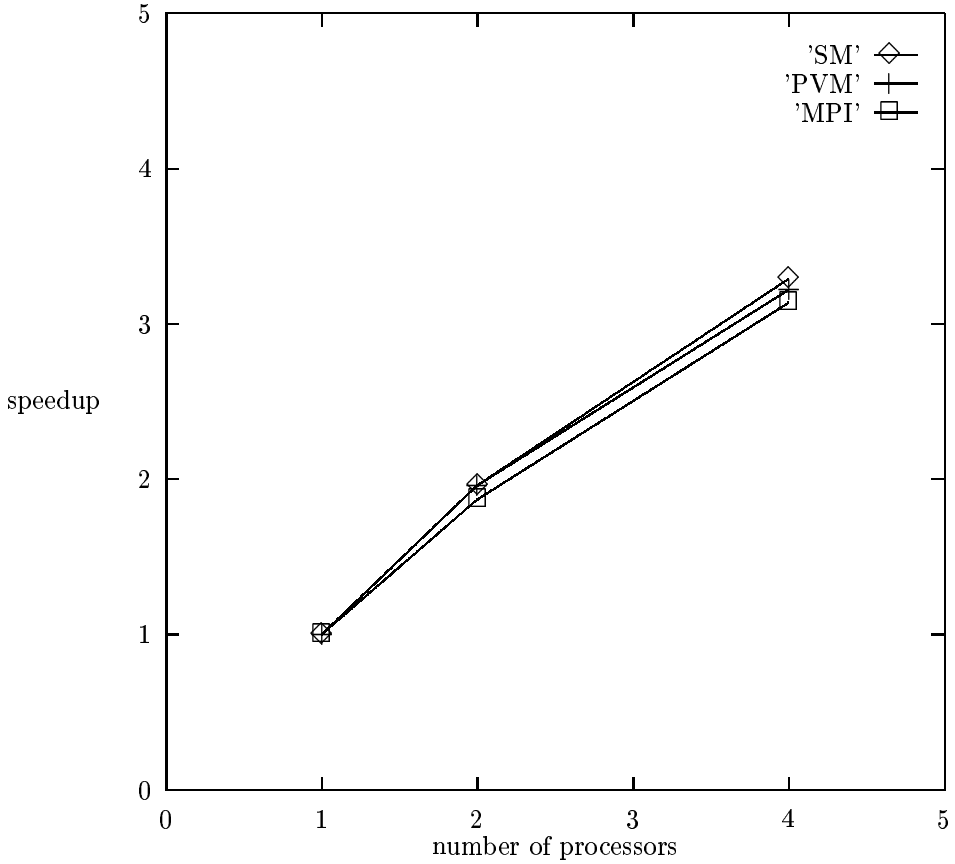

Figure 4.2: Comparison of speedup for K_8 (PVM, MPI, Shared Memory)

Chapter 5

Drawing Planar Graphs

5.1 Overview of the Previous Work and Result by Fáry

Finding the minimum crossing number of a graph is important, but a graph-theoretical representation of the solution is hard to visualize and to comprehend. Also, the process of converting the graph-theoretical representation of the solution into the visual one can be very time consuming, especially as number of vertices increases. This leads us to a problem of drawing graphs on a 2-D plane. A lot of research has been done in this area and number of algorithms for drawing a general graph on 2-D plane have been proposed. One of the first graph drawing algorithms was proposed by Tutte in 1963 [38]. [29] proposed an algorithm for drawing a planar graph when the cyclic order of the edges at each vertex is known. [35] is a good source of papers on this subject, and [1] gives an annotated bibliography of graph drawing algorithms.

The foundation of the algorithms for drawing planar graphs on the 2-D plane was laid down by the work of Fáry [12] in 1948. In this seminal work he proved that every planar graph can be drawn on 2-D plane using only straight lines. [5] offered an algorithm that would draw planar graphs nicely and [4] described a linear time

algorithm for convex drawing of planar graphs.

We will describe algorithms that are based on Fáry's result, namely those by DeFraysseix, Pach, and Pollack[13], and by Chrobak and Payne[6].

First, let's give some definitions that we will be using throughout this chapter.

Definition 5.1.1 *The Manhattan distance between* (x, y) *and* (x', y') *is*

$$| x - x' | + | y - y' |.$$

Definition 5.1.2 *An embedding of a planar graph is called Fáry embedding if every edge is represented by a straight line.*

Definition 5.1.3 *A planar graph is called a maximal planar graph if it can not have any more edges added without destroying its planarity.*

Fáry's embedding is constructed by using *canonical representation* of planar graph, this representation is defined in a lemma given in [13] (see [13] for lemma's proof), and that definition is given next.

Definition 5.1.4 *Let* G *be a maximal planar graph embedded in the plane with exterior face* u, v, w. *Then there exists a labeling of the vertices* $v_1 = u$, $v_2 = v$, $\ldots$, $v_n = w$ *(such a labeling of vertices is called a canonical representation of planar graph) meeting the following requirements for every* $4 \leq k \leq n$:

(i) The subgraph $G_{k-1} \subseteq G$ *induced by* v_1, v_2, $\ldots$, v_{k-1} *is 2-connected, and the boundary of its exterior face is a cycle* C_{k-1} *containing the edge* uv;

(ii) v_k *is in the exterior face of* G_{k-1}, *and its neighbors in* G_{k-1} *form an (at least 2-element) subinterval of the path* $C_{k-1} - uv$. *(see Figure 5.1)*

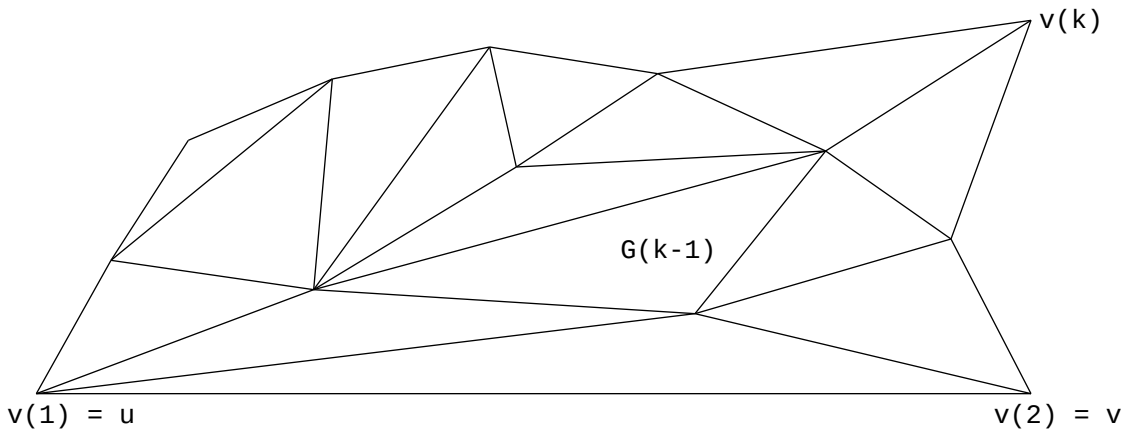

Figure 5.1: Canonical ordering

5.2 Algorithm by DeFraysseix, Pach, and Pollack

This algorithm constructs a Fáry embedding of a planar graph with n vertices on $2n-4$ by $n-2$ grid. The algorithm works in a following manner: we are given a planar maximal graph G with exterior face u, v, w and canonical labeling of its vertices $v_1 = u, v_2 = v, ..., v_n = w$. Suppose that during the kth step of the algorithm, subgraph G_k already has a Fáry embedding on a grid with following properties:

1) v_1 is at $(0,0)$, v_2 is at $(2k-4, 0)$

2) Let $v_1 = w_1, w_2, \ldots, w_m = v_2$ denote the vertices of the exterior face of G_k and $x(w_i)$ their respective x-coordinates. Then

   $x(w_1) < x(w_2) < \ldots < x(w_m)$

3) Slopes of the edges $[w_i, w_{i+1}], 1 \leq j < m$, are either $+1$ or -1.

Condition (3) implies that the intersection point of the line passing through w_i with slope $+1$ and the line passing through $w_j (i < j)$ with slope -1 is a lattice point $P(w_i, w_j)$. If we denote the neighbors of v_{k+1} in G_{k+1} (they form an interval on the boundary of G_{k+1}, see Lemma) as $w_p, w_{p+1}, \ldots, w_q$ then we can consider placing v_{k+1} at point $P(w_p, w_q)$. Our only concern with the choice of $P(w_p, w_q)$ will be that $P(w_p, w_q)$ may be on the same line as w_p, making it impossible to lay down a straight-line edge from w_p to $P(w_p, w_q)$ (see Figure 5.2).

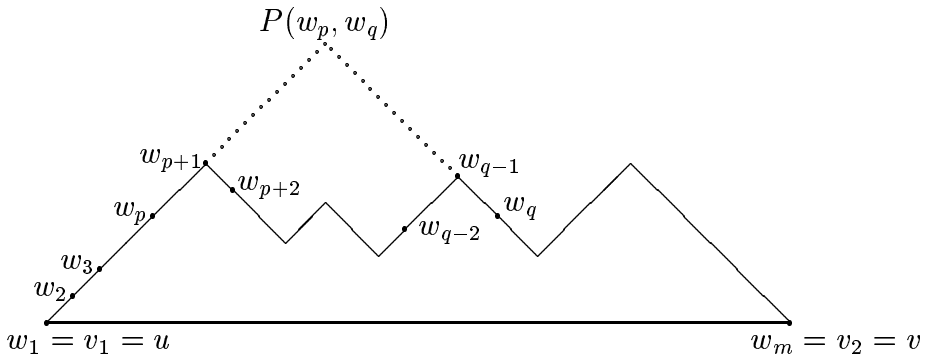

Figure 5.2: Placing v_{k+1}

In order to avoid this problem we should deform the existing Fáry embedding of G_k by moving some vertices one unit to the right, but we should ensure that the deformed embedding is still a Fáry embedding. This means that we should define a set of vertices to be moved and then move them in some consistent manner. One way to accomplish that goal is to define a set of vertices to be moved through recursion. Assume that for every vertex w_i on the exterior face of G_k a subset $M(k, w_i) \subseteq V(G_k)$ has been defined which confirms to the following conditions:

a) $w_j \in M(k, w_i)$, iff $j \geq i$

b) $M(k, w_1) \supset M(k, w_2) \supset \ldots \supset M(k, w_m)$

c) For any set of nonnegative numbers $(\alpha_1, \alpha_2, \ldots, \alpha_m)$ sequential translation of all vertices in $M(k, w_i)$ by α_i to the right $(1 \leq i \leq m)$ does not result in G_k losing its property of being a Fáry embedding (some vertices will be moved several times).

For example, for $k = 3$ given conditions can be satisfied by the following Fáry embedding:

$v_1 = (0, 0), v_2 = (2, 0), v_3 = (1, 1)$ and

$M(3, v_1) = \{v_1, v_2, v_3\}$, $M(3, v_2) = \{v_2, v_3\}$, $M(3, v_3) = \{v_3\}$

Applying condition c) with $\alpha_{p+1} = \alpha_q = 1$, $\alpha_i = 0, i \neq p + 1, i \neq q$ will give us a new Fáry embedding of G_k if we place v_{k+1} at $P(w_p, w_q)$. The vertices of the exterior face of G_{k+1} will be:

$u = w_1, w_2, \ldots, w_p, v_{k+1}, w_q, \ldots, w_m = v$

and sets M will be defined as:

$M(k + 1, w_i) = M(k, w_i) \cup \{v_{k+1}\}, i \leq p$

$M(k + 1, v_{k+1}) = M(k, w_{p+1}) \cup \{v_{k+1}\}$,

$M(k + 1, w_j) = M(k, w_j), j \geq q$

(the resulting embedding is still a Fáry embedding, see [13] for proof). But how do we define sets M? Sets M are defined through a sequence of inductive permutations. First, we define $\pi_1 = (1, 2)$. Now, suppose π_k is defined and vertex v_{k+1} is adjacent to the vertices $v_{i_1}, v_{i_2}, \ldots, v_{i_j}$ in G_k (in counterclockwise order). Then, π_{k+1} is generated

by inserting $k+1$ just to the left of i_2 and $n+k+1$ just to the left of i_j in permutation π_k. If we identify vertex v_j with the index j, then

$M(k, v_i) = \{j | j \leq n, j \text{ does not precede } i \text{ in } \pi_k\} =$

$= \{j | i \leq j \leq k, j \text{ does not precede } i \text{ in } \pi_k\}$

This step is the most expensive one in the algorithm, in terms of time, and it takes $O(n\ log(n))$ to execute (see [13]).

We assumed that a canonical ordering of the graph and its planar embedding are available, when algorithm starts processing the graph. A planar embedding can be obtained using a variety of algorithms, including the one described in the previous chapters, which will give all planar embeddings of a graph, or an algorithm by Hopcroft and Tarjan [22], which will give an embedding of a graph. The canonical ordering of the graph can be obtained using the following procedure:

- Pick two vertices v_1 and v_2, and label the rest of the vertices with -1.
- Process vertex v_k by visiting all of its neighbors and changing their labels. Suppose v is a neighbor of v_k. Then there are three possibilities:
  - v has a label of -1, and it is changed to 0
  - v has a label of 0, which means that one of its neighbors, call it u, has already been visited. If v_k is adjacent to u in the circular ordering of neighbor vertices around v, change v's label to 1. Otherwise make it 2.
  - v has a label of $j > 0$. If two vertices adjacent to v_k in the circular ordering of neighbor vertices around v have been already processed, change v's label to $j - 1$. If only one has been processed, v's label remains j. If none has been processed, change v's label to $j + 1$.

- After v_k has been processed, any vertex with label 1 can be chosen to be v_{k+1} in the canonical ordering. This process continues until no such v_{k+1} can be found or all vertices have been processed. The existence of the canonical ordering guarantees that all vertices will be processed.

This algorithm constructs Fáry embedding of a graph with n vertices on a $(2n - 4) \times (n - 2)$ grid in $O(n\ log(n))$ time.

5.3 Algorithm by Chrobak and Payne and other algorithms

The algorithm by Chrobak and Payne [6] is also based on Fáry's result and it represents a refinement of the algorithm by DeFraysseix, Pach and Pollack, since it reduces the running time to $O(n)$. The idea behind this algorithm is the same as in algorithm by DeFraysseix, Pach, and Pollack, but appropriate distribution of the information in the vertices of the graph yields linear-time complexity.

As in the algorithm by DeFraysseix, Pach, and Pollack a planar embedding and canonical ordering of a graph are needed. Also, $w_1, w_2, \ldots, w_m$ again denotes the cycle C_{k-1}, which is an exterior face of G_{k-1}. If $w_p, \ldots, w_q$ are neighbors of v_k in C_{k-1}, then we say that v_k *covers* $w_{p+1}, \ldots, w_{q-1}$.

G_k is viewed as a forest consisting of trees $L(w_1)$, ..., $L(w_m)$ whose roots are members of $C_k = \{w_1, \ldots, w_m\}$, where $L(v_i)$ is defined as a set of vertices that have to be moved every time v_i's position is adjusted (so sets L are similar to sets M in the previous algorithm). The children of a node are the vertices that it covers. A forest is represented as a binary tree T and left T-child of a node is defined as its leftmost child, and right T-child of a node is its next child to the right. The root of T is v_1, and $C_k = \{w_1, \ldots, w_m\}$ consists of v_1, its right T-child, its right T-child's right

T-child, etc. $L(w_i)$ consists of w_i and its left T-subtree. So, a T-subtree having w_i as its root is $\cup_{j \geq i} L(w_j)$.

When embedding v_k we do not need to know the exact coordinates of w_p and w_q. All that is needed by the algorithm is their y-coordinates and their relative x-coordinates $(x(w_p) - x(w_q))$, which is enough to compute $y(v_k)$, and the relative x-coordinate of v_k, $x(v_k) - x(w_p)$.

Definition 5.3.1 *For every vertex $v \neq v_1$, the x-offset of v is*

$$\Delta x(v) = x(v) - x(w)$$

where w is T-parent of v.

Definition 5.3.2 *If w is an ancestor of v, the x-offset between w and v is*

$$\Delta x(w, v) = x(v) - x(w)$$

In this algorithm the following information is stored for each vertex v:

$$\begin{aligned} Left(v) &= \textit{left T-child of } v \\ Right(v) &= \textit{right T-child of } v \\ \Delta x(v) &= \textit{the x-offset of } v \textit{ from its T-parent} \\ y(v) &= \textit{the y-coordinate of } v \end{aligned}$$

The algorithm has two phases. During the first phase vertices are added to the graph one by one and x-offsets and y-coordinates are computed. During the second phase the whole tree is traversed and x-offsets are accumulated to get the x-coordinates. The pseudocode of the algorithm, as well as proof of its correctness and linear running time is given in [6]. A drawing produced by Chrobak and Payne's algorithm is shown in Figure 5.3.

Harel and Sardas [19] have further refined the algorithm by Chrobak and Payne to produce more aesthetically pleasing drawings of graphs, with the same grid size and running time (see Figure 5.4 for a drawing of the same graph as in Figure 5.3). Schnyder [33] has developed a different algorithm that runs in a linear time and embeds a graph with n vertices onto a grid of size $(n-2) \times (n-2)$.

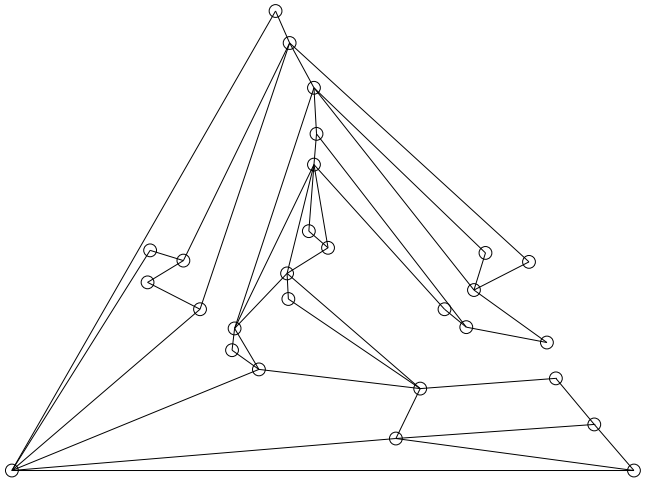

Figure 5.3: Drawing produced by Chrobak and Payne's algorithm

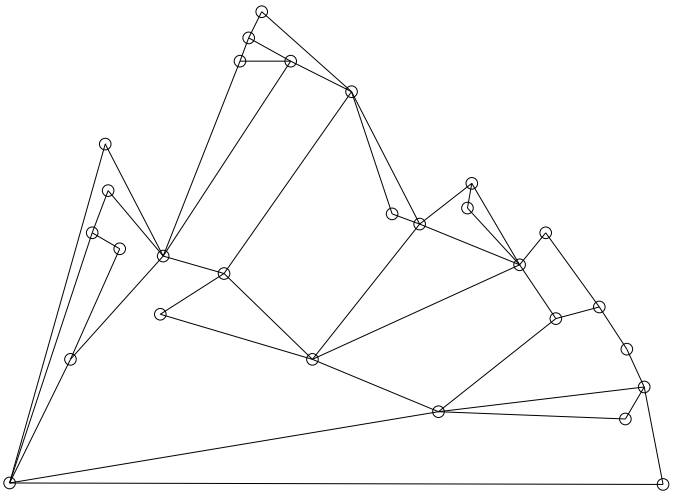

Figure 5.4: Drawing produced by Harel and Sardas' algorithm

Chapter 6

A New Algorithm for Drawing Complete Graphs

As one can see, the drawings produced by the algorithms described in Chapter 5 are not very "nice" looking. That is the reason why we decided to come up with a new algorithm that would produce nicer looking drawings, and also take the burden of converting the graph-theoretical representation into a graphical one off our shoulders.

6.1 Description of the Algorithm

Typical output produced by the algorithm described in Chapter 3 is shown in Figure 6.1. As one can see, translating that output into a drawing is a tedious process, that gets even more tedious when number of vertices increases further. To make most of that pain go away, we designed an algorithm, which produces a graph drawing by analyzing the output given in Figure 6.1.

In order to convert this graph-theoretical representation into a graphical one this new algorithm does the following:

(1) Read in the output of the algorithm, described in the previous chapters and find 2 vertices that are not connected with each other (we will call them center1 and center2).

```
Regions of the optimal solution are:
9-1-6
2-9-6
4-9-2
4-1-9
3-1-4
7-1-3
5-7-3
6-5-2
6-7-5
6-1-7
8-2-5
3-8-5
4-8-3
4-2-8

Number of crossings -- 3
Search for the optimal solution took 0 sec.
```

Figure 6.1: Output produced for K_6

(2) Out of all the regions, that do not contain center1 and center2, pick one with the lowest sum of orders of vertices (call it the outer triangle/region, although it is not necessarily a triangle).

(3) Inside the outer triangle there are 2 cycles, that are defined as cycles that contain all neighbors of a center, but not all elements of a cycle are necessarily connected with the center. The upper cycle is the one that contains zero or one vertex from the outer triangle. If the upper cycle includes one vertex from the outer triangle then that vertex will be the upper point of the outer triangle. If not, i.e. there are zero vertices from the outer triangle in the upper cycle, then we have to find 2 vertices of the outer triangle that are contained in the lower cycle. Those 2 vertices of the outer triangle will form the foundation of the outer triangle, with the remaining vertex being the upper point.

(4) Draw the outer triangle, since we know which point is on the top (upper point), and which ones are in the foundation. Moreover, we can find out which point is on the right and which one is on the left from the region information.

(5) Draw 2 centers inside the outer triangle in the following manner: find the distance between the outer triangle's foundation and the top (denote it d), then put the upper center on a line passing through the top and orthogonal to the foundation $d/4$ units below the top point; put the lower center $d/8$ units above the foundation. In the example, given in Figure 6.2, vertices 1 and 2 are the centers and the outer triangle is formed by vertices $5, 6$, and 7.

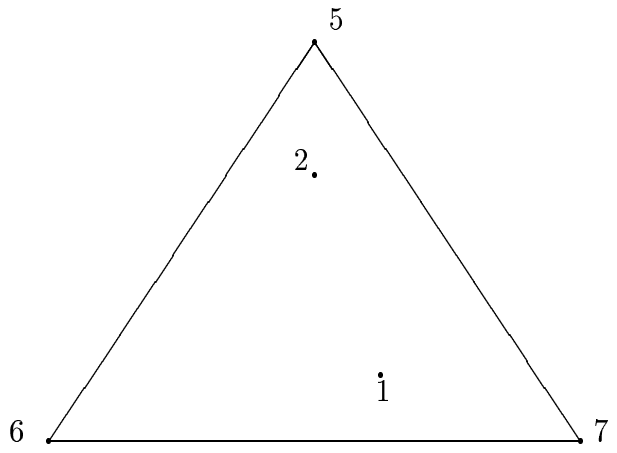

Figure 6.2: Drawing outer triangle and centers

(6) In the lower cycle find the vertex that precedes the right vertex of the outer triangle and draw it inside the outer triangle, close to the edge connecting the right and the top vertices and making sure that the lower center is inside the newly created triangle.

(7) Draw the lower cycle by putting all of its remaining vertices on the edge connecting the vertex added in the previous step, and the left vertex of the outer triangle.

(8) Connect vertices of the lower cycle with the lower center (those that should be connected, see Figure 6.3).

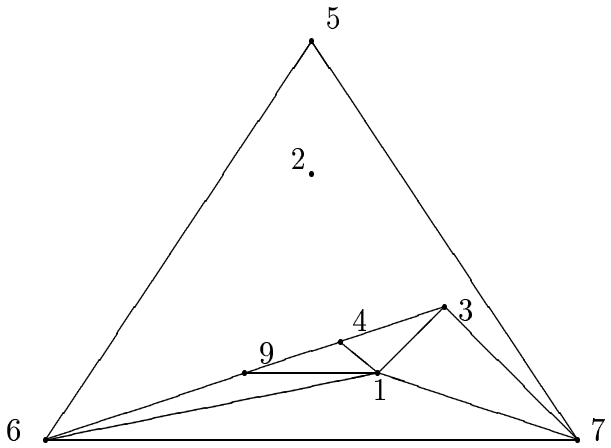

Figure 6.3: Drawing lower cycle

(9) Draw the upper cycle, with its upper point being a vertex that has the fewest number of connections with the members of the lower cycle or the one that is the upper point of the outer triangle. The upper cycle should also form a triangle, like the lower cycle. We have already defined what the upper point of that triangle should be. The foundation of that triangle is formed by the vertices that have the most number of connections with the lower cycle (see Figure 6.4).

(10) Find the vertex with the lowest index and all of its neighbors already placed. Draw that vertex with its coordinates being the average of the coordinates of its neighbors (if there is no such vertex then draw the remaining edges). Find

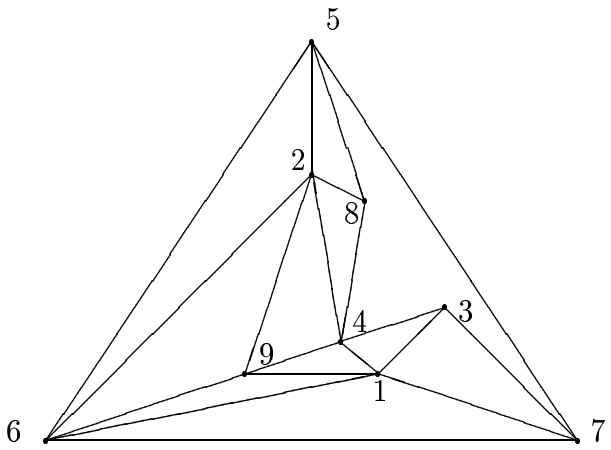

Figure 6.4: Drawing upper cycle

the next such vertex and draw it using the same procedure, etc.

(11) If we missed some vertices in the previous step, because their neighbors were not drawn, then repeat the previous step (see Figure 6.5).

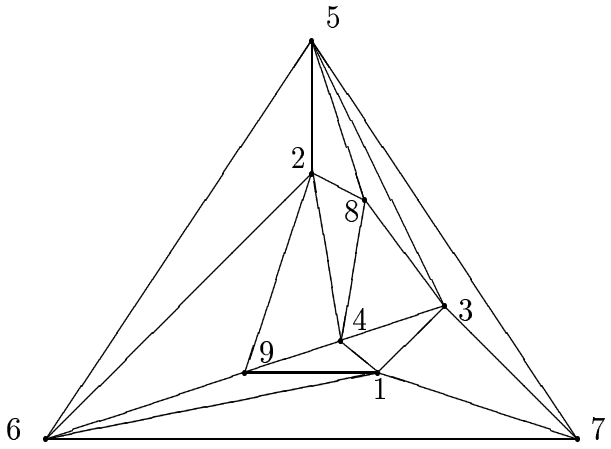

Figure 6.5: Final drawing of K_6

6.2 Results

This algorithm was tested on K_6, K_7, and K_8, and it was implemented using V: $C++$ GUI framework [39]. Drawings for K_7 and K_8 are shown in Figure 6.6 and

Figure 6.7 respectively. In those figures, the original vertices are shown as circles, and the vertices created as a result of crossings are shown as squares.

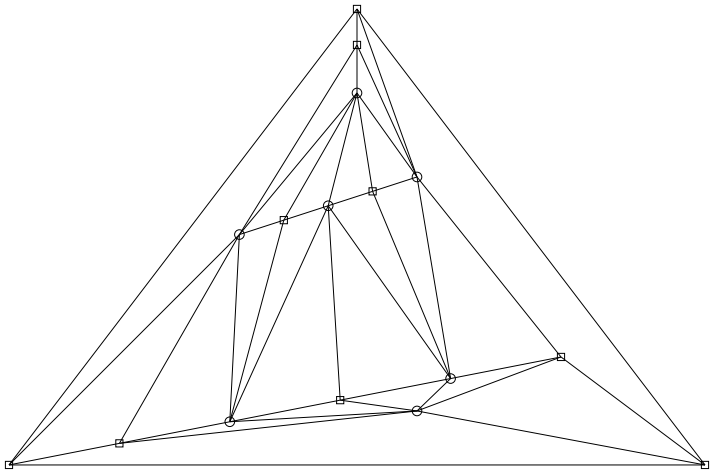

Figure 6.6: Drawing of K_7

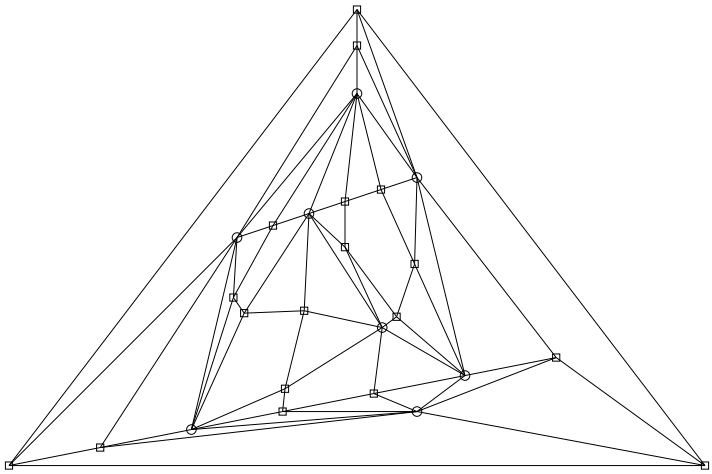

Figure 6.7: Drawing of K_8

Chapter 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 that was presented in [20]. We have implemented this algorithm, and shown its capabilities 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 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 [28] to looking at various graph theoretic ways to legally prune the tree more effectively. Secondly we would look into any similarities between solutions for K_n and K_{n-1} and try to figure out if solution for K_n can be obtained from the solution for K_{n-1} (since we have solutions for K_6, K_7, and K_8).

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 was done with the rectilinear crossing problem [36]) to

the conjecture proposed by Richard Guy [16] many years ago as the exact solution of this problem.

Another direction of future work involves further testing and enhancing current GUI interface, used to produce drawings. After getting results for more graphs, we should test our graph drawing algorithm on those graphs, possibly adding more features.

Bibliography

[1] G. Di Battista, P. Eades, R. Tamassia, and I.G. Tollis. Algorithms for drawing graphs: an annotated bibliography. *Comput. Geom.*, **4**:235–282, 1994.

[2] L.W. Beineke and R.D. Ringeisen. On the crossing numbers of products of cycles and graphs of order four. *J. Graph Theory*, **4**:145–155, 1980.

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

[4] N. Chiba, T. Yamanouchi, and T. Nishizeki. Linear algorithms for convex drawings of planar graphs. In J.A. Bondy and U.S.R. Murty, editors, *Progress in graph theory*, pages 153–173. 1984.

[5] N. Chiba, T. Yamanouchi, and T. Nishizeki. Drawing planar graphs nicely. *Acta Inform.*, **22**:187–201, 1985.

[6] M. Chrobak and T.H. Payne. A linear-time algorithm for drawing a planar graph on a grid. *Inform. Process. Lett.*, **54**:241–246, 1995.

[7] E. Damiani, O. D'Antona, and P. Salemi. An upper bound to the crossing number of the complete graph. *J. Combin. Inform. System Sci.*, **19**(1-2):75–84, 1994.

[8] A.M. Dean and R.B. Richter. The crossing number of $C_4 \times C_4$. *J. Graph Theory*, **19**(1):125–129, 1995.

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

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

[11] 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.

[12] I. Fáry. On straight line representation of planar graphs. *Acta Sci. Math. (Szeged)*, **11**:229–233, 1948.

[13] H. De Fraysseix, J. Pach, and R. Pollack. How to draw a planar graph on a grid. *Combinatorica*, **10**:41–51, 1990.

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

[15] Al Geist, Adam Beguelin, Jack Dongarra, Weicheng Jiang, Robert Manchek, and Vaidy Sunderam. *PVM: Parallel Virtual Machine – A User's guide and tutorial for networked parallel computing*. MIT Press, Cambridge, MA, 1994.

[16] 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.

[17] R.K. Guy and T.A. Jenkyns. The toroidal crossing number of $K_{m,n}$. *J. Combin. Theory Ser. B*, **6**:235–250, 1969.

[18] F. Harary, P.C. Kainen, and A.J. Schwenk. Toroidal graphs with arbitrarily high crossing numbers. *Nanta Mathematica*, **6**:58–67, 1973.

[19] D. Harel and D. Sardas. An algorithm for straight-line drawing of planar graphs. *Algorithmica*, **20**:119–135, 1998.

[20] Frederick C. Harris, Jr. and Cynthia R. Harris. A proposed algorithm for calculating the minimum crossing number of a graph. In Yousef Alavi, Allen J. Schwenk, and Ronald L. Graham, editors, *Proceedings of the Eighth Quadrennial International Conference on Graph Theory, Combinatorics, Algorithms, and Applications*, Kalamazoo, Michigan, June 1996. Western Michigan University.

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

[22] J. E. Hopcroft and R.E. Tarjan. Efficient planarity testing. *J. Assoc. Comput. Mach.*, **21**:549–568, 1974.

[23] D.J. Kleitman. The crossing number of $K_{5,n}$. *J. Combin. Theory Ser. B*, **9**:315–323, 1971.

[24] M. Klešč, R.B. Richter, and I. Stobert. The crossing number of $C_5 \times C_n$. *J. Graph Theory*, **22**(3):239–243, 1996.

[25] 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.

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

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

[28] M.J. Quinn and N. Deo. An upper bound for the speedup of parallel best-bound branch-and-bound algorithms. *BIT*, **26**(1):35–43, 1986.

[29] R.C. Read. A new method for drawing a planar graph given the cyclic order of the edges at each vertex. *Congr. Numer.*, **56**:31–44, 1987.

[30] R.B. Richter and J. Siráň. The crossing number of $K_{3,n}$ in a surface. *J. Graph Theory*, **21**(1):51–54, 1996.

[31] R.B. Richter and C. Thomassen. Intersection of curve systems and the crossing number of $C_5 \times C_5$. *Discrete Comput. Geom.*, **13**:149–159, 1995.

[32] R.D. Ringeisen and L.W. Beineke. The crossing number of $C_3 \times C_n$. *J. Combin. Theory Ser. B*, **24**:134–136, 1978.

[33] W. Schnyder. Embedding planar graphs in the grid. In *Proc. 1st. Ann. ACM - SIAM Symp. on Discrete Algorithms*, pages 138–147, San Francisco, 1990.

[34] U. Tadjiev and F.C. Harris, Jr. Parallel computation of the minimum crossing number of a graph. In Michael Heath, Virginia Torczon, Greg Astfalk, Petter E. Bjorstad, Alan H. Karp, Charles H. Koelbel, Vipin Kumar, Robert F. Lucas, Layne T. Watson, and David E. Womble, editors, *Proc. of the* 8^{th} *SIAM Conf. on Parallel Process. for Sci. Comput.*, Minneapolis, Minnesota, March 1997. SIAM.

[35] R. Tamassia and I. G. Tollis, editors. *Graph Drawing (Proc. GD '94)*, volume **894** of *Lecture notes in Computer Science*. Springer-Verlag, 1995.

[36] J.T. Thorpe and F.C. Harris, Jr. A parallel stochastic optimization algorithm for finding mappings of the rectilinear minimal crossing problem. *Ars Comb.*, **43**:135–148, 1996.

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

[38] W.T. Tutte. How to draw a graph. *Proc. London Math. Soc.*, **13**(3):743–768, 1963.

[39] Bruce Wampler. V: C++ GUI Framework. http://www.objectcentral.com.

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

[41] K. Zarankiewicz. On a problem of P. Turán concerning graphs. *Fund. Math.*, **41**:137–145, 1954.