University of Nevada
Reno

On the Crossing Number of Complete Graphs: Growing Minimal $\mathbf{K}_n$ From Minimal $\mathbf{K}_{n-1}$

A dissertation submitted in partial fulfillment of the
requirements for the degree of Doctor of Philosophy in
Computer Science and Engineering

by

Judith R. Fredrickson

Dr. Frederick C. Harris, Jr./Dissertation Advisor

May, 2006

UNIVERSITY
OF NEVADA
RENO

THE GRADUATE SCHOOL

We recommend that the dissertation
prepared under our supervision by

JUDITH R. FREDRICKSON

entitled

**On the Crossing Number of Complete Graphs:
Growing Minimal Kn from Minimal Kn-1**

be accepted in partial fulfillment of the
requirements for the degree of

DOCTOR OF PHILOSOPHY

Frederick C. Harris, Jr., Ph.D., Advisor

Yaakov L. Varol, Ph.D., Committee Member

Dwight Egbert, Ph.D., Committee Member

Robert Davis, Ph.D., Committee Member

Indira Chatterjee, Ph.D., Graduate School Representative

Marsha H. Read, Ph. D., Associate Dean, Graduate School

May, 2006

Acknowledgments

Many individuals have supported me through the process of completing this project. Without their help, support, and encouragement the entire process would not have been so enjoyable.

First I would like to thank my advisor, Dr. Frederick C. Harris, Jr, for his inspiring and encouraging guidance, and his invaluable discussions over the last several years. Without his foray into graph theory I doubt this dissertation topic would have presented itself. I can only hope to show my future students the constant positive attitude and encouragement he has always shown me.

I am also grateful to my other committee members: Dr. Yaakov L. Varol, Dr. Dwight Egbert, Dr. Robert Davis and Dr. Indira Chatterjee. Each contributed to this experience in numerous ways.

Two individuals who dedicated a great deal of their time helping with the implementation of many of my ideas, modified often out of curiosity, are Bei Yuan and Linda Humphrey. Their help and discussions were invaluable.

Thanks also to Cindy Harris for her meticulous reading of my work, both through the eyes of an editor and through those of someone who knows the topic well.

I must thank my dear friend and colleague Nancy Latourette. Her almost daily questions, discussion, and support kept me going when excited and also when wondering if things would ever come together. Her willingness to always have an open door for an extended "hello" was greatly appreciated and her diversions into other life conversations were presented when I needed them most.

The Department of Computer Science and Engineering has been most supportive in providing a good working environment and supporting me with a lecture position within the department.

Finally, I thank my husband, Joel, and my two daughters, Kelci and Kirsha. Their love and support throughout this entire process is what ultimately made it possible. I dedicate my work to them.

April 25, 2006

Abstract

This dissertation addresses the generation of minimal complete graphs. A complete graph, denoted K_n, is a graph in which each vertex is connected to every other vertex of the graph with an edge. A minimal graph is a representation of a graph that exhibits its crossing number. The crossing number of a graph is the smallest number of edge crossings over all planar representations of the graph.

Presented is a constructive framework for iteratively generating minimal K_n from minimal K_{n-1}. The process described allows for complete enumeration of all minimal K_n derivable from minimal K_{n-1}, including those graphs that are only locally minimal. A graph is locally minimal if it reflects the smallest number of crossings possible for a given region placement of vertex n in minimal K_{n-1}. All graphs are classified into isomorphic families, allowing for efficient iteration to the next set of complete graphs of order $n+1$ due to needing only one representative from each K_n isomorphic family for complete results.

In addition to determining the crossing number for K_n generated by K_{n-1}, all minimal graphs are available for exploration of their underlying structure. This allows for substructure isomorphic analysis and localized region neighborhood analysis, both of which will lead to yet more efficient iteration of K_n for increasing n. Information culled from these graphs may also shed light on new or improved lower bound estimation techniques.

A seven-step process is presented that allows for iterative growth. The main module in the process, Star Analysis, takes a global view of growing minimal K_n by focusing on optimizing the enumeration of all star graphs, $K_{1,n-1}$, centered at vertex n placed into initial minimal K_{n-1} at all possible different region locations.

This research introduces new conjectures and opens some new questions for exploring the structure of minimal K_n. Though this work focuses on minimal complete graphs, the process appears to be applicable to an assortment of other graph families. An example is presented of its application to complete bipartite graphs.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Determining how to draw a graph in the plane with the minimum number of edge crossings is referred to as the crossing number problem. It is generally acknowledged that the birth of the crossing number came about due to an experience of Paul Turán while in a labor camp at a brick factory near Budapest, Hungary in 1944. His story [30] is as follows:

> Our work was to bring out bricks from the ovens where they were made and carry them on small vehicles which run on rails in some of several open stores which happened to be empty. Since one could never be sure which store would be available, each oven was connected by rail with each store. Since we had to settle a fixed amount of loaded cars daily it was our interest to finish it as soon as possible. After being loaded in the (rather warm) ovens the vehicles run smoothly with not much effort; the only trouble arose at the crossing of two rails. Here the cars jumped out, the bricks fell down; a lot of extra work and loss of time arose. Having this experience a number of times it occurred to me why on earth did they build the rail system so uneconomically; minimizing the number of crossings the production could be made much more economical.[*sic.*]

Turán's brick factory problem asks the question of whether the minimum number of edge crossings of a complete bipartite graph of order n can be determined. (Refer to Section 2.1 for a complete listing of all related definitions as needed.) This problem

proved to be extremely difficult and, over the years, in addition to work on bipartite graphs, researchers explored minimizing crossings of general graphs as well as many other graph families.

The crossing number problem has a number of applications. In addition to possible application in logistical situations like that encountered by Turán, two main areas of direct application are information display and circuit design.

As Battista *et al.* state [8], the visualization of complex conceptual structures is a key component of support tools for many applications in science and engineering. Graphs are abstract structures used to model information; thus many information visualization systems require graphs to be drawn so they are easy to read and understand. Minimizing the crossings of the edges of these graphs makes them easier to read as well as more esthetically pleasing. Those interested in exploring graph visualization are referred to [7] and [8].

The crossing number problem has also found application in the area of Very Large Scale Integration (VLSI) circuit technology. Leighton [38, 39] shows the relevance of crossing numbers to the problem of reducing chip layout area and minimizing chip size. Area and size reduction lower production costs and allow for more reliable performance of larger chips. He showed the close relationship between the crossing number and the bisection width of a graph. Both bisection width and the crossing number are important properties of graphs that affect minimum layout area for VLSI design.

Finding the exact value for crossing numbers is not an easy task. It is a hard combinatorial problem, examples of which can be found in [12] and [31]. The difficult nature of determining the exact value of the crossing number of a complete graph of order n, denoted $\nu(K_n)$, even for small values of n is illustrated by the small amount of literature from the early 1960s to date, for example see [19, 23, 29, 30, 31, 33, 34, 52, 53, 61]. However, this is not a reason to leave the problem for estimates only. Being able to determine the exact crossing number of a graph and having access to graphs with this number of crossings will allow researchers to thoroughly

investigate the nature of these graphs. Information may lie within the structure of these minimal graphs that will bring new insight into minimal graphs in general. Work by Aichholzer *et al.* [1, 3, 4, 5] is a good example of computational experiments, performed with care and in combination with a theoretical basis, helping shed light on difficult combinatorial problems.

This dissertation addresses the crossing number problem as related to complete graphs. The contribution to the crossing number problem is the creation of an effective constructive technique for building minimal complete graphs of order n from minimal complete graphs of order $n-1$. This technique allows for the construction of all representative graphs of minimal K_n grown from minimal K_{n-1}. In addition to globally minimal K_n, locally minimal K_n (given specific region placement of vertex n into K_{n-1}) are available for construction. All isomorphic families available from this growth pattern will be available for enumeration and drawing as well as topological, spectral, and other desired analysis. Also presented are results leading to several new open questions to explore related to complete graphs and other graphs families. Example images of minimal K_{10} and K_{11} with 60 and 100 crossings, respectively, are included.

The technique introduced, though applied to complete graphs in this paper, has direct application to other graph families with no modification of the process needed. An example of its application to the complete bipartite graph of order m, n is shown in Chapter 5.

The remainder of this dissertation is structured as follows: Chapter 2 presents basic definitions and background information on graphs, including a survey of the related literature. Description of the algorithm from which this work begins, along with its shortfalls are addressed. Chapter 3 presents the new contributions that lead to an effective algorithm for functional calculation of minimal K_n from minimal K_{n-1}. Improvements achieved, due to search space restrictions and a new technique, Star Analysis, are included in Chapter 3. Chapter 4 discusses the iterative framework developed for growing minimal complete graphs. Conclusions and a look at open

problems that have presented themselves, including new conjectures related to minimal K_n, appear in Chapter 5.

Chapter 2

Background

In this chapter the basic definitions needed for reading this dissertation are presented. All background definitions are collected in Section 2.1. Section 2.2 covers the history and related work on crossing numbers with the main focus on the crossing number of complete graphs. A brief overview of graph isomorphism, as it relates to this work, is given in Section 2.2.3. Finally, Section 2.3 describes in detail the exhaustive search algorithm from which this new research initiated. This background allows for clear understanding as progress is made toward growing minimal K_n from minimal K_{n-1}.

2.1 Basic Definitions

There are several books available on graph theory. [13] and [57] are two that each have a small section dedicated to crossing numbers. The basis for the definitions in this chapter come from [13].

Definition 1 *A simple* ***graph*** *G is a finite nonempty set of objects called vertices together with a set of unordered pairs of distinct vertices of G called edges.*

The vertex set of G is denoted V(G) and the edge set is denoted E(G). Throughout this dissertation, all graphs are simple unless otherwise noted.

Given two vertices, u and v, of graph G, the edge $e = (u,v)$ joins u and v. Common notation for the edge $e = (u,v)$ is uv. This more convenient notation is used when possible. If $e = uv$ is an edge of G, then u and v are called *adjacent vertices*, u and

e are *incident* as are v and e. If e_1 and e_2 are distinct edges of G with a common vertex, it is said that e_1 and e_2 are *adjacent edges*.

Definition 2 *The **order** of graph G is the cardinality of the vertex set of G, commonly denoted n(G) or n.*

Definition 3 *The **size** of graph G is the cardinality of the edge set of G, commonly denoted m(G) or m.*

Definition 4 *The **degree** of a vertex v is the number of edges incident with v, commonly denoted deg v.*

In figure 2.1 the above terms are illustrated for two graphs. The graph in (a) has order $n = 4$ and size $m = 4$. Vertices 1 and 2 each have degree 2, vertex 3 has degree 3, and vertex 4 has degree 1. Graph (b) has order $n = 5$, size $m = 10$, and all five vertices have degree 4.

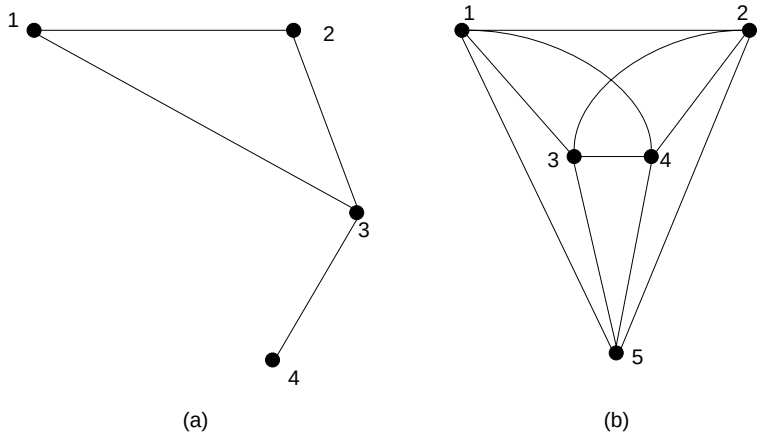


Figure 2.1: Example Graphs

Definition 5 *A graph G is **isomorphic** to graph H if there exists a one-to-one mapping ϕ, called an **isomorphism**, from V(G) onto V(H) such that ϕ preserves adjacency; i.e. $uv \in E(G)$ if and only if $\phi(u)\ \phi(v) \in E(H)$.*

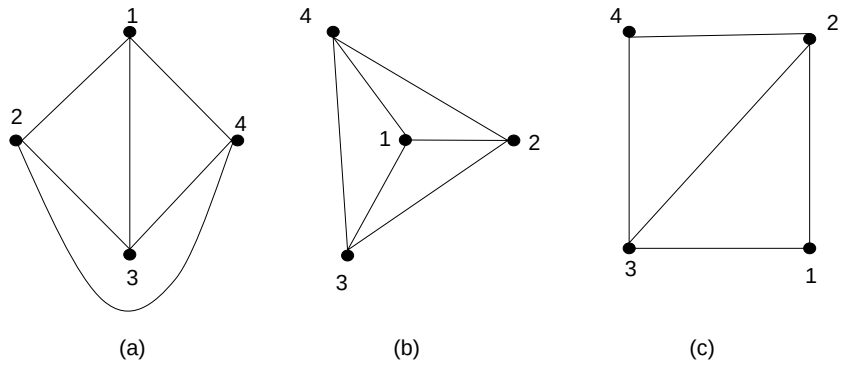


Figure 2.2: Isomorphic and Nonisomorphic Graphs

The two graphs in Figure 2.2(a) and (b) are isomorphic, but graph (c) is not isomorphic to either (a) or (b).

Definition 6 *A graph G is a* ***subgraph*** *of graph H if V(G)* $\subseteq$ *V(H) and E(G)* $\subseteq$ *E(H).*

Definition 7 *A graph G is* ***complete*** *if every two of its vertices are adjacent. The notation for the complete graph of order n is* K_n.

In Figure 2.3, graph (a) is a complete graph of order six. Graph (b) is a complete graph of order 5, and graph (c) is not complete. Graphs (b) and (c) are both subgraphs of graph (a).

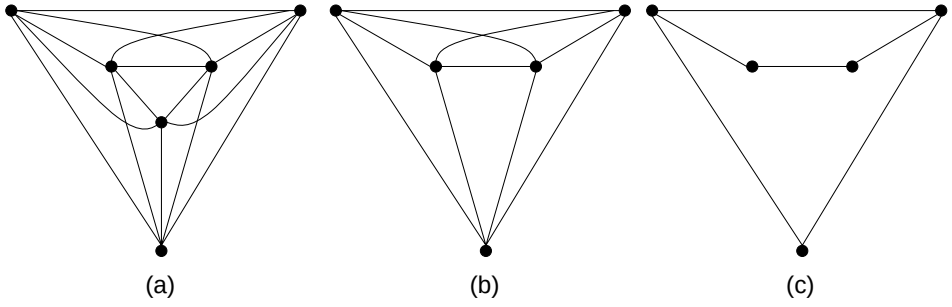


Figure 2.3: A Complete Graph and Two Subgraphs

Definition 8 *A graph G is* ***k-partite****, $k \geq 1$, if it is possible to partition V(G) into k subsets, $V_1, V_2, ..., V_k$, called partite sets, such that every element of E(G) joins a vertex of V_i to a vertex of V_j, $i \neq j$.*

Definition 9 *A* ***complete k-partite*** *graph G is a k-partite graph with partite sets $V_1, V_2, ..., V_k$ having the additional property that if $u \in V_i$ and $v \in V_j$, $i \neq j$, the $uv \in E(G)$.*

The *complete bipartite graph* consisting of two partite sets of order m and n is denoted K(n,m) or $K_{n,m}$. The graph $K_{1,n}$ is called a *star*.

A complete bipartite graph illustrating Turán's brick factory problem is seen in Figure 2.4. To visualize the brick factory layout, let vertices 1 and 2 represent the brick ovens (one partite set), vertices 3, 4, and 5 represent the storage areas (the second partite set), and the edges of the graph represent the rails connecting all ovens to all storage locations.

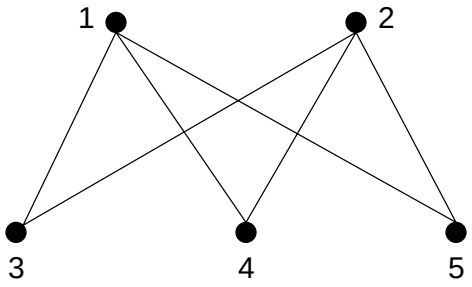


Figure 2.4: A Bipartite Graph

Definition 10 *The* ***cartesian product*** *graph $F = G \times H$ has V(F) = V(G) × V(H), and two vertices (u_1, u_2) and (v_1, v_2) are adjacent iff either $u_1 = v_1$ and $u_2v_2 \in E(H)$, or $u_2 = v_2$ and $u_1v_1 \in E(G)$.*

Definition 11 *A* ***u-v walk*** *of graph G is a finite, alternating sequence*

$$u = u_0, e_1, u_1, e_2 ..., u_{k-1}, e_k, u_k = v$$

of vertices and edges, beginning with vertex u and ending with vertex v, such that $e_i = u_{i-1}u_i$ *for i=1, 2, ..., k. The length of the walk is denoted by k.*

A *u-v path* is a *u-v* walk in which no vertex is repeated.

Definition 12 *A vertex u is* ***connected*** *to vertex v in graph G if there exists a u-v path in G. A graph G is connected if every two of its vertices are connected.*

It should be obvious that every complete graph, K_n, is connected.

Definition 13 *The* ***bisection width*** *of a graph is the minimum number of edges that must be removed from the graph in order to disconnect it into two equal-sized pieces. Two pieces are equal-sized if the number of vertices in each differs by no more than one. Bisection width is denoted bw(G).*

Figure 2.5 shows a graph with the dashed lines indicating the edges that need to be deleted to disconnect it into two equal-sized pieces. The bisection width of the graph is 3.

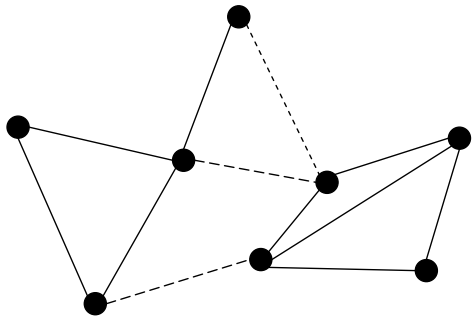


Figure 2.5: The Bisection Width of a Graph

Definition 14 *The* ***cutwidth*** *of a graph G with vertex set V,edge set E, and injection* $\phi : V \leftarrow \{1,2,3,...,|V|\}$ *is defined as*

$$cw(G) = \min_{\phi} \max_{i} |\{uv \in E : \phi(u) \leq i < \phi(v)\}|$$

.

The cutwidth problem is one of arranging vertices in a line so that the maximum number of edges crossing the ith place, for all i, is minimized. The illustrations in Figure 2.6 aid in understanding the cutwidth definition. Graph (a) is shown as a linear embedding in (b). The horizontal dashed line crosses the region between vertices 2 and 4 three times, so the region between 2 and 4 has a cut of 3 indicated by the three edges intersected. The cutwidth of the graph is the minimum of all possible maximum cuts over all possible linear embeddings. The cutwidth of graph (a) is three.

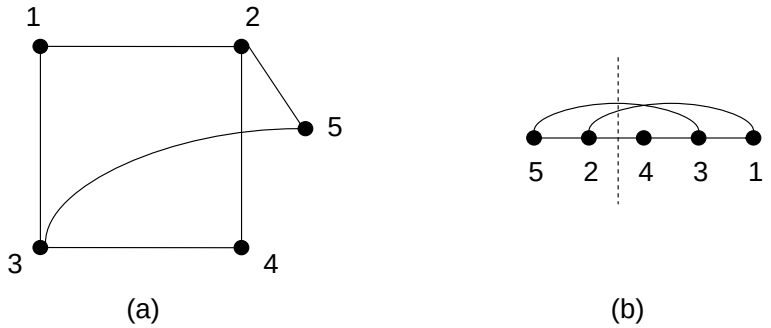


Figure 2.6: The Cutwidth of a Graph

Definition 15 *A graph G with order n and size m is **realizable** or **embeddable** on a surface S if it is possible to distinguish a collection of m curves, pairwise disjoint except possibly for endpoints, on S that correspond to the edges of G such that if a curve A corresponds to the edge $e = uv$, then only the endpoints of A correspond to vertices of G, namely u and v.*

Definition 16 *A graph is **planar** if it can be embedded in the plane.*

Embedding a graph in the plane is equivalent to embedding it on the sphere.

Definition 17 *A planar graph that is embedded in the plane is called a **plane** graph.*

Figure 2.7 graph (a) is a planar graph, though as drawn it is not plane. The illustration in (b) is its plane representation.

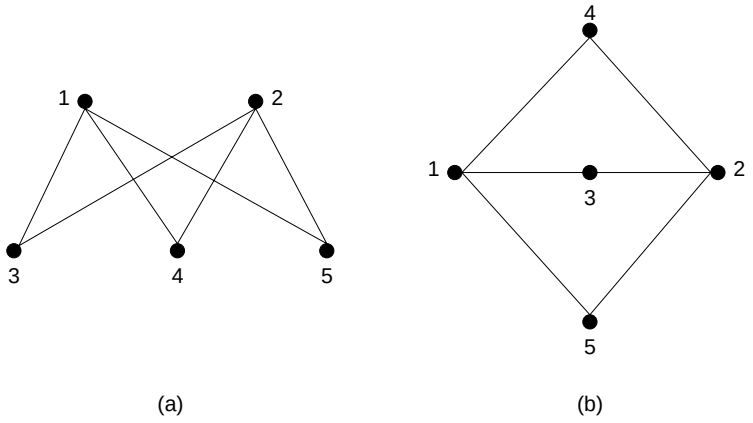


Figure 2.7: A Planar and Plane Graph

Definition 18 *Given a plane graph G, a* ***region*** *of G is a maximal portion of the plane for which any two points may be joined by a curve A such that each point of A neither corresponds to a vertex of G nor lies on any curve corresponding to an edge of G.*

The regions of G can be thought of as the disjoint portions of the plane remaining after all the edges and vertices have been removed.

Definition 19 *The* ***boundary*** *of a region R of a plane graph G consists of all points x corresponding to vertices and edges of G having the property that x can be joined to a point of R by a curve, all of whose points that differ from x belong to R.*

For illustration, Figure 2.8 shows the regions of K_4 along with their boundaries. Region R1 is bounded by vertex 1, edge (1,2), vertex 2, edge (2,4), vertex 4 and edge (4,1). The other region boundaries can be determined similarly. Notice that there is an exterior region, R4. It is bounded by vertex 1, edge (1,3), vertex 3, edge (3,2), vertex 2, and edge (2,1). For the discussion in this dissertation it is said that a vertex or edge on the boundary of a region is adjacent to that region.

The order, size, and number of regions of any connected plane graph are related by Euler's Formula, stated in Theorem 1.

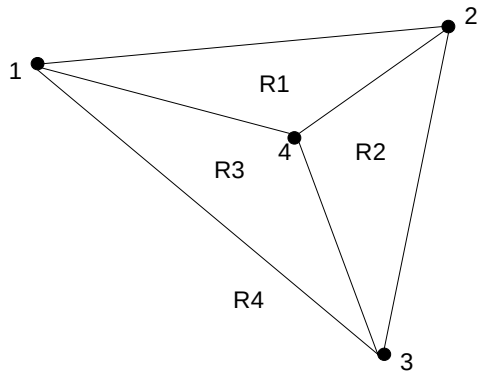


Figure 2.8: The Regions of K_4 and Their Boundaries

Theorem 1 *[Euler's Formula] If G is a connected plane graph with n vertices, m edges and r regions, then*

$$n - m + r = 2.$$

It follows from Theorem 1 that every two embeddings of a connected planar graph in the plane result in plane graphs with the same number of regions. This allows for discussion of the number of regions of a connected planar graph.

Nonplanar graphs cannot be drawn in the plane; some edges must cross. This leads to the definition of the crossing number of a graph, a standard measure of graph nonplanarity, as well as to the definition of a good drawing.

Definition 20 *The **crossing number** $\nu(G)$ of a graph G is the minimum number of edge crossings among the drawings of G in the plane.*

Definition 21 *A **good drawing** of a graph G satisfies the following:*

- *adjacent edges never cross,*
- *two nonadjacent edges cross at most once,*
- *no more than two edges cross at a point of the plane, and*
- *no edge passes through a vertex of the graph G.*

Figure 2.9 illustrates the four edge relationships listed in Definition 21 that are not allowed in good graph drawings.

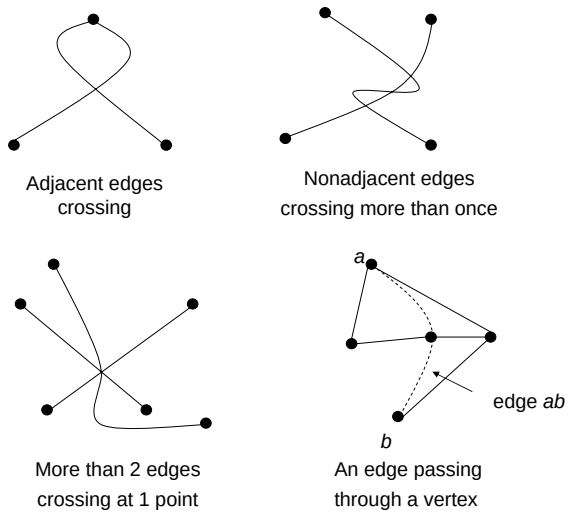


Figure 2.9: Illegal Edges for a Good Graph

Definition 22 *A **minimal graph** G is a graph with the minimum number of edge crossings among the drawings of G.*

Definition 23 *The **rectilinear crossing number** of a graph G, denoted $\bar{\nu}(G)$, is the minimum number of edge crossings among all drawings of G in the plane in which each edge is a straight line segment.*

Fáry [21] is known for the earliest result concerning the drawing of a graph in the plane. He showed that every planar graph can be embedded in the plane so that every edge is a straight line segment. In terms of crossing numbers this implies that if $\nu(G) = 0$ then $\bar{\nu}(G) = 0$.

The following definitions are relevant to the discussion in Section 2.3.1 of Edmonds' Rotational Embedding Scheme. A compact orientable 2-manifold is a surface that can be thought of as a sphere with attached handles. The number of handles is referred to as the genus of the surface.

Definition 24 *The **genus** of a graph G is the smallest genus of all surfaces on which G can be embedded. The genus of a graph is denoted gen(G).*

Since, as mentioned, the embeddings of graphs on planes or spheres is equivalent, the graphs of genus 0 are exactly planar graphs. This dissertation is concerned only with graphs of genus 0.

Definition 25 *A region of a graph G is **2-cell** if any simple closed curve in that region can be continuously deformed or contracted in that region to a single point.*

Every region of a connected graph embedded on the sphere is 2-cell. However this may not be the case for connected graphs embedded on surfaces of positive genus.

Definition 26 *If all the regions of a graph G embedded on a surface S are 2-cell, the embedding is called a **2-cell embedding**.*

Via an extension of Euler's Formula (Theorem 1), it is noted that every embedding of a connected graph G on a surface of genus gen(G) results in the same number of regions.

2.2 Background and Related Literature

This section discusses the background of crossing numbers and provides a brief survey of the relevant literature. The main focus is on the crossing number of complete graphs. The crossing number problem, so difficult to attack in general, has followed many splintered paths as individuals have attempted to make progress on the problem from different perspectives. There are many results with not a great deal of generalization evident. Bits and pieces have been proven over a wide variety of graph families and different types of edge crossings. An overview of the crossing number of complete graphs is presented in Section 2.2.1. Section 2.2.2 briefly looks at the work being done on other types of crossing numbers and other graph families. Section 2.2.3 concludes with an overview of graph isomorphism as it relates to this body of work.

2.2.1 The Crossing Number of Complete Graphs

As mentioned in the Chapter 1, Turán's brick factory problem asks the question of finding the crossing number of a complete bipartite graph. Zarankiewicz [62] proposed a solution to the problem in 1953. Guy [30] points out an error in Zarankiewicz's proof that was discovered in 1965 by Kainen and Ringel and shared with Guy *via* private communication. The error was the assumption that among the m graphs $\mathrm{K}_{m,1}$ that compose $\mathrm{K}_{m,n}$ it is always possible to find two which do not contain a crossing. To date, Zarankiewicz's conjecture has not been proven or disproven and stands as an upper bound.

Theorem 2 *[Zarankiewicz] The crossing number of the complete bipartite graph $K_{m,n}$ satisfies the inequality*

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

Conjecture 1 *[Zarankiewicz] The crossing number of the complete bipartite graph $K_{m,n}$ satisfies the equality*

$$\nu(K_{m,n}) = \left\lfloor \tfrac{m}{2} \right\rfloor \left\lfloor \tfrac{m-1}{2} \right\rfloor \left\lfloor \tfrac{n}{2} \right\rfloor \left\lfloor \tfrac{n-1}{2} \right\rfloor.$$

In 1960, Guy [29] popularized the search for the crossing number of a complete graph with the introduction of an upper bound, confirmed by Blazek and Koman [11]. Guy mentioned that Erdös had been looking at the problem for at least twenty years prior. No improved upper bound has been published to date.

Theorem 3 *[Guy] The crossing number of the complete graph K_n satisfies the inequality*

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

Another representation of the inequality in Theorem 3 is:

$$\nu(\mathrm{K}_n) \leq \tfrac{1}{64} n(n-2)^2(n-4), \text{ for } n \text{ even.}$$
$$\nu(\mathrm{K}_n) \leq \tfrac{1}{64} n(n-1)^2(n-3)^2, \text{ for } n \text{ odd.}$$

It is interesting to note that if equality holds for Theorem 3 for n odd, then by a straightforward induction proof it can be shown to hold for the next value of n (even). The inductive step from even to odd cannot be made. As Guy says, "We are trying to walk, using only one leg." This same one-legged induction holds for the inequality in Theorem 2 [31].

It has been conjectured [29, 33] that equality holds in Theorem 3 for all n. This conjecture, made in the early 1960s, has not been proven or disproven to date.

Conjecture 2 *[Guy [29], Harary and Hill [33]] The crossing number of the complete graph K_n satisfies the equality*

$$\nu(K_n) = \tfrac{1}{4}\left\lfloor \tfrac{n}{2} \right\rfloor \left\lfloor \tfrac{n-1}{2} \right\rfloor \left\lfloor \tfrac{n-2}{2} \right\rfloor \left\lfloor \tfrac{n-3}{2} \right\rfloor.$$

In 1963, Harary and Hill mention the rectilinear crossing number, denoted $\bar{\nu}(\mathrm{K}_n)$ for complete graphs with n vertices, and illustrate examples of conjectured differences *via* drawings of $\nu(\mathrm{K}_8)$ and $\bar{\nu}(\mathrm{K}_8)$. They composed Table 2.1 of conjectured values for $\nu(\mathrm{K}_n)$ and $\bar{\nu}(\mathrm{K}_n)$. It is included here as a historical fact and as a reflection on the slow advances in the crossing number problem. Table 2.1 as related to $\nu(\mathrm{K}_n)$ has seen no modification since its inception. For the current status of $\bar{\nu}(\mathrm{K}_n)$, see Section 2.2.2.

n	2	3	4	5	6	7	8	9	10
$\nu(\mathrm{K}_n)$	0	0	0	1	3	9	18	36	60
$\bar{\nu}(\mathrm{K}_n)$	0	0	0	1	3	9	19	36	63

Table 2.1: Conjectured Crossing Numbers in 1963

In the same discussion, by Harary and Hill, Conjecture 3 was proposed.

Conjecture 3 *[Harary and Hill] For complete graphs, the rectilinear crossing number, $\bar{\nu}(K_n)$, exceeds the crossing number, $\nu(K_n)$, for $n = 8$ and all $n \geq 10$.*

In 1971, Singer [50] verified Harary and Hill's conjecture that $\nu(\mathrm{K}_n) \neq \bar{\nu}(\mathrm{K}_n)$ for $n = 8$ and $n = 10$. He proved that $\bar{\nu}(\mathrm{K}_8) = 19$ and $60 < \bar{\nu}(K_{10}) < 63$.

Guy supplied us with Theorem 4 in 1972. The reader is encouraged to refer to [31] for proofs establishing equality of Theorem 3 for $n \leq 10$.

Theorem 4 *[Guy] For the complete graph K_n, for $n \leq 10$, it is the case that*

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

In addition to proofs for $n \leq 10$, [19] and [31] supply a plethora of information, some of which is briefly described now. The introduction of the responsibility argument, which is later used [12] in a proof that $\bar{\nu}(\mathrm{K}_{10}) = 62$, is presented. The *responsibility* of a vertex is defined as the total number of crossings on all edges incident to that vertex. Theorem 5 from [19, 31] supplies information on the parity of K_n, and Theorem 6 addresses the crossing number of sub-drawings.

Theorem 5 *[Guy,Eggleton] The parity of the number of crossings in all good drawings of K_n is the same for n odd.*

Theorem 6 *[Guy] Given any drawing of K_n with c crossings, there exists at least one sub-drawing K_{n-1} with at most, and one with at least, $\left\lfloor\frac{n}{n-4}c\right\rfloor$ crossings.*

Guy also conjectured a solution to the rectilinear crossing number problem. See Section 2.2.2 for details. Eggleton [19] and Harbo et al. [28] supply work on generating all nonequivalent complete graphs for $n \leq 6$ as well as for some other graph families.

Garey and Johnson helped explain why advancement in solving the crossing number problem was so trying. They proved the crossing number problem to be NP-complete [25] as Theorem 7 states.

Theorem 7 *[Garey and Johnson] Given a graph G and an integer K, the question of if $\nu(G) \leq K$ is NP-complete.*

After this discovery, much research turned away from the crossing number problem to other related problems, although it was not abandoned completely. Research mainly focused on lower bound tightening, and the quest for exact values fell off. For the reader interested in NP-completeness, refer to [24]. The rectilinear crossing number problem has been found to be NP-hard [9]. Pach and Toth [44] show that the

odd crossing number problem is NP-complete and that the pairwise crossing number problem is NP-hard.

Harris and Harris [34] proposed an exhaustive search algorithm for calculating the crossing number of a graph. This algorithm was implemented in parallel by Tadjiev and Harris [52, 53]. The test family of graphs was the complete graph family. Needed improvements, a dynamic work queue and graph theoretic reductions of the search space, became evident from this initial parallel implementation. This algorithm is explained in more detail in Section 2.3 since it is the algorithm from which this dissertation grew.

While there are a few infinite classes of graphs for which tight bounds are known [10, 40], efficient lower bound methods for estimating the crossing number of a variety of graphs are lacking. One of the methods seeing a lot of use in the lower bound search is the bisection method. It has proven to be one of the most powerful. In Leighton's article [39] discussing the use of the crossing number to estimate the required chip area for VLSI circuit layout of graph, he proved a general lower bound for $\nu(\mathrm{K}_n)$ using the relationship between the crossing number and the bisection width. This lower bound was also found independently by Ajtai *et al.* [6]. Many extensions of this lower bound have been discovered [38, 39, 43, 51]. The best of these bounds combines Leighton's bound with the constant $\frac{1}{33.75}$ by Pach *et al.* [46], producing Theorem 8.

Theorem 8 *[Leighton, Ajtai et al., Pach et al.] Let G be a graph with n vertices and e edges with $e \geq 7.5n$. Then*

$$\nu(G) \geq \frac{1}{33.75}\frac{e^3}{n^2}.$$

A similar inequality holds for the odd crossing number, see [44], with a crossing constant of $\frac{1}{64}$.

Another technique, using the cutwidth of a graph, was spawned from this extension of Leighton's lower bound. An improvement of that bound was discovered [16] by replacing the bisection width with the cutwidth of the graph. In using the cutwidth

of a graph, Djidjev and Vrt'o were also able to find an upper bound on pathwidth of G in terms of its crossing number.

Pach *et al.* [46] prove the conjecture by Erdös and Guy [20] that $K(n,e)\frac{n^2}{e^3}$ tends to a positive constant as n approaches infinity and $n << e << n^2$ to produce Theorem 9. $K(n,e)$ is defined to be the minimum of ν(G) taken over all graphs with n vertices and at least e edges.

Theorem 9 *[Pach et al.] Given a graph G, with n vertices and at least e edges, if* $n << e << n^2$,

$$\lim_{n\to\infty} K(n,e)\frac{n^2}{e^3} = C > 0$$

where K(n,e) is the minimum of ν(G) taken over all graphs with n vertices and at least e edges.

They also reported on some new bounds for graphs with monotone properties. Specifically, given a graph G with n vertices, $e \geq 4n$ edges without a cycle of length 4, then its crossing number is at least $\frac{ce^4}{n^3}$ where $c > 0$ is a suitable constant. A graph without a cycle of length 6 results in a crossing number of at least $\frac{ce^5}{n^4}$. The bisection width and crossing number [38] aided in these results.

In the search for an improved lower bound for $\nu(K_{7,n})$, de Klerk *et al.* [15], using quadratic optimization techniques, discovered improved bounds for K_n. They present their results in terms of asymptotic ratios as follows. Let $G(K_n)$ be the conjectured crossing number for the complete graph of order n as stated in Conjecture 2, then

$$\lim_{n\to\infty} \frac{\nu(K_n)}{G(K_n)} \geq 0.83$$

Asymptotic ratios are an eloquent way to appreciate how close research is to Guy's conjectured result. The limit above, call it C, was described and shown to exist [48] along with A and B as follows:

$$A = \lim_{n\to\infty} \frac{\nu(K_{m,n})}{Z(K_{m,n})} \quad and \quad B = \lim_{n\to\infty} \frac{\nu(K_{n,n})}{Z(K_{n,n})}$$

where Z($K_{m,n}$) is Zarankiewicz's Conjecture 1, and it is shown that $C \geq B$. These results allowed de Klerk *et al.* to arrive at their new lower bound for $\nu(K_n)$ as an extension of the following bipartite lower bounds they found, also reported as asymptotic ratios:

$$\lim_{n\to\infty} \frac{\nu(K_{m,n})}{Z(K_{m,n})} \geq 0.83\frac{m}{m-1} \quad \textit{and} \quad \lim_{n\to\infty} \frac{\nu(K_{n,n})}{Z(K_{n,n})} \geq 0.83$$

Richter and Thomassen [48] have a good overview of $\nu(K_n)$ and $\nu(K_{m,m})$ in addition to the above mentioned asymptotic ratios.

Grohe has recently shown [27] that the crossing number problem is fixed parameter tractable.

Definition 27 *A problem is fixed parameter-tractable if there is a constant $c \geq 1$ such that for every fixed k the problem can be solved in time $O(n^c)$.*

He showed that for every fixed k there is a quadratic time algorithm that decides if a graph G has crossing number at most k. This result has only theoretical interest as the run time is $O(f(k)n^2)$ where f is at least doubly exponential. Yet, as Grohe states, knowing that the crossing number problem is fixed parameter-tractable may help researchers find algorithms that have practical application for small values of k.

For a fairly complete and up to date list of papers related to crossing numbers see [56]. Liebers' work [40] has a section on crossing numbers along with several references to crossing number work related to the hypercube of dimension n, Cartesian product graphs, and others. Section 2.2.2 mentions other crossing number references. Additionally, Pach and Toth [44] discuss several problems on crossing numbers. Their paper includes several open questions along with a list of references.

2.2.2 Rectilinear and Other Crossing Numbers

The crossing number is the minimum number of crossings with which it is possible to draw graph G in the plane. The result is the same whether G is drawn in the plane or on the surface of a sphere. This is not the same number obtained by drawing G on

a sphere with the edges as arcs of great circles, the rectilinear crossing number [33]. Rectilinear research restricts edges to geodesics, *i.e.* straight lines in the plane or to great circle arcs on the sphere. The rectilinear problem has a geometric nature that can be exploited, the convex hull, making it more accessible than the standard crossing number to researchers hoping to find exact values. For this reason, rectilinear crossing numbers have seen extensively more research than crossing numbers.

Guy conjectured that the upper bound for the rectilinear crossing number, given in Theorem 10, and presented in [19, 36], was an equality. The conjecture has been disproven (see [4, 12, 50, 54]), but the upper bound still stands.

Theorem 10 *[Guy] The rectilinear crossing number of the complete graph K_n satisfies the inequality*

$$\bar{\nu}(K_n) \leq \lfloor \frac{(7n^4-56n^3+128n^2+48n\lfloor \frac{(n-7)}{3} \rfloor+108)}{432} \rfloor.$$

The search for the exact value of the rectilinear crossing number problem has seen advancement recently. These advances were accomplished *via* exhaustive search techniques. The exact value of $\bar{\nu}(\mathrm{K}_n)$ for $n \leq 9$ has been known for some time [20]. That $\bar{\nu}(\mathrm{K}_{10}) = 61$ or 62 was found by Singer in 1971 in an unpublished manuscript [50]. This long standing question was answered, $\bar{\nu}(\mathrm{K}_{10}) = 62$, by an intricate combinatorial proof presented by Brodsky *et al.* [12], in 2000. This result was confirmed by Aichholzer *et al.* [4] *via* exhaustive enumeration of all combinatorially inequivalent sets of points (order types). See [2, 5] for a description of this enumeration method. Aichholzer *et al.* [1, 2] have determined $\bar{\nu}(\mathrm{K}_n)$ for up to $n = 17$. Table 2.2 illustrates the rectilinear crossing number results to date.

n	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
$\bar{\nu}(\mathrm{K}_n)$	0	0	0	1	3	9	19	36	62	102	153	229	324	447	603	798

Table 2.2: Rectilinear Crossing Numbers up to $n = 17$

Guy *et al.* [32] introduced the idea of the toroidal crossing number, $cr_1(K_n)$, with some preliminary results for $n \leq 10$. The examination of the torus was a step toward

looking at crossing numbers on orientable surfaces of higher genus. They also make a reference (unpublished) to Ringel as introducing the idea of a local crossing number.

Pach and Toth [44, 45] define pairwise crossing number, pair-cr(G), and odd crossing number, odd-cr(G), and examine the relationship between them and the crossing number and ask if it is true that, for every graph G

$$\textit{odd-cr(G)} = \textit{pair-cr(G)} = \nu(G).$$

Just recently, Pelsmajer *et al.* [47] claim to have shown that $odd - cr(G) \neq \nu(G)$ using weighted maps on the annulus, a disk with a hole in the center.

Conjecture 1 for the crossing number of $\mathrm{K}_{m,n}$ has been verified for $\min(m, n) \leq 6$ by Kleitman [37] and also by Woodall [58] for the cases $7 \leq m \leq 8$ and $7 \leq n \leq 10$. It is unknown for all other values. For discussion on the best known lower bounds for differing cases of $\nu(\mathrm{K}_{m,n})$, see [15]. These bounds were achieved using combinatorics in conjunction with quadratic optimization techniques. Shahrokhi *et al.* [49] looked at a lower bound argument for bipartite graphs based on Menger's Theorem which relates the bipartite crossing number of a graph to the edge connectivity properties of the graph. They also make use of spectral graph theory.

One graph family for which there has been success in finding exact values is the Cartesian products of cycles $C_m \times C_n$. Glebsky and Salazar [26] have proven for all but finitely many n, and for each m the long standing conjecture that $\nu(C_m \times C_n) = (m-2)n$ for all m, n such that $n \geq m \geq 3$.

2.2.3 Graph Isomorphism

An overview of the state of the graph isomorphism problem through 1996 is presented in [22]. Solving the graph isomorphism problem is generally done using one of two distinct approaches. Both usually involve one or many vertex invariants to improve efficiency. The first approach finds isomorphism between two given graphs by directly finding the mapping between their vertices. The technique can employ approximation or exact methods. This brute force graph matching has applications in several contexts related to computer vision and pattern recognition contexts.

The second approach involves canonical labeling of the graphs in question. A function that produces a canonical label C(G) for a given graph G is defined. Canonical labeling works due to the fact that C(G) = C(H) if and only if graphs G and H are isomorphic. The most widely used canonical labeling algorithm is *nauty* [42].

nauty, standing for "No AUTomorphisms, Yes?", is considered to be one of the most powerful and practical methods available for solving graph isomorphism problems and has been labeled as the "world's fastest isomorphism testing program" [42]. Refer to Section 4.1 for an overview of *nauty*'s application to this dissertation.

2.3 Exhaustive Search Algorithm

As noted, this research originated with an exhaustive-search branch-and-bound algorithm proposed by [34]. An overview of it is presented to allow for understanding of the new work presented in Chapter 3. This algorithm makes use of Edmonds' Rotation Embedding Scheme, so discussion of that approach begins the review.

2.3.1 Edmonds' Rotational Embedding Scheme

Edmonds' Rotational Embedding Scheme was first formally introduced by Edmonds [18] in 1960 and then discussed in detail by Youngs [59] a few years later. The following is the formal statement of the Rotational Embedding Scheme as presented in [13] on pages 196-197.

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

For example, consider Figure 2.10 which gives a planar embedding of a graph. From this graph, the following are counterclockwise permutations associated with each vertex:

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

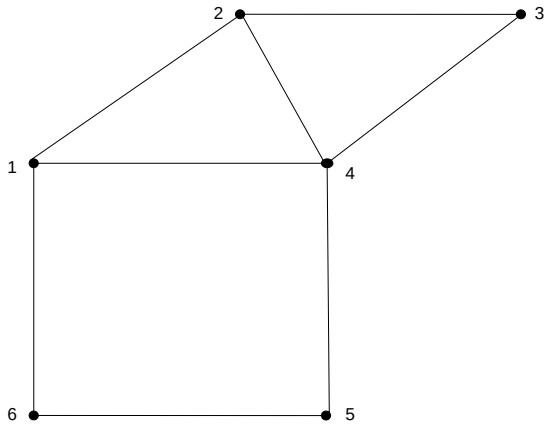


Figure 2.10: A Planar Embedding of a Graph

From these permutations the edges of the graph and the number of regions of the graph are determined. 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) Refer to permutation π_2 to determine which vertex follows 1. It is 4; therefore the second edge is (2,4).
3) Refer to permutation π_4 to determine which vertex follows 2. It is 1; therefore the third edge is (4,1).
4) Refer to permutation π_1 to determine which vertex follows 4. It is 2, which corresponds to the original edge (1,2), so the trace is finished.

The region traced is 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: every collection of vertex permutations corresponds to an

embedding on some surface. Given a set of permutations, the edges can be traced and the genus of the surface determined.

2.3.2 The Exhaustive Search Algorithm

The algorithm presented in [34] maps the solution space of the crossing number problem onto a tree. The tree is then searched for the crossing number with a branch-and-bound depth-first search (DFS). A DFS 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 process continues until the entire tree has been traversed. The branch-and-bound approach changes one small part of the DFS algorithm. When the cost to get to a vertex v exceeds the current optimal solution, the DFS algorithm does not traverse the subtree having vertex v as its root. This modification saves having to cover a section of the search space that is guaranteed to cost more than the current optimal solution. The branch-and-bound DFS algorithm is as follows:

Begin with the vertex set for the graph in question, and start to add edges. After each edge is added, determine, using the Rotational Embedding Scheme, whether the partial graph is still planar. Once no more edges can be added while keeping the partial graph planar, the mapping to the tree is performed.

At this stage a partial graph is the root of the tree. The first option is the many different ways to draw this graph. The root of the tree has a branch for each possible planar embedding. Now, select the first embedding, and begin to build the rest of its tree. Do this by considering laying down the next edge (which will go from vertex i to vertex j). The first option is through which one of the k regions to which vertex i is adjacent this edge should leave. These regions represent the next layer of the tree. Once a region is selected, the next option is to select which of the l edges of that region the edge will cross. Making this decision creates a cross vertex (degree 4), places an edge from vertex i to the cross vertex, and tries to lay the edge from the

cross vertex to vertex j. This may be possible directly, or it may require more cross vertices.

Lay down the remaining edges in a similar manner, and, when they are all laid down, a leaf in the tree is reached. At this point a cost for the current solution, which is the number of cross vertices, is known. This number of crossings becomes the new bound. Continue by backing up and trying other branches in the tree using this bound as a stopping criterion. Proceed in this fashion until the entire solution space is traversed.

In order to understand this approach, a walk through the algorithm is now presented with K_5 as an example. Figure 2.11 shows the vertex set for the graph with all the edges that can be added to the graph without making it nonplanar.

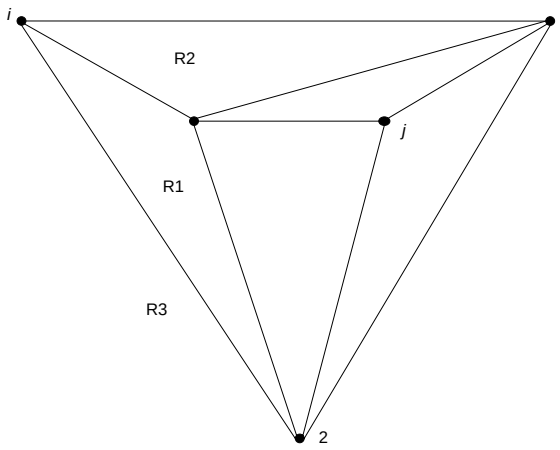


Figure 2.11: Planar Portion of K_5

At this stage the algorithm states that an attempt be made to take all remaining edges and lay them down one at a time. This is fairly simple in this case since there is only one edge left to be added, the one from vertex i to vertex j. Now three choices arise, and these are the three regions to which vertex i is adjacent (R1, R2, and R3). Selecting R1, which has 3 edges, leads to the conclusion that there is only one way to lay down edge ij since it cannot legally cross 2 of the edges because they are incident with i. The algorithm then calls for placing a cross vertex on this edge and

connecting an edge from i to the cross vertex as shown in Figure 2.12(a). The next step is determining if the edge from the cross vertex to j can be drawn while keeping the graph planar. In this case it can, and this edge, having one crossing, is complete. This solution is shown in Figure 2.12(b). The algorithm then backtracks and tries the other regions to which i is adjacent and finds that there are multiple ways to draw K_5 with one crossing. This exhaustive search algorithm is the foundation for the new work which is presented in Chapter 3.

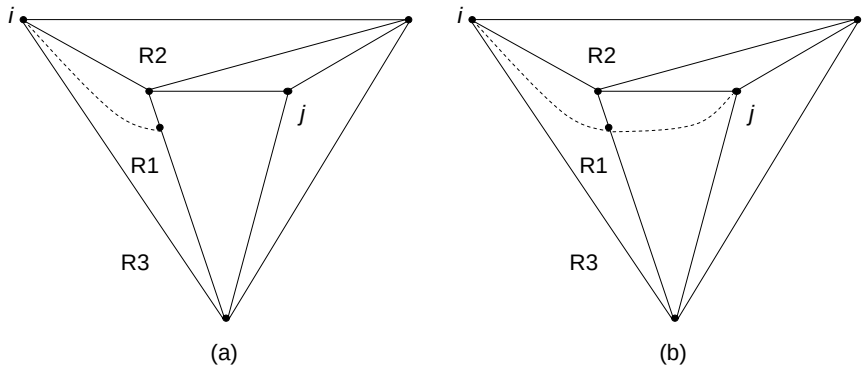


Figure 2.12: Laying Down an Edge for K_5

Chapter 3

Progression toward an Effective Algorithm

Work in the direction of growing minimal K_n from minimal K_{n-1} starts with the exhaustive search algorithm, presented in Section 2.3, modified such that the root of the search tree is minimal K_{n-1}. Guy [31] states that if Conjecture 2 is true, then it is false to conjecture that a minimal K_n always contains a minimal K_{n-1} for n odd and $n \geq 9$. If Conjecture 2 is true, it can be demonstrated that minimal K_n always contains a minimal K_{n-1} for n even, but no demonstration is available for n odd, specifically for $n = 9$ and probably for larger odd n. A formal proof showing minimal K_n does not contain minimal K_{n-1} for n odd has not been found in the literature nor has an example of a minimal K_9 that does not contain a minimal K_8 been located. In personal correspondence, Guy indicated that at one time there was a census of the minimal drawings of K_9. In the attempt to recover these graphs for evaluation it was learned that they have been lost over the years. It is also noted that the definition of isomorphism used for the missing census of minimal K_9 is different from the one employed for this body of work. These differences and the fact that Conjecture 2 is a conjecture leaves the door open for the exploration presented.

Although no example of a minimal K_9 that does not contain minimal K_8 has presented itself at this time, it is evident *via* mathematical evaluation that the possibility that such a drawing may theoretically exist cannot be disregarded. It is easily verified that removal of any vertex from minimal K_9 does not always produce a minimal K_8;

however, this does not show irrefutably that minimal K_9 does not contain a minimal K_8. Figure 3.1 illustrates a minimal K_9 having both minimal and non-minimal K_8 subgraphs. Figure 3.2(a) shows a minimal K_8 subgraph with 18 crossings, and Figure 3.2(b) shows a non-minimal K_8 subgraph with 22 crossings.

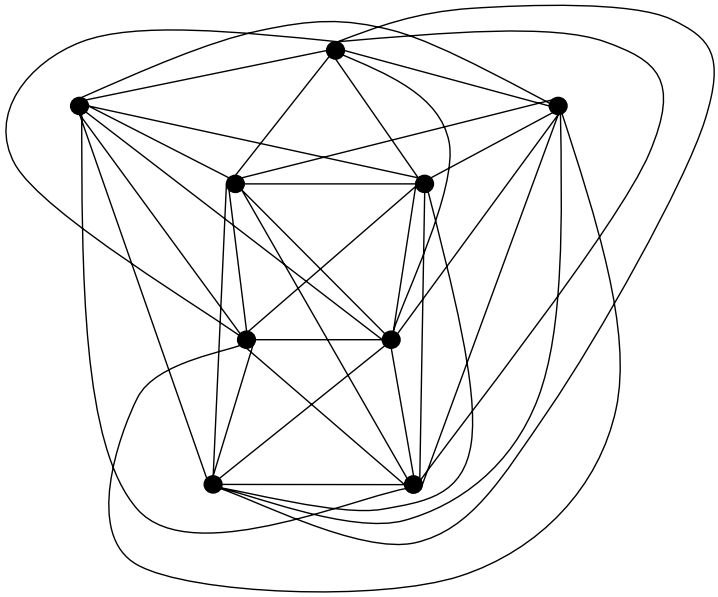


Figure 3.1: A Minimal K_9 with Minimal and Non-minimal K_8 Subgraphs

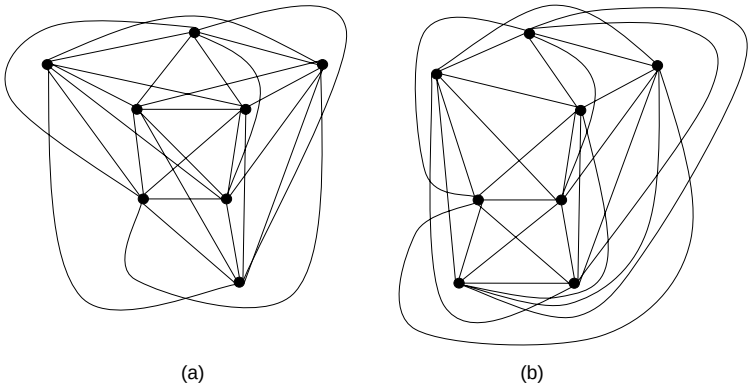


Figure 3.2: Minimal and Non-minimal K_8 Subgraphs of Minimal K_9

Recall that the responsibility of a vertex is defined by Guy [31] as the total number of crossings on all edges incident to that vertex. Given that each crossing

is the responsibility of four vertices, the total responsibility of a graph is $4c$, where c is the number of crossings. Start by considering a minimal K_7. A drawing of minimal K_7 contains a vertex with responsibility at least $\lceil \frac{4c}{7} \rceil = 6$. This is the average responsibility of all vertices in the graph. Removal of this vertex and all edges incident to it leaves a drawing of K_6 with at most $c - \lceil \frac{4c}{7} \rceil = \lfloor \frac{3c}{7} \rfloor = 3$ crossings. All vertices of minimal K_7 must have responsibility 6. If there existed one with larger responsibility, its removal would leave a K_6 with fewer than 3 crossings which is not possible. Thus removal of any vertex from minimal K_7 will leave a minimal K_6. A similar argument exists regarding minimal K_8 containing minimal K_7 subgraphs.

In the case of minimal K_9 with K_8 subgraphs, we would get a similar result if minimal K_9 contained 34 crossings, but it does not. Minimal K_9 has 36 crossings and by the responsibility argument, the average responsibility of its nine vertices is 16. Removal of a vertex with responsibility 16 will leave a K_8 with 20 crossings, which is nonminimal. Thus removal of any vertex from a minimal K_9 does not guarantee a minimal K_8 subgraph. Keep in mind that the vertex removed has the average responsibility of all vertices in the minimal K_9. Given that minimal K_8 has 18 crossings, it may be the case that all minimal K_9 have at least one vertex with responsibility 18. Removal of this one vertex would result in a minimal K_8 subgraph.

The question becomes: does there exist a minimal K_9 such that each of its nine K_8 subgraphs is not minimal? Specifically, the question is posed: does there exist a K_8 with 19 crossings that can generate a minimal K_9 with a maximum vertex responsibility of 17? All vertices of this minimal K_9 cannot have responsibility 16, or minimal K_9 would have 34 crossings, and removal of any vertex would result in nonminimal K_8 with 20 crossings. There would have to be at least 1 vertex with responsibility of 17. Removal of this vertex would result in a K_8 with 19 crossings. This would have to be the maximum responsibility since if a vertex with responsibility 18 existed in the graph, its removal would present a minimal K_8 subgraph and, of course, responsibility larger than 18 cannot exist or the crossings in minimal K_8 would be fewer than the 18 crossings of minimal K_8. Until this question is answered, it is

not confirmed whether every minimal K_9 contains a minimal K_8 or not. It is the case for all $n < 9$ that removal of any vertex from minimal K_n does leave a minimal K_{n-1} subgraph.

Given the possibility that not every minimal K_9 can be derived from minimal K_8, and that no proof to the contrary has presented itself, it is agreed, at this point, that some minimal K_n may be lost, specifically those created from nonminimal K_{n-1}, for n odd and $n \geq 9$ by using minimal K_{n-1} as the root of the search tree. It is assumed, for now, that it is possible that branches are pruned off the search tree that do not step through a minimal K_{n-1} on a path leading to minimal K_n. Refer to Chapter 5 for future work to further explore this topic. Given this concession, from this point forward, the discussion is based on using minimal K_{n-1} as the root graph of the search tree.

This chapter presents the new ideas that mold the exhaustive search algorithm, outlined in Section 2.3, into the foundation for an effective algorithm that can be used in growing minimal K_n from minimal K_{n-1}. Section 3.1 addresses parallel load balancing issues in the Tadjiev and Harris [52, 53] implementation. An overview of the new parallel implementation developed is presented in Section 3.2 for clear understanding of the three new search space reductions using this parallel scheme that are addressed in Section 3.3. The chapter concludes with a new technique, Star Analysis, discussed in Section 3.4 that steps away from the job-based parallel implementation of Section 3.2.

3.1 Parallel Load Balancing

In the initial parallel implementation of the exhaustive search algorithm overviewed in Section 2.3, Tadjiev and Harris [52, 53] implemented a basic static partitioning of the search tree across the processors used. Static partitioning is most beneficial when the problems being addressed have data sets that are completely defined prior to run-time and that are often divided among the processors, each computing its own results from its individual data set. A drawback of static partitioning is that when

one processor finishes working it must wait idly while the other processors complete their task. The work load is not evenly distributed across the processors. There is unrealized concurrent execution of the data in this case [61]. Tadjiev and Harris note this problem with the initial implementation they produced. Figure 3.3 illustrates one way an unbalanced search tree may be allocated to processors. Processors P2 and P3 will finish their allocated load and sit idle while P1 processes its larger load without the ability to share it.

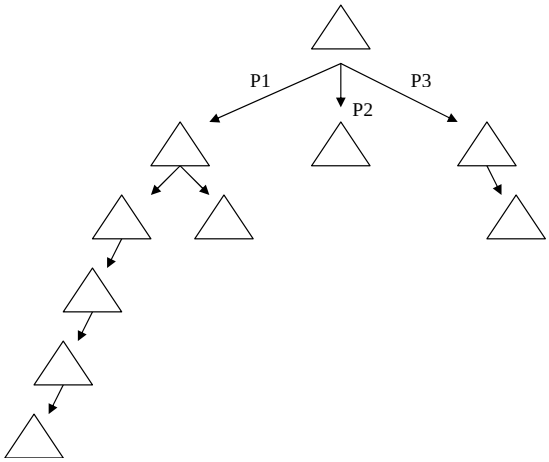


Figure 3.3: Unbalanced Work Load

One solution to the unbalanced load problem is dynamic load balancing. A work queue is one method of implementing dynamic load balancing and thereby ensuring a work load that is evenly distributed across many processors. This balancing can be centralized, residing with a master process, or decentralized, controlled by each slave, or a unified system that may be used combines the two . The work queue is especially useful in load balancing when dealing with irregular data structures such as unbalanced search trees.

In centralized load balancing, the tasks to be performed are held by the master and distributed to the slaves as they finish other tasks and become idle. Efficiency is maximized as slave idle time is minimized. One disadvantage of centralized load balancing is the possibility of a master task distribution bottleneck. A bottleneck

occurs when many slaves request tasks simultaneously but the master can issue only one task at a time. In decentralized load balancing, local processors keep their own work pools. This strategy has the benefit of avoiding a master task distribution bottleneck. Decentralized load balancing is similar to static partitioning in its apparent problems. In a unified system, the slaves may request work from each other or from a centralized master queue.

To alleviate the balancing problems noted by Tadjiev and Harris, a dynamic queuing system was created with a unified system employing both a master queue and individual slave queues. The slaves request work from a central queue and also maintain their own individual queue. The implementation, illustrated in Figure 3.4, has the master creating the first m jobs and placing them on the central queue (managed by the master). The master then sends one job to each slave processor. Each processor will create more jobs while processing. These jobs are kept in a local work queue managed by each individual slave. When a slave's local work queue reaches a user-defined limit indicating a large queue size, it sends the extra jobs to the master for placement in the central queue. If a slave depletes its local queue, it requests a new job from the master to supply its queue again so it may continue processing. If the master queue is empty and a slave requests a job to process, the master will request jobs from the other working slaves. The entire process terminates when all slaves are idle and the central queue is empty. Refer to [41], [60], and [61] for more details on this unified queuing system implementation.

3.2 Parallel Implementation Overview

As stated at the beginning of Chapter 3, a minimal K_{n-1} is the root of each search tree examined in the search for minimal K_n. A brief overview of how the parallel implementation works is presented now to facilitate clarity of the remaining sections of this chapter.

Figure 3.5 illustrates a minimal K_6 graph at the root of the search tree to be examined in a branch-and-bound depth-first search method in pursuit of minimal

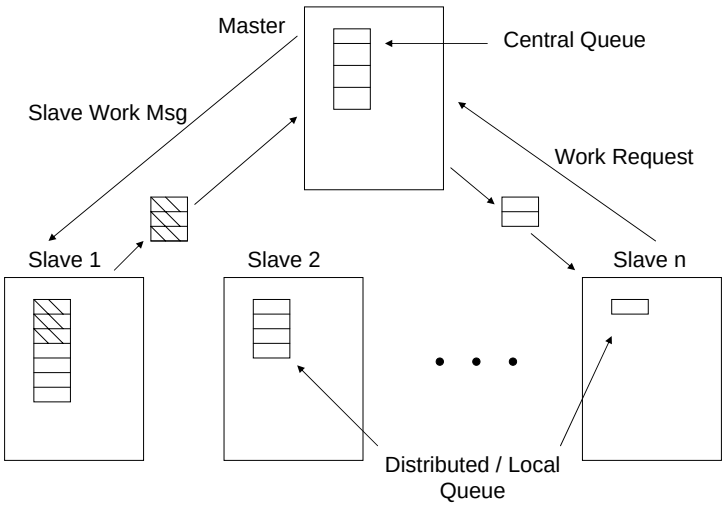


Figure 3.4: Parallel Work Queue

K_7. For this discussion, the terminology *native vertex* is introduced to represent a true vertex of a graph, not a crossing. A crossing vertex is introduced to planarize the graph each time an edge segment is placed unless it is to a native vertex on the boundary of a current region. An edge segment will terminate at a native vertex only if the vertex is the destination of the original edge. Crossing vertices are indexed starting one greater than the native vertex, indexed n, being added to the graph. In Figure 3.5 the native vertices of the graph K_6 are numbered 1 through 6, while the crossing vertices are numbered 8, 9 and 10. The index 7 is reserved for the new vertex being placed into the graph to generate K_7. A preprocessing step adds any direct connections from the new vertex n to any native vertices on the boundary of the region in which the new vertex is placed. Each region of the root graph is used as a possible location for the new vertex placement. Notice in Figure 3.5 that because vertex 7 was randomly placed in the region defined by vertices 2 and 4 and cross vertex 10, the edges (7,2) and (7,4) do not appear in the list of edges to add.

In generating minimal K_n, there is more than one minimal K_{n-1} root graph for $n \geq 7$. Refer to Chapter 4 for more details on how these root graphs are derived.

The parallel implementation makes use of one master (the manager) and several

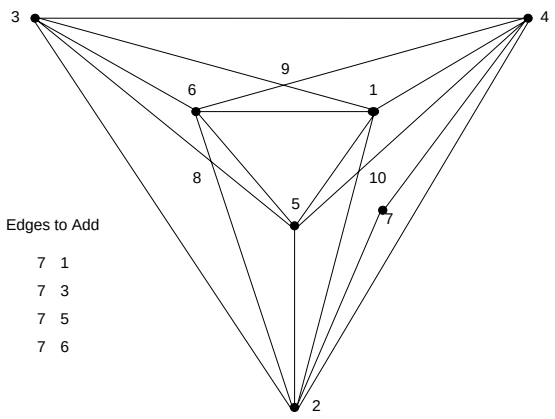


Figure 3.5: Minimal K_6 Prepared as the Root of the Search Tree

slave processors. When initiated, the program reads in each of the minimal K_{n-1} root graphs along with the edges that need to be added to each graph to generate K_n and begins the process by laying down one edge segment to each nonadjacent edge through each possible initial region for each new graph construction. Each edge segment laid down creates a new graph with a modified list of edges to add (less the newly laid down edge segment). Figure 3.6 illustrates the result of this process as applied to the K_6 in Figure 3.5, given that vertex 7 is placed in the region shown. Three possible segments can be laid down from vertex 7, thus three graphs are constructed. The edge segment laid down introduces a new crossing to the graph, cross vertex 11.

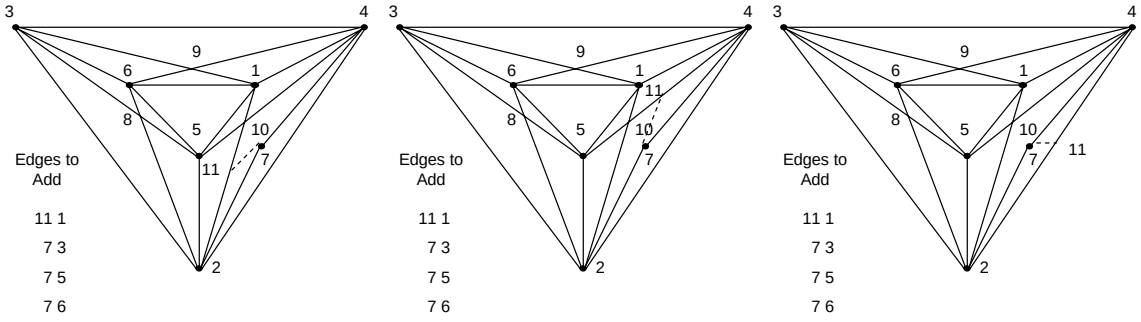


Figure 3.6: Stepping toward K_7 from Minimal K_6

The new graph, with its updated list of edges to add, is enqueued as a job in the master queue. Each slave processor requests an initial job, and the master sends a job

to each requesting slave. As the slaves create new jobs, they add them to their local queues. The master monitors requests from the slaves for more work and transfers jobs from slaves that have jobs enqueued locally to those in need of jobs to process.

Each graph in any job is treated as planar. As mentioned, a crossing vertex is introduced to planarize a graph after each edge segment is placed. If the number of crossing vertices exceeds the current minimum number of crossings found from previously examined graphs or Guy's conjectured solution, Conjecture 3, then the job is disposed of. Otherwise it is packaged, with the updated list of edges to add, as a new job and enqueued, and the search algorithm is employed again with this planar graph as the root of its own subtree.

When the list of edges to add is empty, minimal K_n has been constructed. The crossing vertices are eliminated, and crossings are reintroduced in their place. Figure 3.7 illustrates this process. The graph at the center top is minimal K_5 with the following preparatory additions: the one crossing is replaced with crossing vertex 7, vertex 6 is placed in one region, and direct connections are made to vertices 1, 2 and 5. The remaining edges to add are (6,3) and (6,4). Two new crossings are introduced while generating minimal K_6 and are labeled as crossing vertices 8 and 9, as illustrated in the lower left figure. The image in the lower right is the final minimal K_6 with the crossing vertices removed and replaced with crossings. The final product is a minimal K_6 with 3 crossings.

3.3 Search Space Reduction

A unified queuing system, as discussed in Section 3.1, was incorporated into a parallel implementation of the exhaustive search algorithm of Section 3.2. Exhaustive search algorithms are not efficient, and the search trees built become huge quickly. Lengthy runtime problems arise while creating complete graphs of order 8. Techniques for search space reduction are critical in achieving solutions for larger n. Three search space reduction techniques applied to the unified queuing parallel implementation are presented in Sections 3.3.1 through 3.3.3.

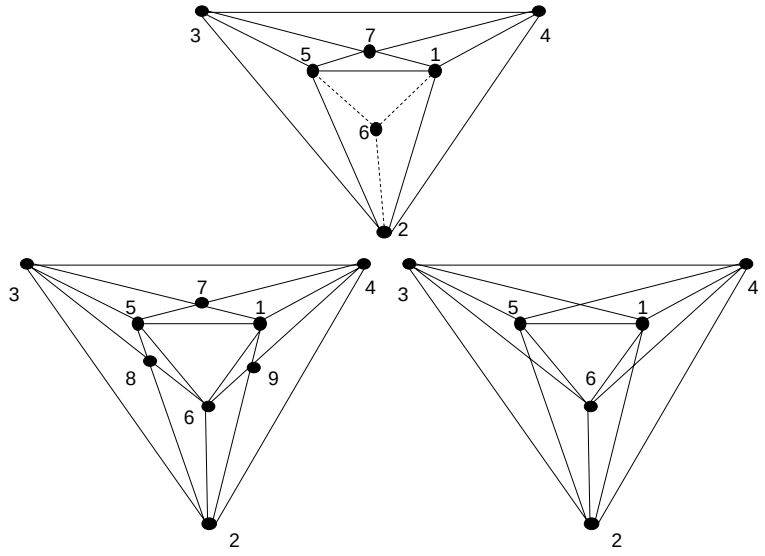


Figure 3.7: K_6 Constructed from K_5 with 1 Crossing Vertex

3.3.1 Order of Edge Placement

Given that the initial graph used in generating K_n is a complete graph of order $n-1$, the process of generating K_n is essentially that of placing a star graph $K_{1,n-1}$ centered at vertex n upon K_{n-1} with vertex n in a region of K_{n-1}. This leads to the following theorem.

Theorem 11 *In the process of generating graph descriptions of K_n from K_{n-1}, it does not matter in which order the edges incident to a new vertex n are laid down.*

Proof: *Let G be a minimal K_{n-1}. Let n be the vertex being placed into any one region of G. To create K_n, $n-1$ edges must be placed, each one connecting n to one of the $n-1$ vertices of K_{n-1}. All $n-1$ edges being added to G are adjacent edges since they are all incident with vertex n. By the definition of a good graph these edges cannot cross each other or meet at any point other than vertex n. Thus each edge is independent of every other. The placement of any one does not affect the placement of any other. Therefore, the order in which the edges are placed does not matter.* ⋈

The ability to disregard order of edge placement leads to a significant computational savings as compared to having to examine all permutations of edge placement.

3.3.2 Region Restriction

In generating graph descriptions for minimal K_n, the unified queue parallel implementation that adheres to the definition of a good drawing is inefficient because an edge path may visit an unnecessary region or return to an already visited region of the graph, sometimes repeatedly. This observation leads to the next search space reduction: Region Restriction.

Figure 3.8 illustrates four of many possibilities of a u-v path with the first edge segment crossing through region R1. Each of the drawings is a good drawing based on Definition 21. R1 has five edges, three of which are legal for uv to cross. Recall that each edge adjacent to R1 and not incident to vertex u will be used in generating all valid uv paths, so it is assured that all edge placements shown in Figure 3.8 will be generated. Figures 3.8(a) and (b) both illustrate placement of edge uv with one crossing, and Figures 3.8(c) and (d) have two and four crossings, respectively. Note that the new edge in both (c) and (d) exits R1 on the same edge and enters R3 on the same edge, but the edge in Figure 3.8(d) travels less efficiently by entering R4 twice, creating more crossings than the drawing shown in (c). Thus the drawing in Figure 3.8(d), though a good drawing, will not aid in producing a complete graph with a minimum number of crossings because the drawing in (c) has fewer crossings. This observation leads to Lemma 1.

Lemma 1 *Given a simple connected graph G with nonadjacent vertices u and v and at least two regions, restricting the edge uv from reentry into an already visited region of G does not eliminate any possible minimal graphs from being generated.*

Proof: *Let R be a region of graph G on whose boundary u lies. By definition of a region, exactly two edges of R are adjacent to u, and any region of G has at least three edges. When applying the rules for a good drawing, at least three non-adjacent edges*

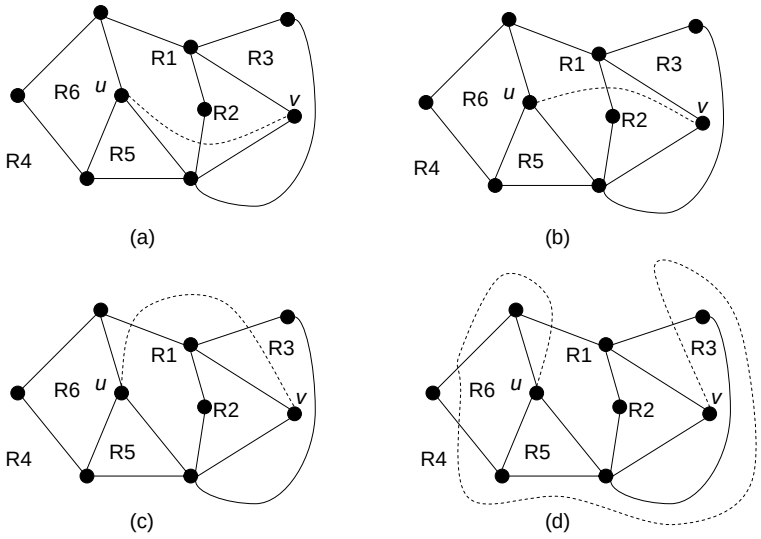


Figure 3.8: Good Drawings for Placement of Edge uv

(nae) are needed to exit, enter, and reenter a region. Thus, we need only consider regions with five or more edges.

Examining the case of the u-v path exiting, reenter, and reexiting R, let $f : u = u_0, i_1, p_1, e_1, ...e_j, p_2, i_2, p_3, e_m...e_n = v$ *represent the u-v path where:*

- p_1 *is the cross vertex created as the path exits R,*
- p_2 *is the cross vertex created as the path reenters R,*
- p_3 *is the cross vertex created as the path reexits R,*
- i_1 *and* i_2 *are the edges of the path internal to R,*
- *and* $e_1, ...e_j$ *and* $e_m, ...e_n$ *are the portions of the path external to R.*

The subpath $i_1, p_1, e_1, ...e_j, p_2, i_2, p_3$ *can be replaced with a single edge,* i_3*, thus shortening the path and reducing the number of crossings by at least two (represented by cross vertices* p_1 *and* p_2*). Figure 3.9 illustrates this substitution. Thus, restricting region reentry will eliminate only those graphs with nonminimal crossings.*

The argument is the same for any region of G, (not just those on whose boundary u lies), encountered on the u-v path as well as for more reentries and reexits of any

particular region.

⋈

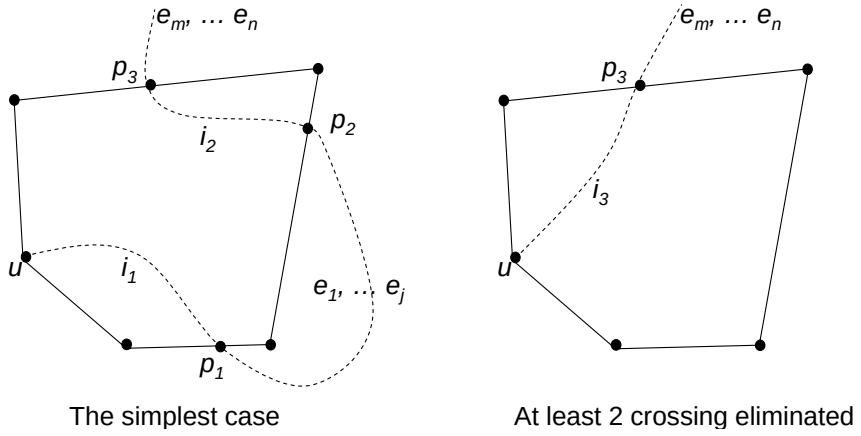


Figure 3.9: Subpath Replacement

A second observation is illustrated in Figure 3.10. It illustrates two u-v paths in graph G, both of which visit region R2. Figure 3.10 (b) generates more crossings than (a) in its path through region R2.

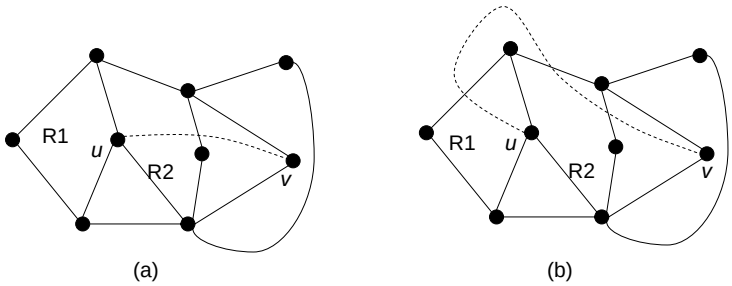


Figure 3.10: Two Possible Paths for Laying down Edge uv

Figure 3.11 shows the laying down of the six possible initial edge segments at the start of the uv path. After laying down the initial edge segment, any other edge segments on the uv path are restricted from entering regions R1, R2, and R3. With each additional edge segment placed, the relevant regions will join the list of restricted regions for any given edge.

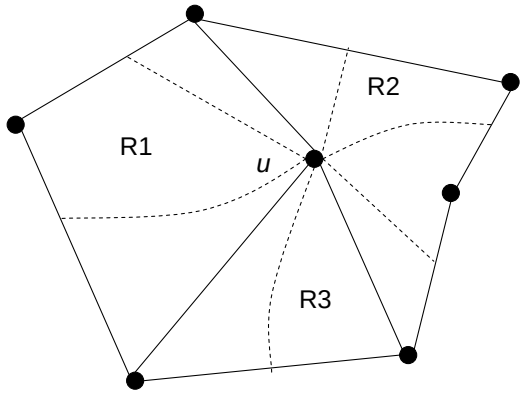


Figure 3.11: Initial Edge Segments from Vertex u to Other Nonadjacent Vertices

For the following lemma, define a *neighboring region* to be any region which shares a common edge with the region of focus.

Lemma 2 *Given a simple connected graph G with nonadjacent vertices u and v and at least two regions, restricting the edge uv from entering the neighboring regions of a region it crosses does not eliminate any possible minimal graphs from being generated.*

Proof: *Let R_1, ..., R_i be the regions of G on whose boundaries lie vertex u. We consider only the case in which vertex v does not lie on the boundary of R_1, ..., R_i (otherwise an edge with no crossings could connect u and v). Let p_n be the last vertex in the u-v path that lies on the boundary of one of R_1, R_2, ..., R_i. That is, the u-v path can be written $f : u = u_0, e_1, p_1, e_2, p_2, ...e_n, p_n, ...v$ where $p_1, p_2, ...p_n$ are all cross vertices created by the u-v path. The segment of the path from u to p_n can be replaced with one edge thus shortening the path and removing the crossings represented by $p_1, p_2, ...p_n$. Thus, restricting an edge path from entering neighboring regions of a region it crosses eliminates only nonminimal graphs.* ⋈

These two lemmas lead us to a new definition that places a tighter bound on the definition of a good drawing of a graph.

Definition 28 *Let*

$$f : u = u_0, e_1, u_1, e_2..., u_{k-1}, e_k, u_k = v$$

denote the u-v path laid down in graph G where e_i is an edge segment through one region of G. Let $H = G + f$. The graph H is a ***region restricted good graph*** *if every region of H has at most two vertices of f on its boundary.*

Definition 28 in conjunction with Lemmas 1 and 2 allows for the statement of the following theorem.

Theorem 12 *Employing the definition of a region restricted good drawing of a graph while creating minimal K_n from minimal K_{n-1} does not eliminate any possible minimal graphs from being generated.*

Modification to the graph generation algorithm in order to apply Region Restriction is minimal. Whenever an edge segment is laid down, its initiating vertex is used to add to, or create, a list of restricted regions for that edge. Region Restriction performed well when generating K_8, yet encountered runtime difficulties when generating K_9.

Table 3.1 shows a comparison of the number of partial edges laid down when building K_n for $5 \leq n \leq 8$ with and without Region Restriction. Results in this table were generated running on six parallel processors. For all n in the table, Region Restriction generated the same graphs with the minimum number of crossings as the non-restricted test results.

Vertices (n)	$\nu(K_n)$	**Good Graph**	**Restricted Good Graph**
5	1	3	3
6	3	203	71
7	9	1,498,775	19,979
8	18	*	46,697,854

Table 3.1: Region Restricted *versus* Good Graph Jobs Processed

A substantial search space reduction resulted, as can be seen by comparing results for $n = 7$. The search space is reduced by a factor of nearly 75. Running without Region Restriction, K_8 results were not achieved in over a week of runtime. With

Region Restriction, all jobs were fully processed in under four hours. This time reduced to just over two hours when eight processors were employed.

Figure 3.12 illustrates execution time using Region Restriction to run K_8 on six to sixteen processors.

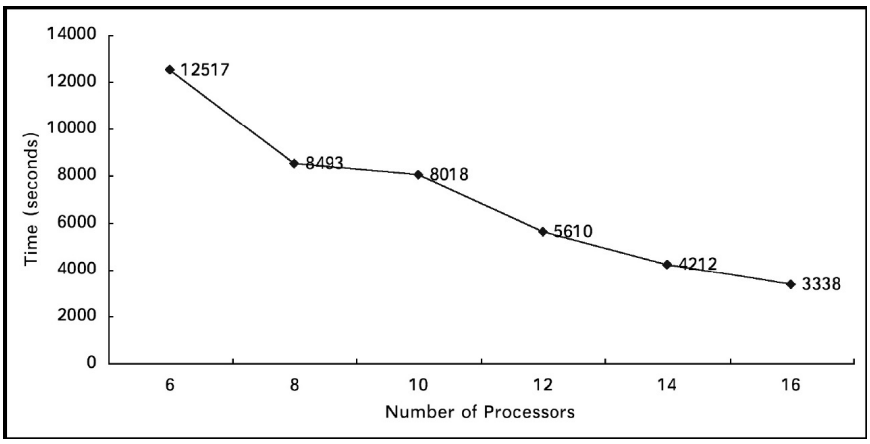


Figure 3.12: Region Restriction Execution Time for K_8 on P Processors

An attempt to generate minimal K_9 using Region Restriction was not successful given a runtime of one week on sixteen processors. This indicated that tighter restrictions on the search space were necessary. More pruning was imperative.

3.3.3 Radical Region Restriction

Radical region restriction is a greedy edge placement based on the analysis of the number of regions between vertices. When laying down all edges from vertex u to vertex v of graph G, only those edges that are of the minimum region separation from u to v are generated. For example, Figure 3.13 shows four of the ten possible uv edge placements for the given graph. Figure 3.13(a) and (b) are the only two possible uv edge placements that have the minimum number of regions (in this case two) separating u and v. The minimum number of separating regions, being two, indicates that only one crossing will be generated in each of these cases. Figure 3.13(c) and (d) show two other uv edge placements of the remaining eight, each of which separates u and v by three regions. A three region separation will introduce two crossings.

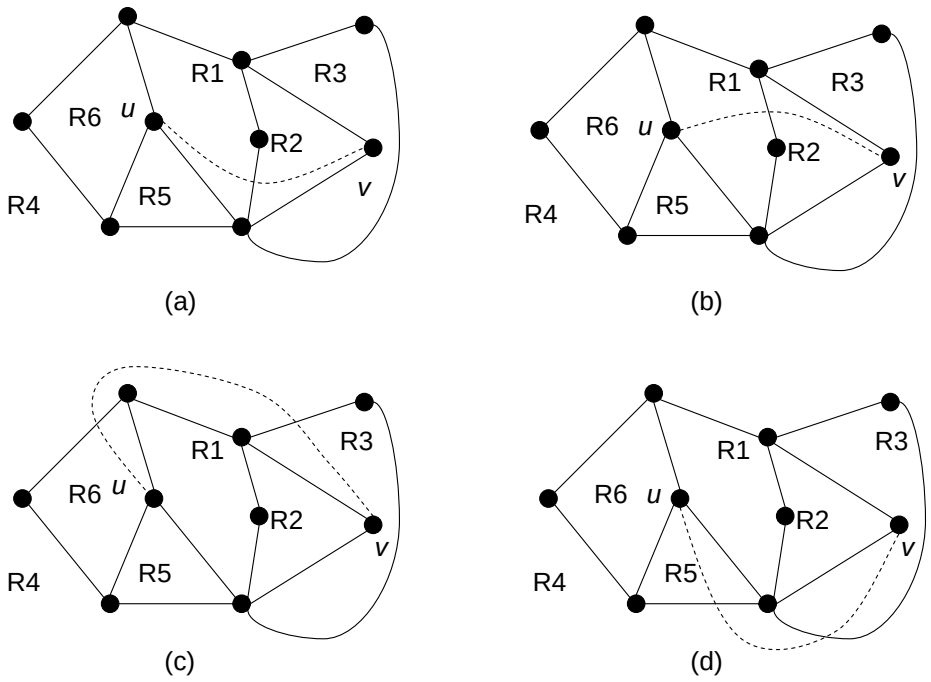


Figure 3.13: Placement of Edge uv

Radical Region Restriction determines, for each of the $n-1$ edges that need to be laid down on the given K_{n-1} graph the minimal number of regions that need to be crossed to lay down the complete edge. It then proceeds to follow all branches of the search tree that do not exceed this minimal number of region crossings for each edge needing to be laid down. This assures that all edges of minimum length are generated.

Theorem 13 *In generating minimal K_n from minimal K_{n-1}, the use of Radical Region Restriction does not eliminate any possible minimal graphs from being generated.*

Proof: *Radical Region Restriction dictates that all edges from vertex u to vertex v be the shortest length possible with the length of an edge determined by the number of regions crossed by the edge uv. Assume minimal K_n exists with f : a u-v path from u to v with nonminimal length. It is known from Theorem 11 that no two edges from new vertex n to any vertex $n-i$, $1 \leq i \leq n-1$, interfere with each other, so the nonminimal length of path f is not due to placement of the other $n-1$ edges. Path*

f will add at least one more crossing to K_n than a graph constructed with a uv edge of minimal length. Thus a contradiction arises and path f cannot exist in minimal K_n. ⋈

Radical Region Restriction examines each edge to be laid down individually as it is encountered as an edge to be added to the graph. The unified queuing parallel algorithm is employed. Each edge is laid down in segments with each new edge segment causing an update of the edge being laid down and a new job being created and enqueued. Those jobs that exceeded the minimal edge length for the given edge are abandoned as they would exceed the minimal number of crossings for the final graph being described. The initial search algorithm and region restriction both use a queue. With the implementation of Radical Region Restriction a stack is utilized to lay down an entire edge before moving on to the next one. This allows for graphs that exceed an edge length across regions to be abandoned prior to possible generation of partial edge segments for other edges.

Radical Region Restriction has been used to generate minimal K_n graphs for $5 \leq n \leq 10$ to date. It is a viable tool for larger n although as n increases by only a small amount it is found that long runtimes will be necessary to achieve results. Table 3.2 illustrates comparative runtimes of Region Restriction and Radical Region Restriction for minimal K_8 through minimal K_{10} on sixteen processors. The jump in timing for minimal K_9 to that of minimal K_{10} is due to the number of initial feeder graphs needed. To generate minimal K_9 only three minimal K_8 initial feeder graphs were necessary with forty regions each. Generation of minimal K_{10} required fourteen hundred and fifty three minimal K_9 feeder graphs, each with sixty-five regions.

Vertices (n)	$\nu(K_n)$	**Region Restriction**	**Radical Region Restriction**
8	18	2 hr 36 min	5 min
9	36	>> 1 week	8 min
10	60	unknown	≈ 1 week

Table 3.2: Radical *versus* Region Restriction Timing

Refer to Section 4.1 for discussion on initial feeder graph selection. Chapter 5 discusses some possible directions for reducing the number of regions required.

3.4 Star Analysis

The parallel implementation discussed in Section 3.2 with the search space reductions applied to it in Section 3.3 lays down one edge segment, repackages the graph definition as a job, and enqueues it again with the edge to be added being modified to reflect the new edge segment. This is done iteratively until there are no more edges to be added to the graph. All search tree paths are followed, with graph descriptions being built for each path as encountered. Not every graph is a minimal graph, but it must be built up to the point where either its number of crossings exceeds the current minimum number of crossings or it is complete. For those graphs that do exceed the current minimum number of crossings, the energy spent creating them is lost when they are discarded for possessing too many crossings.

Radical Region Restriction and its examination of region crossings for each complete edge, along with the fact that order of edge placement does not matter, leads to a new technique for edge placement, Star Analysis. This technique steps away from the iterative parallel edge segment placement and examines all the shortest complete edges that need to be placed. This is accomplished through the construction of a Distance Tree and its associated Path List, defined in Section, 3.4.1. Star Analysis finds all the minimal length star graphs centered at vertex n that will be laid down onto minimal K_{n-1}.

Star Analysis allows for evaluation of the minimum number of crossings for K_n from each region of new vertex placement in minimal K_{n-1} prior to graph construction. For every n, there is not always a minimal graph achieved with new vertex placement in every region. Creation of Path Lists prior to any graph construction allows for selectively choosing which graphs to construct. Selection for minimal K_n will focus on only those graphs with minimal crossings across all possible graphs examined. This technique, in addition to allowing for exploration of the graphs with minimal

crossings, also gives access to those that are locally minimal (graphs that have the minimal number of crossings given the region placement of the new vertex n into minimal K_{n-1} but are not globally minimal). These graphs are discarded as exceeding the minimal while being constructed according to the method discussed in Section 3.3.

With star analysis, parallel jobs representing partial edge segment placement no longer exist. A single Distance Tree and associated Path List description are created sequentially. Parallelism reduces time by having multiple Distance Tree/Path List combinations created, for each different region placement of new vertex n, simultaneously across multiple processors. The job queuing, message passing overhead of the previous techniques from Section 3.3 are no longer of issue.

Following the introduction of Distance Trees and Path Lists in Section 3.4.1, the selection of graphs for which to construct descriptions and their construction is presented in Section 3.4.2.

3.4.1 Distance Trees and Path Lists

Given a minimal K_{n-1} with r regions, the search for minimal K_n proceeds with the placement of the new native vertex n into each region of K_{n-1}. As stated previously, not all region placements will generate minimal K_n for all n. A Distance Tree is used to find the shortest region paths to each native vertex of K_{n-1} from the new native vertex n. A region path is the path an edge follows through regions of a graph.

A Distance Tree is a full (each level is complete) tree. It is built in a breadth-first fashion as follows: (1) The tree starts with a root node. (2) Each child node of the root is then added, creating the first level of the tree. (3) The children of the first level nodes are added, creating the second level, *etc*, until all levels have been created.

The root of a Distance Tree is the region in which the new native vertex n being added to the graph K_{n-1} is located. Each node of the tree represents a region being crossed in the path from vertex n to one of the $n-1$ other native vertices of K_n. Each branch of the tree represents a different complete path from the region containing vertex n to a region on whose boundary lies vertex $n-i$, $1 \leq i \leq n-1$.

The leaf, terminating point, of each branch is the last region crossed to reach native vertex $n - i$.

In employing breadth-first search to build the Distance Tree examination is not performed on graph G but on graph H whose vertices represent the regions of G. Adjacent vertices of H share an edge in G. The source region is the region in which vertex n is placed to generate K_n. Figure 3.14 illustrates a minimal K_6 with its region graph.

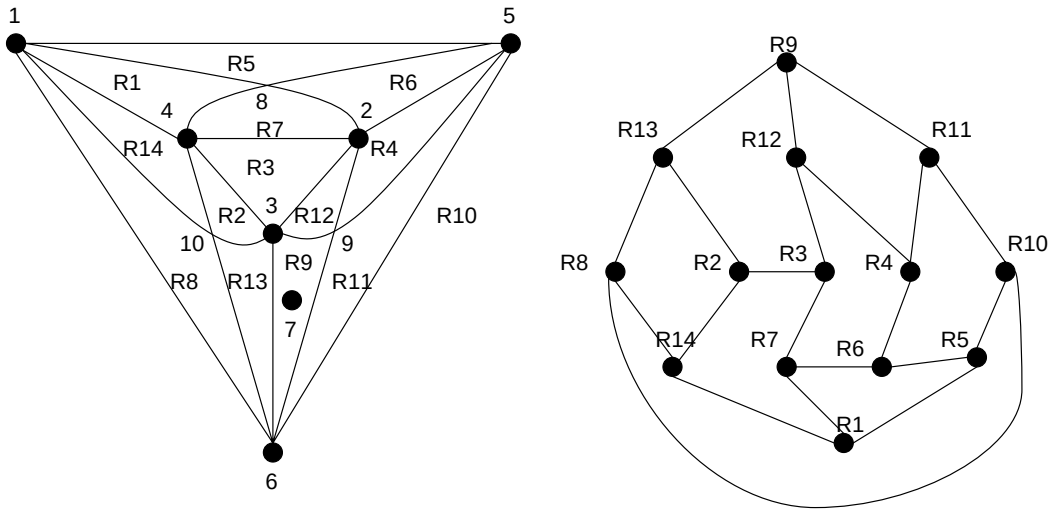


Figure 3.14: K_6 and its Region Graph Used for Breadth-First Search

Breadth-first search is one of the simplest algorithms for searching a graph. Given a graph H = (V,E) and a source vertex u, breadth-first search systematically computes the distance (smallest number of edges) from u to each reachable vertex. For any vertex v reachable from u, the shortest path(s) are located. More information related to breadth-first search and shortest-path algorithms is available in [14].

Because each vertex of H, and hence, each node of the Distance Tree, represents a region of G, the region restriction stated in Lemma 1 in Section 3.3.2 can be restated as follows:

Definition 29 *Given a Distance Tree as described, a* ***Distance Tree Region Restriction*** *states that once one complete level of a Distance Tree is constructed, no*

region on that level may be visited by any branch of the Distance Tree at any later level.

Lemma 1 restated in terms of Distance Tree Region Restriction becomes as follows:

Theorem 14 *In generating a Distance Tree for minimal K_n from minimal K_{n-1} as described, the use of the Distance Tree Region Restriction does not eliminate any possible minimal graphs from being generated.*

Once one complete level of the Distance Tree is constructed, all regions on that level are restricted from appearing in any branch of the tree at any later level.

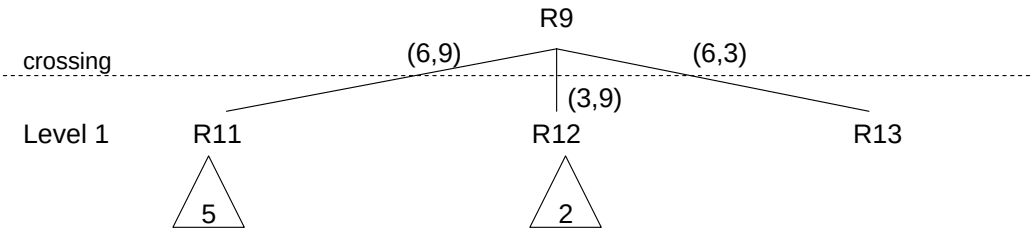


Figure 3.15: Beginning Distance Tree for One Minimal K_6

The following series of figures will illustrate the process of building the Distance Tree given minimal K_6 with native vertex 7 placement in region R9. Figure 3.14 shows the minimal K_6 with the vertices and regions labeled for this discussion. In viewing Figure 3.14 notice that two native vertices, 3 and 6, can be directly connected to vertex 7 without any crossings. These directly connected vertices are not included in the Distance Tree but are addressed in the Path List construction at the end of this section. Crossings will be generated in connecting vertex 7 to the remaining four vertices 1, 2, 4 and 5. Region R9 has three edges on its boundary, each possible as a crossing from vertex 7. Thus, three branches emanate from the root representing region R9. Figure 3.15 illustrates the root and the three initial branches. In addition to the region traveled to, the tree must remember the edge crossed in getting there

so the path can be recreated. This edge information is reflected in Figure 3.15 also. Notice that there are two direct connections possible through the level 1 regions R12 and R11 *via* the crossing edges (3,9) and (6,9) to vertices 2 and 5, respectively. Regions R11 and R12 are terminal leaves in the tree for their paths from Region R9 to vertices 2 and 5. Level 1 of the Distance Tree is complete.

With each level of the tree encountered, there is at least one crossing that will be reflected in the K_7 graph being considered. With vertices 2 and 5 on the boundaries of regions at Level 1, there will be two new crossings to add to the already existing three crossings of minimal K_6 in the hunt for minimal K_7. More levels, implying more crossings, are necessary to connect to vertices 1 and 4.

Figure 3.16 shows the addition of Level 2 to the Distance Tree construction being illustrated. Level 2 indicates that two crossings are necessary to get to any vertex on the boundary of a region at this level. Both vertex 1 and vertex 4 lie on the boundaries of regions that appear at level 2. Thus, for each of these vertices, 2 crossings will be necessary in connecting them to vertex 7. Adding these 4 crossings to the existing 5 tells us that a minimal K_7 with nine crossings can be constructed from the minimal K_6 shown in Figure 3.14 with vertex 7 placed in Region R9.

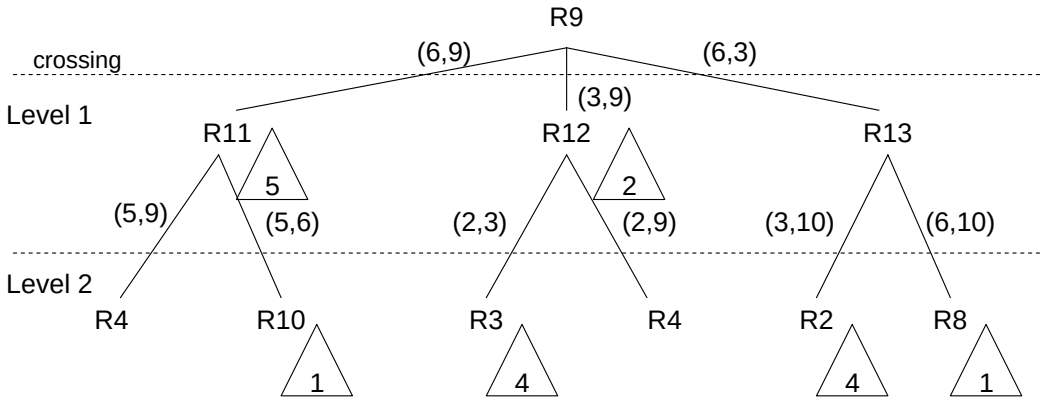


Figure 3.16: Distance Tree for One Minimal K_6

Figure 3.16 also illustrates that for this example, vertices 1 and 4 each have two paths possible from the root Region R9. Considering all possible path combinations,

there are four minimal K_7 graphs, each with nine crossings, described in this one Distance Tree.

Because edges of the Distance Tree correspond to crossings in the K_n construction, it would be possible to terminate the Distance Tree construction if that crossing count should exceed an upper bound, such as Guy's Conjecture 3. Or, the Distance Trees can be created and all local crossing numbers will be determined.

Returning to the issue of the Distance Tree Region Restriction introduced with Definition 29 and Theorem 14, refer to Figure 3.17, and note that in the search for the shortest path from vertex 7 to all other native vertices $n-i$, $1 \leq i \leq n-1$, region exploration takes place in an expanding fashion starting with the initial region (labeled "start" in this illustration) and working outward to neighboring regions until connections to all vertices $n-i$ are made. From the starting region, progress moves out to regions immediately adjacent from level to level. Level 2 regions are the neighbors of regions R11, R12, and R13 that have not yet been visited, namely regions R2, R3, R4, R8,and R10. Higher level regions can be determined similarly. As noted previously, once a region appears in a level, it cannot appear again in any other level.

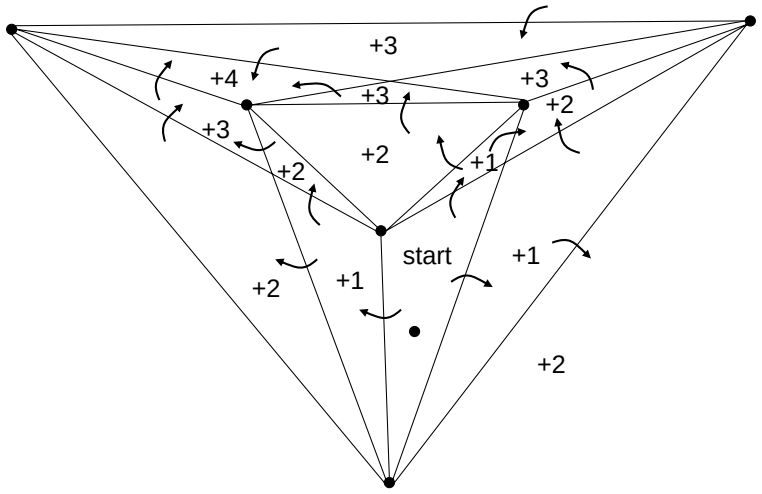


Figure 3.17: Outward Expanding Region Neighborhoods

The paths from vertex n to vertices $n-i$ represented in a given Distance Tree hold all the shortest path information for at least one K_n that is minimal, though

possibly only locally minimal. These paths are saved in Path List descriptions as described below for future analysis and minimal K_n graph description construction.

For each initial minimal K_{n-1} graph description and source region, the paths from the Distance Tree with the minimum number of crossings are saved as a unit in a Path List description. The Path Lists are available for future graph description construction or other analysis.

A Path List is a list of all the possible paths found in the Distance Tree. Recall that vertices on the boundary of the region in which vertex n was placed are not considered in the Distance Tree. Instead they are included in the Path List.

The Distance Tree in Figure 3.16 can been seen to reflect directly to the Path List in Figure 3.18. Reading the first line notice that the path starts in region R9 and crosses edge (6,3) to enter region R13. From region 13 edge (6,10) is crossed to region R8 from which a direct connection is made to vertex 1. As seen in the Path List, there are two shortest paths to vertices 1 and 4. Also note the two direct connection paths with no crossings for vertices 3 and 6.

R9	(6,3)	R13	(6,10)	R8	V1
R9	(9,6)	R11	(5,6)	R10	V1
R9	(3,9)	R12	V2		
R9	V3				
R9	(6,3)	R13	(10,3)	R2	V4
R9	(3,9)	R12	(3,2)	R3	V4
R9	(9,6)	R11	V5		
R9	V6				

Figure 3.18: Path List for Minimal K_7

A Path List description contains all the information necessary to describe a minimal, although possibly only locally minimal, K_n. If the goal is to generate global minimal K_n, across all K_n, those Path List descriptions with the minimum number

of crossings are used to generate graph descriptions for all the graphs possible to construct from all possible combinations of the paths in the Path List.

This section has explained the construction of the Distance Tree for K_n given an initially minimal K_{n-1}. All shortest paths from vertex n to each vertex $n-i$, $1 \leq i \leq n-1$, are described in the Path List for the given minimal K_{n-1} and region of vertex n placement. For any initial graph and region placement for a new vertex, the Path List description makes all locally minimal shortest paths available for analysis. The ability is now available to examine the regions supporting the same local minimum across, or within, graphs, path length statistics, *etc.*.

3.4.2 Graph Descriptions from Path Lists

Once all the Path List descriptions are created for a given input set, the selection of the set of graphs for which to build descriptions presents itself. For example, the input set might be all minimal K_7 with vertex placement in each possible region (there are 25 regions). Generating Path List files for this input set would supply all shortest path K_8 possible with the number of crossings seen if the graphs were to be constructed. Given the goal of constructing all minimal K_8, only those Path List files with the minimum number of crossings would be examined. In addition to the ability to look at just global minimal graphs, this technique allows for the building of graph descriptions of all shortest path K_8 graphs.

To illustrate the construction of graph descriptions, refer to Figure 3.18. Four minimal K_7 graphs can be constructed from this path list. The process of constructing one of the four is presented.

Figure 3.19 shows a minimal K_7 as constructed from the initial K_6 of Figure 3.14 with vertex 7 placed in region 9. The Path List of Figure 3.14 contains one path to each of vertices 2, 3, 5, and 6. The first paths to vertex 1 and vertex 4 from the path list are used with the only paths to vertices 2, 3, 5, and 6 to describe the first star graph that we place on K_6. Each $n-(n-i)$ path is laid down edge by edge, segment by segment onto minimal K_6 to arrive at this minimal K_7.

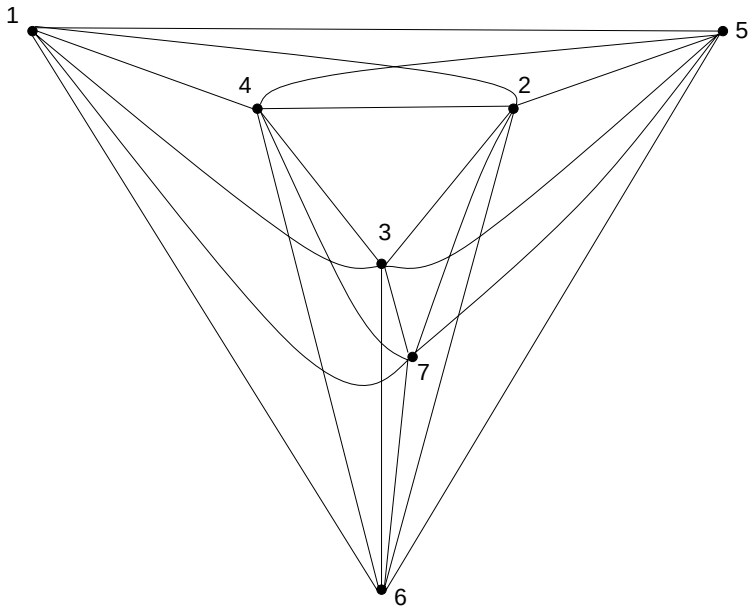


Figure 3.19: Minimal K_7 from Minimal K_6

At this time, it cannot be stated that a mathematical calculation of the number of ways the different $n-1$ paths may be combined definitively indicates the exact number of graphs defined by a Path List. It is possible that some combinations may lead to graphs that do not meet the definition of a good graph. An example of what may theoretically occur is as follows.

Figure 3.20 shows a portion of a hypothetical graph with possible Path Lists from vertex n to vertices 4 and 5. All combinations of paths are shown on the figure. All combinations of paths n-4 and n-5 cannot be used together. The two paths highlighted by stars (*) may not be placed together on the graph. If they were, they would cross, and the graph would fail to be good. Figure 3.21 illustrates examples of paths P1 and P4 used in valid n-4, n-5 path combinations.

Refer to [35] for further details on the implementation of Path Lists and Graph Description construction. With the development of Star Analysis complete, the discussion turns in Chapter 4 to the process of iteratively growing minimal K_n from minimal K_{n-1}.

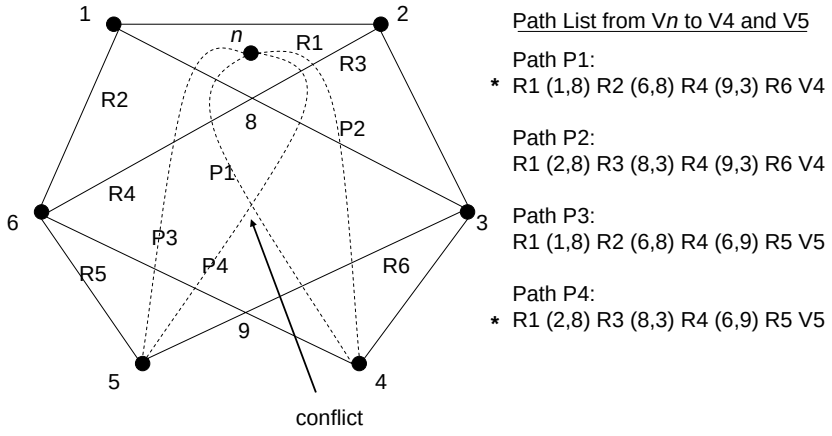


Figure 3.20: A Hypothetical Path List Conflict

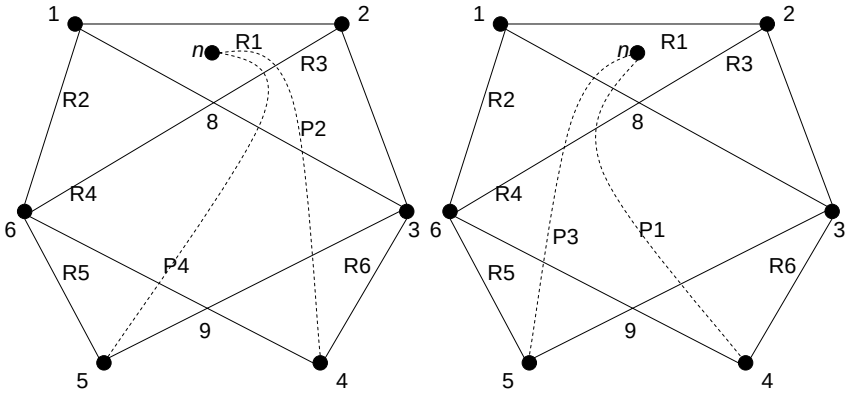


Figure 3.21: Two Different Non-conflicting Placement of Paths

Chapter 4

Construction of Minimal Complete Graphs

Using search space reductions described in Section 3.3 and the Star Analysis technique outlined in Section 3.4, an effective algorithm has been developed for building those minimal K_n derivable from one minimal K_{n-1}. This chapter describes a system that iteratively builds all minimal K_n from all minimal K_{n-1}, making use of only a subset of minimal K_n. Section 4.1 discusses the use of graph isomorphism to reduce the number of minimal K_{n-1} graphs that need to be used as root graphs in the growth process. An overview of the entire inductive process is covered in Section 4.2.

4.1 Isomorphic Families of Minimal K_n

If two graphs are isomorphic (refer to Definition 5) they differ only by the labeling of their vertices and possibly the physical view chosen for representation. The defining characteristics, or mathematical structures, of the graphs are identical. In the process of generating minimal K_n from minimal K_{n-1}, it is not necessary to use all minimal K_{n-1} graphs as root graphs. Only a single representative is needed from each isomorphic family, composed of all graphs that are isomorphic.

It is imperative to categorize graphs of order n into isomorphic families to be able to select the minimum number of necessary representatives to use as root graphs for the next iteration of minimal graph derivation. The software product *nauty*, written by Brendan McKay [42], is employed for isomorphism testing. A C program

was written to call *nauty* for building sets of isomorphic families. *nauty* compares two graphs and notifies the calling program if they are isomorphic or not. Each new graph in the list to be tested is compared to a current family representative and added to that family if it fits, otherwise a new family is created. After testing of all graphs, a set of root graphs is compiled by selecting one representative from each isomorphic family.

Graphs of order $n \leq 4$ all have one isomorphic family. Table 4.1 lists the number of isomorphic families of minimal K_n for $5 \leq n \leq 11$.

n	5	6	7	8	9	10
Graphs generated	12	4	76	20	4,560	56,618
Iso families	1	1	5	3	1,453	5,679

Table 4.1: Isomorphic Families of Minimal K_n

The need for exploiting isomorphic graphs is evident by examination of Table 4.1. For example, for $n = 9$, 4560 graphs were generated from minimal K_8 using Radical Region Restriction. The number of non-isomorphic graphs necessary for iteration to minimal K_{10} was 1453, a reduction of sixty-eight percent.

Further algorithmic exploitation of the topological structure of the root graphs such as symmetry, local and global, or region neighborhood analysis may allow for reduction of the current requirement that all regions of the root graph be used for new vertex placement. Refer to Chapter 5 for more discussion of this topic.

4.2 Growing Minimal K_n from Minimal K_{n-1}

Minimal K_4 was used as the initial starting point in growing minimal K_n inductively from minimal K_{n-1}. Minimal K_4 is a plane graph with no crossings, and there is only one minimal K_4 up to isomorphism. Figure 4.1 illustrates one labeling of the initial graph used to grow minimal K_5 with its associated region list representation. A region list is just one way a graph can be represented. Using Figure 4.1 think of standing inside region R1. Begin at vertex 1 with your left hand on edge (1,2). Walk around R1 keeping your left hand on the boundary of R1, and you will trace

the region description for R1: 1, 2, 4. Vertex 1 is adjacent to vertex 2 is adjacent to vertex 4 is adjacent to vertex 1. All regions of the graph are described in this fashion.

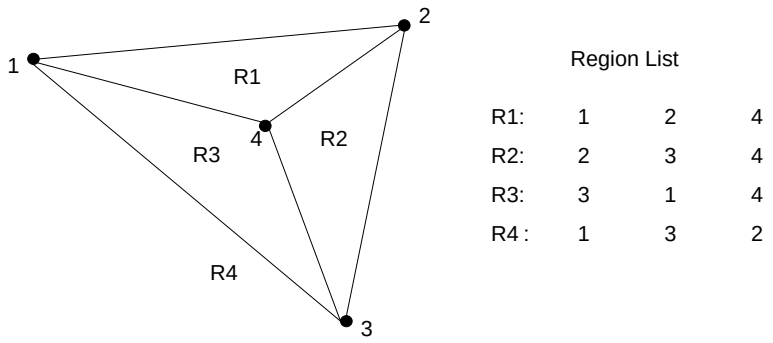


Figure 4.1: K_4 and its Region List Representation

Using minimal K_4 as the example, the entire process of generating the isomorphic family representatives of minimal K_5 necessary to generate minimal K_6 is illustrated in the following steps.

Step 1: Minimal K_4, the feeder graph, represented as a region list, is presented to the Input Generator. The Input Generator adds to the region list a list of edges to add to the graph necessary to construct minimal K_5. Each region in K_4 is used as an initial placement location for vertex 5. Thus 4 different configurations are presented to the Star Analysis module. Figure 4.2 illustrates these four initial configurations.

Step 2: For each of the four configurations in Figure 4.2, the Star Analysis Module is instantiated. The resulting Distance Tree and Path List for the configuration placing vertex 5 in R1 is illustrated in Figure 4.3.

In moving from Figure 4.2 to Figure 4.3, notice that the edges to add, (5,1), (5,2), and (5,4), are all direct connections from within region R1. They are not included in the Distance Tree because they will not produce any crossings in the final graph descriptions. To place edge (5,3), there are three distinct regions sharing boundary edges with region R1: R2 *via* edge (2,4), R3 *via* edge (1,4) and R4 *via* edge (1,2). The Distance Tree reflects these three paths. From each of these three regions there is a

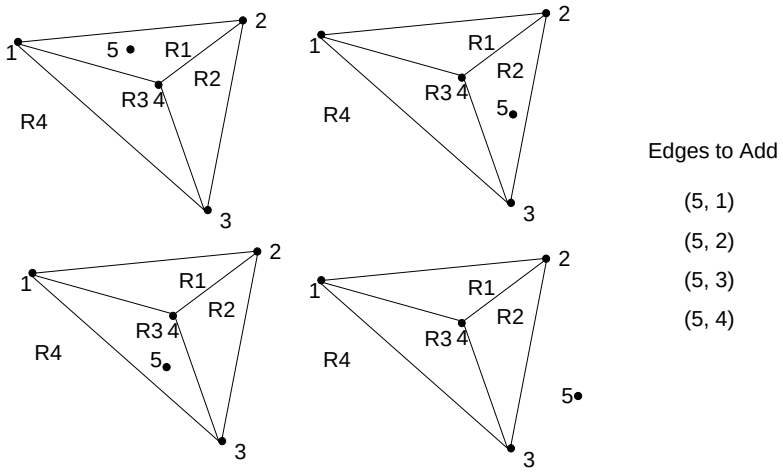


Figure 4.2: K_4 with Vertex 5 Placement and Edges to Add

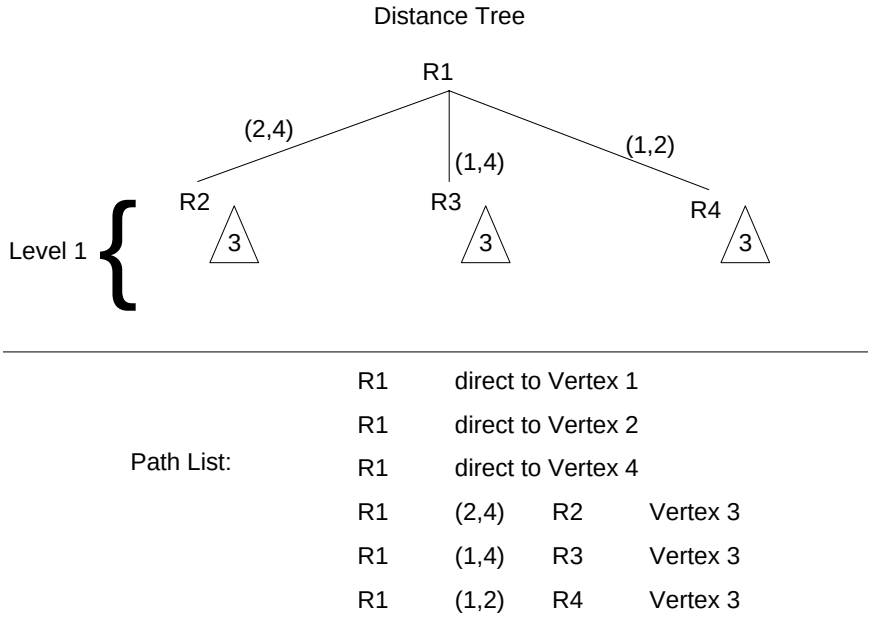


Figure 4.3: Star Analysis: K_4 (with Vertex 5 in Region 1) to K_5

direct connection to vertex 3. Thus, there are three distinct paths with one crossing each from region R1 to vertex 3. Each path is reflected in the Distance Tree. From the Distance Tree it is determined that exactly one crossing is required in growing the initial configuration, minimal K_4 with vertex 5 in region R1, to a K_5. No more

than (in this case exactly) three graphs will achieve this conclusion.

The Path List reflects all paths from vertex 5 in region R1 to each of the four vertices to which it must connect. The directly connected vertices are included here as well as all possible shortest paths to other vertices (only vertex 3 for this example).

Step 3: Selection of graphs for construction follows the Path List generation. For the example of K_4 to minimal K_5, recall from Figure 4.2 that there are 4 configurations to examine. Each of these configurations generates a Distance Tree and Path List that is similar to the one shown in Figure 4.3, each representing one crossing and three graphs to achieve minimal K_5. So four Path List files are created, each with minimal K_5 (one crossing) from minimal K_4 descriptions. In this case all Path Lists represent minimal K_5, so all Path List files are examined for construction of minimal K_5. Twelve graph descriptions will be created (three from each path list), each representing a minimal K_5. The graph description construction process lays down all K_{n-1} star graphs centered at vertex n created from the Path List description into the appropriate minimal K_{n-1}. The process splits regions as edge segments are placed and the star edges are laid down from each $n-i$, $1 \leq i \leq n-1$, vertex to vertex n.

In the general case of constructing minimal K_n from minimal K_{n-1}, it is common for n even for the Star Analysis Module to locate an assortment of crossings, thus generating locally minimal K_n based on the region in which vertex n is placed. For example, Figure 4.4 shows a minimal K_7 with four regions labeled.

A portion of the Path Lists generated for K_8 created from the K_7 of Figure 4.4 is shown in Figure 4.5. The file extension following the underscore is interpreted as: *isomorphic family.region number.crossings*. This listing indicates that Vertex 8 placement in region 10 leads to generation of locally minimal K_8 with 19 crossings. Vertex 8 placement in region 11 leads to minimal K_8 with 20 crossings, *etc.* In the search for minimal K_8, graph generation is performed on those Path List files that indicate globally minimal K_8, or 18 crossings, constructed by placing the new vertex in region 12 in Figure 4.4.

In general, minimal K_n will be found by examining those Path Lists that represent

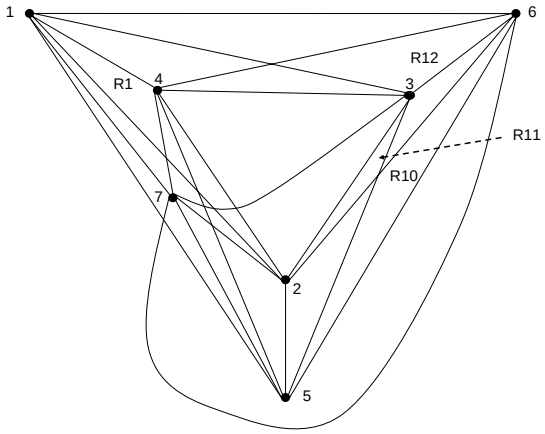


Figure 4.4: A Minimal K_7

```
467 Mar 30 12:38 PathK7K8_1.10.19
712 Mar 30 12:38 PathK7K8_1.11.20
533 Mar 30 12:38 PathK7K8_1.1.21
499 Mar 30 12:38 PathK7K8_1.12.18
.
.
.
```

Figure 4.5: Path Lists with Locally Minimal K_8

min(min K_n for region i in initial graph j), $1 \leq i \leq r$, where r is the number of regions in the initial root graph(s). This is a substantial savings over the search space reduction techniques of Section 3.3. In those cases, energy was expended growing all graphs until they exceeded the current minimum value found. When a graph exceeded the current minimum it was discarded. Not only was energy spent growing a locally minimum graph, but the graph was discarded prior to completion so it was not available for future analysis of the underlying structure of those nonminimal graphs.

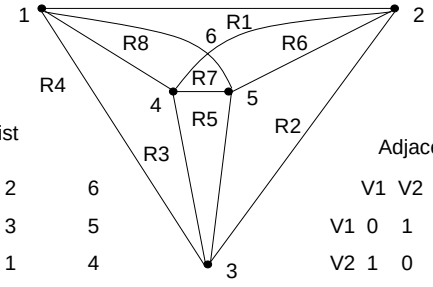
To date, for n odd, $5 \leq n \leq 9$, all Path Lists represent minimal graphs. This means that vertex n placement in any region of K_{n-1}, for n odd, generates at least

one globally minimal K_n. This phenomenon is discussed further in Chapter 5.

Returning to the example of preparing minimal K_5 for generating minimal K_6, four Path List descriptions of K_5, with a total of twelve graph descriptions among them, are ready for building.

Step 4: Generation of graph descriptions is, as stated above, a selective process based on the interest in minimal graphs or locally minimal graphs. In this example, all K_5 Path Lists indicate minimal graphs, so all are evaluated, and twelve minimal K_5 graph descriptions are constructed.

Step 5: A format conversion is required for isomorphic testing. *nauty* requires that graphs be represented as adjacency matrices. Figure 4.6 illustrates both region list and adjacency matrix for minimal K_5. Notice that the adjacency matrix has a 1 for each (row, column) vertex pair that is adjacent in the region list, otherwise the entry is a 0.



Region List

R1:	1	2	6
R2:	2	3	5
R3:	3	1	4
R4 :	1	3	2
R5:	3	4	5
R6:	2	5	6
R7:	4	6	5
R8:	1	6	4

Adjacency Matrix

	V1	V2	V3	V4	V5	V6
V1	0	1	1	1	0	1
V2	1	0	1	0	1	1
V3	1	1	0	1	1	0
V4	1	0	1	0	1	1
V5	0	1	1	1	0	1
V6	1	1	0	1	1	0

Figure 4.6: K_5 Region List and Associated Adjacency Matrix

Step 6: Isomorphic testing is performed on all minimal K_5 graphs. Isomorphism testing generates a listing of all the isomorphic families for the graphs tested and enumerates all the members in each family. Figure 4.7 shows the isomorphic testing results for the twelve minimal K_5 graphs tested. Notice that in the case of minimal K_5 there is only one isomorphic family, so only one representative is necessary for

File: K5Iso

Graphs are labeled starting at 1
Number of graphs in this file is 12
Number of total vertices is 6

Family of graph #1: 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12,
Total number of graphs in this family is 12

Total number of families in this file is 1

Figure 4.7: Isomorphic Test Results on K_5

moving ahead to grow minimal K_6.

Actually, all four initial K_4 configurations are isomorphic, and all regions of K_4 are isomorphic. Thus, placement of vertex 5 in any region in any configuration will always generate a graph description isomorphic to any other placement. This represents further analysis that may be performed at the Step 1 - Step 2 interface to see if algorithmic topological structure analysis of the graphs being processed may be exploited to result in the testing of fewer regions. See Chapter 5 for more discussion on this.

Step 7: Isomorphic family representative selection is performed to extract one representative from each isomorphic family from minimal K_n to use as a feeder graph in generating minimal K_{n+1}. The representative index is determined from the isomorphic testing results and the associated graph description (region list) is extracted from the graph descriptions generated in Step 4. All the isomorphic family representative graph descriptions are consolidated into one file for ease of handling, and the process begins again at Step 1 using minimal K_n to step up to minimal K_{n+1}.

In summary, Figure 4.8 illustrates the seven-step process just outlined.

It is noted that at no step in this process does the system know that it is working on complete graphs. The only defining factor indicating complete graphs is the initial feeder graphs used along with the list of edges to add. Thus, it is anticipated that

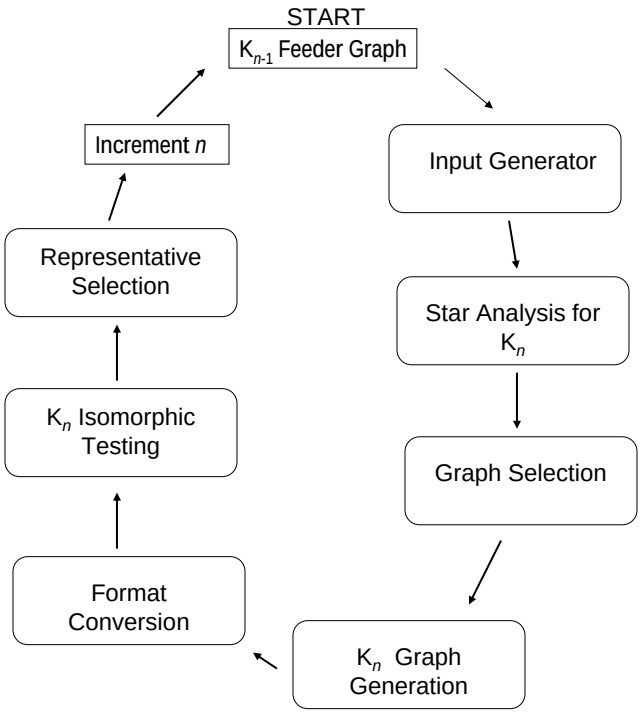


Figure 4.8: Iterative Process to Grow Minimal K_n

this process will seamlessly allow for examination of other graph families with no modification other than the initial graph description(s) and the list of edges to add. A specific example of complete bipartite graph growth from $K_{2,3}$ to $K_{3,3}$ and $K_{2,4}$ is presented in Chapter 5.

Chapter 5

Summary and Open Questions

This dissertation presents an effective framework for iteratively generating minimal K_n from minimal K_{n-1}. The process allows for complete enumeration of all minimal K_n that can be derived from minimal K_{n-1}, including those graphs that are only locally minimal. A graph is locally minimal if it reflects the smallest number of crossings possible for a given region placement of vertex n in minimal K_{n-1}. All graphs can be easily classified into isomorphic families so one representative may be evaluated instead of many. This classification also makes the process of growing the next complete graph of order $n+1$ efficient since only one family representative from each isomorphic family is necessary for complete results. Prior to this work, the largest confirmed minimal crossing number was for complete graphs of order 10. All non-isomorphic graphs for larger n will be visually available for each increasing value of n (we include drawings of one representative of both minimal K_{10} and K_{11} later in this chapter). No prior drawings of graphs larger than K_9 have been located.

With access now available to all minimal K_n given minimal K_{n-1} graph descriptions, exploration of the underlying structure of the graphs is possible. Access allows for substructure isomorphic analysis and localized region neighborhood analysis, both of which will lead to yet more efficient iteration of K_n for increasing n. A multitude of questions may now be explored that were previously unattainable without a working set of data to analyze.

Though this work focuses on minimal complete graphs, the only defining elements directly related to complete graphs are the initial graph descriptions and the edges

that need to be added to those graph descriptions to iterate to the next higher n. Later in this chapter an illustration of using this process on complete bipartite graphs demonstrates its applicability to an assortment of graph families other than complete graphs. Enumerated data for other graph families will allow for analysis of their underlying structure also.

Section 5.1 briefly summarizes the new algorithm development presented in previous chapters. The questions that have arisen out of the data now available for analysis, and some new conjectures regarding complete graphs are discussed in Section 5.2.

5.1 A Synopsis of New Developments Presented and Conclusions Drawn

Prior to development of an iterative system for growing minimal K_n for increasing n, it was necessary to develop a practical algorithm for migrating to K_n that exhibited the ability to keep functioning as n grew. The starting point for this development was a theoretically sound, yet realistically impractical, exhaustive search algorithm [34] for determining the crossing number of K_n, discussed in Section 2.3. Its impracticality comes from the very nature of exhaustive search algorithms. The required search space balloons, even for very small n, rapidly making it functionally useless. However, it was a solid foundation from which to start exploring ways to develop an effective iterative process with an existing parallel implementation available to analyze [52, 53]. A modification to the initial feeder graphs used by the algorithm for the purposes of growing all minimal K_n from minimal K_{n-1} is to always use a minimal K_{n-1} as the input to the algorithm. Making this decision raises the question of whether the process presented finds all minimal K_n or if some are lost, specifically those, if any exist, that may be generated from non-minimal K_{n-1}. It is acknowledged at this time that there is a theoretical possibility that minimal K_9 may exist that cannot be grown from minimal K_8, and this may also be the case for larger odd n.

The first improvement, a load balancing framework for the parallel system, aided

in more fully utilizing the concurrent nature of the parallel system being used. A unified load balancing system, described in Section 3.1, helped balance the work load across processors for better exploitation of all processors over time. This improvement made the parallel system more efficient, but the search space for the exhaustive search algorithm running on the parallel system had to be reduced drastically.

Three search space reduction techniques were presented in Section 3.3. The first deals with removing branches of the search space that lay down edges being added to the initial feeder graph in all permutations possible. Theorem 11 proves that order of edge placement has no bearing on the resulting graphs generated. It was no longer necessary to examine all order of edge placement to see if different graphs resulted. As an example of savings, consider one minimal K_8. Eight edges need to be added to new vertex 9 to grow any K_9. There are 8!, or 40,320, different edge placement permutations that would be necessary if order of edge placement did matter.

The second search space reduction presented is Region Restriction. Region Restriction restricts a new edge being laid down from revisiting a previously visited region of the graph being built upon and restricts visiting regions incident to the initial edge vertex or any crossing created by the edge. Both of these restrictions, shown in Lemmas 1 and 2 not to introduce a loss of minimal K_n allow for Definition 28, which places a tighter bound on the definition of a good graph for the purposes of generating minimal K_n. Region Restriction did allow for results for greater n than without its use, but runtimes for generating K_9 were considered too slow for practical application.

The final search space reduction technique that applied to the exhaustive search algorithm is Radical Region Restriction. Given that the order of edge placement does not matter, because the edges do not interfere with each other, it makes sense to generate shortest edge paths from new vertex n to the other $n-1$ edges of the initial minimal K_{n-1}. The use of Radical Region Restriction is proven with Theorem 13 not to lose minimal K_n . Radical Region Restriction has proven to be a viable algorithm for generating minimal K_n iteratively for larger n than possible before. To date it has

been used to generate up to K_{10}. Examining optimization of the algorithm related to how the shortest edge paths could be constructed lead to the Star Analysis technique which steps away from the parallel implementation utilized up to this point. With the development of Star Analysis, the Radical Region Restriction algorithm became obsolete, although without its development Star Analysis would most likely not have presented itself.

Star Analysis makes the parallel processing of the previous search space reduction techniques obsolete. Parallel running of instances of the Star Analysis module is exploited to take advantage of the ability to run disjoint instances (different region placement for vertex n) simultaneously. But the overhead related to job creation and queuing, dequeuing and slave sharing issues, as well as message passing, is removed.

Star Analysis looks at the global picture of growing minimal K_n from minimal K_{n-1} by focusing on optimizing the enumeration of all star graphs, $K_{1,n-1}$, centered at vertex n that can be placed into initial minimal K_{n-1} feeder graphs with vertex n placed in each region of K_{n-1}. Figure 5.1 illustrates the two star graphs placed into a minimal K_6 in the indicated region that each generate a minimal K_7. The dashed lines, in each image, highlight the star graph $K_{1,6}$, centered at vertex 7, placed into the underlying minimal K_6.

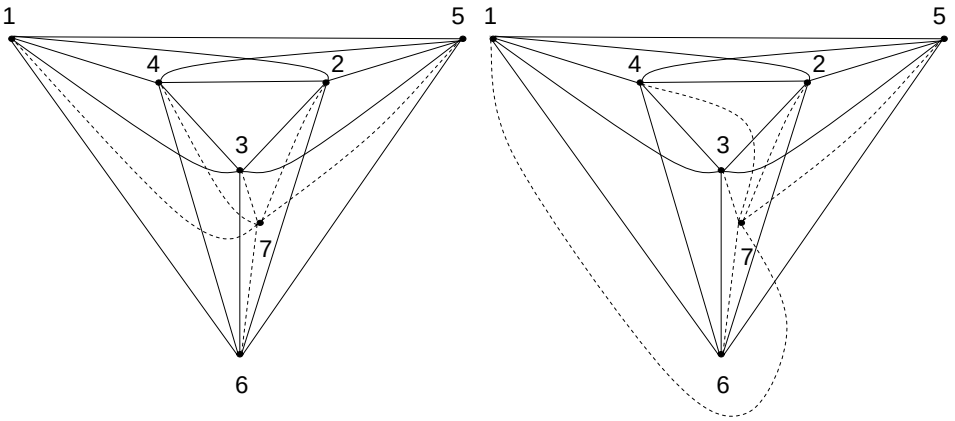


Figure 5.1: Star Graph Overlays on Minimal K_6 to Construct Minimal K_7

Two new constructs are introduced, Distance Trees and Path List descriptions,

to facilitate enumeration of all possible star graphs, with each edge of the star being of minimal length, that may be placed into initial minimal K_n feeder graphs. This process allows for enumeration of all minimal K_n, local and global, grown from minimal K_{n-1}. Local minimums relate to the smallest number of crossings possible given a specific region placement for vertex n in minimal K_{n-1}. The global minimum represents the actual crossing number for K_n.

Path List descriptions are generated for each vertex n placement in every region of each initial feeder graph, minimal K_{n-1}. The generation of these Path Lists is enough to determine the crossing number of K_n generated from minimal K_{n-1}. Graph descriptions, allowing for visually drawing the desired graphs and for iterating to minimal K_{n+1}, may be selectively chosen for construction based on the crossings of interest (local or global). This is a huge savings over all preceding implementations. Previously, construction was started on each graph as edge placement was tried. A graph was disposed of if the number of crossings exceeded the currently known minimum but had to be constructed to that point. Only the globally minimal K_n are completely generated and saved due to the overhead involved in building all graphs. This process was expensive even with the use of parallel processing. The separation of creating Path List descriptions from that of generating graph descriptions allows for holding the data for global and local minimal K_n and the selection of graphs for which to generate descriptions. For iterating to minimal K_{n+1}, only globally minimal K_n graph descriptions are necessary. The other graph descriptions will be of use for structural analysis of locally minimal K_n in an effort to determine if it is possible, and how, to reduce the current requirement of having to place vertex n into each region of minimal K_{n-1}.

To date Path List descriptions have been generated for minimal K_n, $5 \leq n \leq 10$. In generating these descriptions, it was discovered that in each case of growing minimal K_n, for n odd, each and every region placement of vertex n into minimal K_{n-1} generates at least one minimal K_n. Review of the data generated by all previous search space reduction techniques mentioned above verified this result. The Path List

description filenames made the result standout and be noticed immediately. Table 5.1 shows the number of initial feeder graphs (non-isomorphic family representatives) with their associated number of regions and the number of regions that generated a minimal K_{n+1}. These results lead to Conjecture 4. A multitude of questions arise from this observation. See Section 5.2 for a list.

n	5	6	7	8	9
family reps	1	1	5	3	1,453
regions	8	14	25	40	65
regions growing minimal K_{n+1}	< 8	14	< 25	40	< 65

Table 5.1: Regions of K_n Generating Minimal K_{n+1}

Conjecture 4 *In generating minimal K_n, for odd n, from minimal K_{n-1}, vertex n placement into any of the r regions of minimal K_{n-1} generates at least one minimal K_n.*

With the development of Star Analysis, one valuable step in the process of iteratively generating minimal K_n from minimal K_{n-1} is available.

The entire process involves seven steps as covered in Section 4.2. Briefly, the process starts with all non-isomorphic representatives of minimal K_{n-1}. In starting initially with K_4, only one graph is required. (1) Input Generation prepares the initial feeder graph(s) by constructing the list of edges to add to the feeder graph(s), indicating edges from vertex n to the other $n-1$ vertices already in the feeder graph. (2) For each region of each feeder graph, an instance of Star Analysis is initiated to generate Path List descriptions for minimal K_n. (3) Selection of graphs for which to create graph descriptions is done (all globally minimal) for generating minimal K_n. (4) Graph descriptions for the selected graphs are created. (5) A format conversion necessary for isomorphic testing is performed on all graph descriptions. (6) To minimize the number of feeder graphs for growth to K_{n+1}, isomorphic testing is done on all pairs of graph descriptions. This categorizes the graphs into isomorphic families. (7) One representative is taken from each isomorphic family. The combined

set of non-isomorphic family representatives become the feeder graphs for growth of minimal K_{n+1}.

The iterative process has been utilized to generate minimal K_5 through minimal K_{11} to date. Verification of Guy's Conjecture 2 for K_{11} is complete for minimal K_{11} generated from minimal K_{10}, $\nu(K_{11}) = 100$. The question still stands if minimal K_n, for n odd and $n \geq 9$, always contains at least one minimal K_{n-1} subgraph. Conjecture 5 is proposed.

Conjecture 5 *All minimal K_n contain at least one minimal K_{n-1} subgraph.*

If this conjecture can be verified, the process presented generates all minimal K_n without needing the preface of being generated from minimal K_{n-1}. Related inquiry into this problem and associated questions are discussed in Section 5.2.

As mentioned, no known illustrations of minimal K_n for $n > 9$ have been located, Figures 5.2 and 5.3 illustrate one of the many non-isomorphic drawings available of both minimal K_{10} (60 crossings) and minimal K_{11} (100 crossings). Figure 5.3 was constructed by placing the star $K_{1,10}$ over minimal K_{10} from Figure 5.2.

In conclusion, the above process for generating minimal K_n from minimal K_{n-1} is proposed to apply to other graph families with modification only to the initial feeder graphs and their associated list of edges to add. The following is an example of how this process can be applied to complete bipartite graphs. Figure 5.4 shows the complete bipartite graph $K_{2,3}$. Figure 5.4(a) illustrates it in a standard presentation, and (b) shows the same graph with the minimal number of crossings of which there is only one up to isomorphism. The graph from (b) is used as the initial feeder graph.

A region list description for Figure 5.4(b) is shown in Figure 5.5(a). The list of edges to add for generating minimal $K_{3,3}$ and minimal $K_{2,4}$, are shown in (b) and (c) respectively.

For brevity, examination of new vertex 6 placement into only region R1 is done. The associated Distance Tree and Path List for Minimal $K_{3,3}$ is shown in Figure 5.6. From the Path List Description it is seen that four minimal $K_{3,3}$, with one crossing

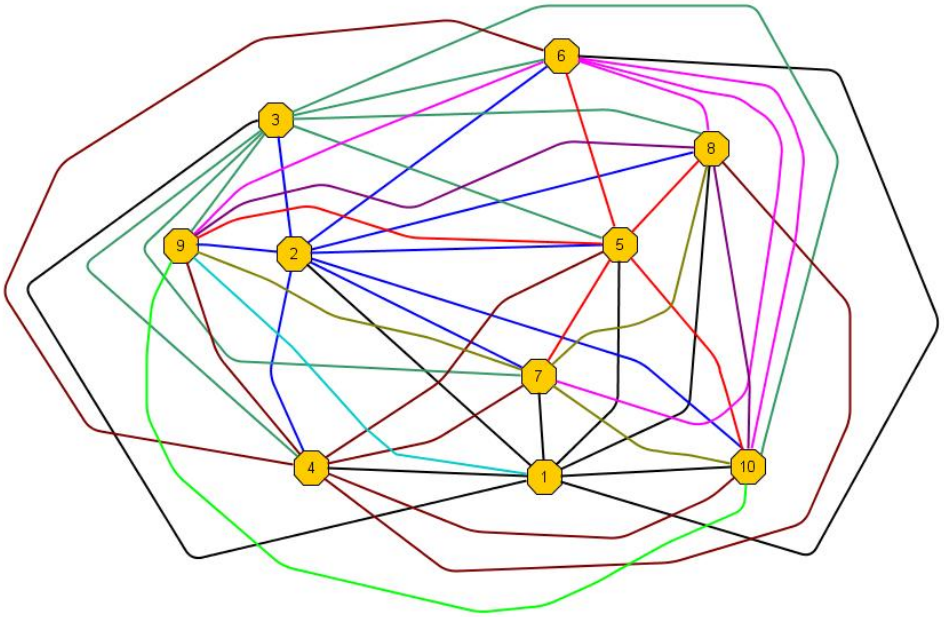


Figure 5.2: One Drawing of a Minimal K_{10}

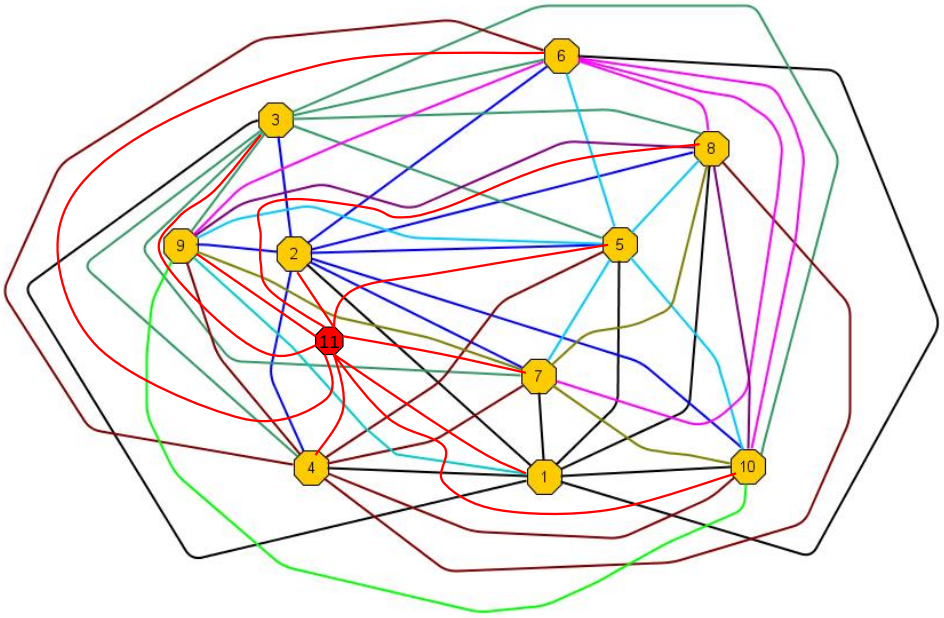


Figure 5.3: One Drawing of a Minimal K_{11}

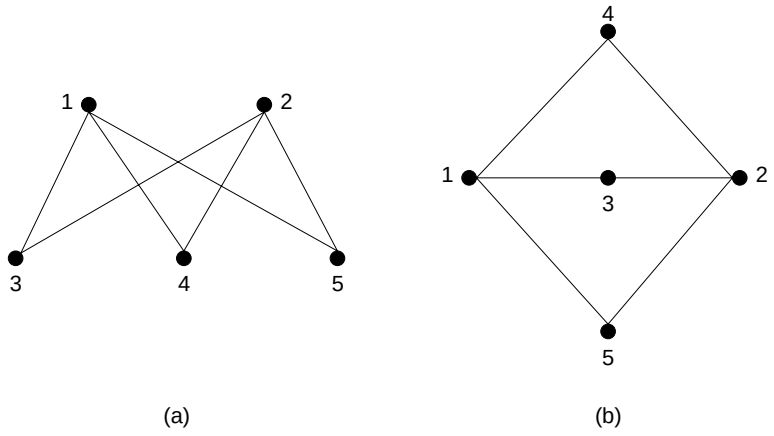


Figure 5.4: $K_{2,3}$ with Three Crossings, and Minimal with Zero Crossings

Minimal K(2,3) Region List	Edges to Add for K(3,3)	Edges to Add for K(2,4)
R1: 1 4 2 3	(6,3)	(6,1)
R2: 1 3 2 5	(6,4)	(6,2)
R3: 1 5 2 4	(6,5)	
(a)	(b)	(c)

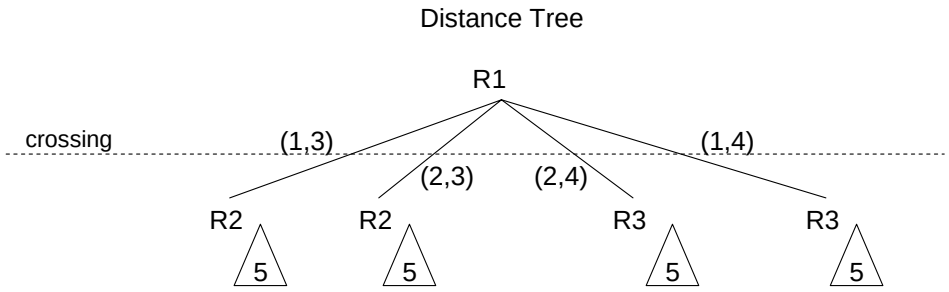
Figure 5.5: Minimal $K_{2,3}$ Region List and Edges to Add for Minimal $K_{3,3}$ and $K_{2,4}$

each, can be constructed. The four graphs are illustrated in Figure 5.7.

Isomorphic testing on the four minimal $K_{3,3}$ from Figure 5.7 results in one isomorphic family. Placement of Vertex 6 in any other region of minimal $K_{2,3}$ results in the same graph, up to isomorphism. Thus, one minimal $K_{3,3}$ is all that is necessary as a feeder graph to generate minimal $K_{3,4}$.

Skipping over the Distance Tree/Path List for minimal $K_{2,4}$, the one graph that is generated given vertex 6 placement in region R1 of $K_{2,3}$, as described in Figure 5.5, is illustrated in Figure 5.8. Placement of vertex 6 in any other region of minimal $K_{2,3}$ results in a graph isomorphic to this, so there is one minimal $K_{2,4}$ from which to grow minimal $K_{2,5}$ and minimal $K_{3,5}$.

This demonstrates the applicability of the framework presented for generating



Path List:	R1	direct to Vertex 3		
	R1	direct to Vertex 4		
	R1	(1,3)	R2	Vertex 5
	R1	(2,3)	R2	Vertex 5
	R1	(2,4)	R3	Vertex 5
	R1	(1,4)	R3	Vertex 5

Figure 5.6: Minimal $K_{3,3}$ Distance Tree and Path List

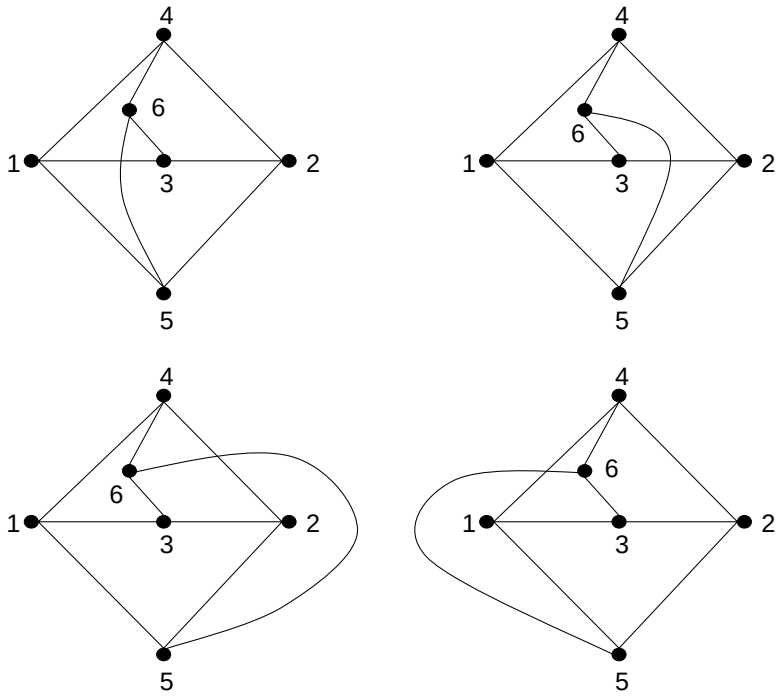


Figure 5.7: Four Isomorphic Minimal $K_{3,3}$ Constructed from Minimal $K_{2,3}$

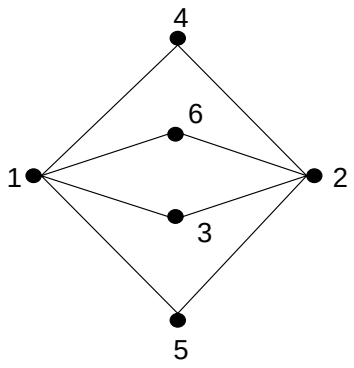


Figure 5.8: Minimal $K_{2,4}$ Constructed from Minimal $K_{2,3}$

minimal $K_{n+1,m}$ and minimal $K_{n,m+1}$ from minimal $K_{n,m}$. Section 5.2 looks toward pursuing minimal $K_{n,m}$ further and applying this framework to other graph families.

5.2 Future Work and Open Questions

Conjectures 4 and 5 are obviously two open questions posed as a result of this research. The following is a list of the directions to which this body of work points with discussion as appropriate.

Analysis of substructures within non-isomorphic, as well as isomorphic, minimal K_n needs to be explored. This analysis, directed at individual regions, tightly local and expanded region neighborhoods, native and crossing vertex relationships within and across regions, a variety of subgraphs based on different structural parameters such as crossing edges or native vertex edges, and other features may allow for deeper understanding of the underlying structure of minimal K_n.

It is obvious, given some graphs, K_4 for example, that some regions of the initial feeder graph in the process of generating minimal K_n are isomorphic. It does not matter into which region of the graph vertex n is placed, the outcome will be the same in all cases. Substructure analysis will proceed to develop an algorithmic technique for determining which regions and larger substructures will result in isomorphic results. This will allow for eliminating duplication of work currently being performed.

Spectra analysis is one approach to pursue in analyzing these substructures. Spectral graph theory is a branch of mathematics that is concerned with characterizing the properties of a graph and extracting information from its structure *via* the eigenvectors of the adjacency or Laplacian matrix of the graph. The spectrum of the graph is obtained from the eigendecomposition of the matrix representation used. Isomorphic graphs are known to be cospectral; that is, they have the same eigenvalues with respect to the matrix used. Some non-isomorphic graphs are cospectal also. Having access to isomorphic family representatives for each minimal K_n generated affords a body of data with which to work. A brief review of graph spectrum information may be found in [55]. Another reference book is [17].

Substructure analysis may help reduce the number of regions in which to place vertex n and may also help with verifying or disproving Conjecture 4. The question arises as to which characteristics are found in minimal K_{n-1}, $n-1$ even, that produce this result, and what different characteristics are found in minimal K_{n-1}, $n-1$ odd, that do not produce the result. Can algorithmic analysis be performed on the regions of K_{n-1}, $n-1$ even, to determine "isomorphic family producing regions"? In other words, can a certain characteristic be found for a set of regions that generate isomorphic K_n? This would allow for vertex n placement in fewer than all regions. Of course, this question can also be asked of minimal K_{n-1} for $n-1$ odd. The question of whether the even/odd growth pattern exhibited in K_n occurs within other graph families awaits Star Analysis of them.

Conjecture 5 points toward recreating non-isomorphic K_9 that Guy says existed but were lost over time. A comparison of his graphs, stated to represent all non-isomorphic K_9, can be made against those created *via* this newly presented process to check for differences. If any exist, the illusive minimal K_9 not derivable from minimal K_8 may be found and be available to explore how it may be constructed and added to the existing set of minimal K_9 from minimal K_8. If no difference is found, movement is one step closer to verifying Conjecture 5. A related issue that needs addressing is if the definition of isomorphism used by Guy is his graph generation differs from the

standard definition used in this body of work. Analysis of these minimal K_n using his definition would be an interesting study.

Aside from this approach in examining K_9, another way to approach Conjecture 5 is to ask the question: Does there exist a K_8 with nineteen crossings that generates a minimal K_9 with a maximum responsibility of 17 for any vertex?

In explanation of this question, recall that the average vertex responsibility of K_9 is sixteen. If a vertex with responsibility 16 is removed, a K_8 with twenty crossings is left. If a vertex with responsibility 18 is removed, a minimal K_8 is left. All vertices cannot have responsibility 16, or minimal K_8 could not have eighteen crossings. So, if these special minimal K_9 do exist, there must exist a minimal K_9 with maximum responsibility of seventeen, which contains no minimal K_8 subgraph. Removal of this vertex with responsibility seventeen would result in a K_8 subgraph with nineteen crossings. Of course, this question addresses only minimal K_9, not larger odd n, but answering it may lead in the direction of a general answer.

An encouraging note in support of Conjecture 5 is the fact that when Guy's conjecture of the equality of Theorem 10 failed, he stated that he anticipated that Conjecture 2 would fail also as n approaches 14. The underlying argument from which he deduced that there exist minimal K_9 lacking minimal K_8 subgraphs is based on the validity of Conjecture 2.

As seen by the final example in Section 5.1, there are application possibilities to graph families other than complete graphs. Complete graphs are not the only family lacking exact results and having little available data for analysis. Examination of other families is expected to supply new exact results from the growth of graphs of order $n-1$ to graphs of order n, as well as build the enumerated isomorphic family representatives for analysis of these families.

All the above mentioned work has application beyond the improvement of the framework of this presentation. As mentioned, lower bound techniques for estimating lower bounds of the crossing number are lacking. Discoveries from the above analysis may lead to new or improved estimating techniques.

One further area of research to pursue from this work is that of graph drawing. The graphs generated *via* this process are represented as region lists. A software package was developed specifically to aid in drawing these graphs since most packages available were not found to be receptive to this representation. Because the package developed is a first generation product, it is lacking in features and ability to produce pleasing visual representations of these graphs. The regions of the graphs are known, and with Path Lists available, the paths are known. This information should aid in developing a system for generating visual representation of these graphs that is pleasing to the eye. It is known that any graph may be topologically morphed such that any region of the graph may be the exterior region. A tool to try different exterior region configurations in combination with other known information about the graph is one interesting direction to pursue.

Bibliography

[1] Oswin Aichholzer, Franz Aurenhammer, and Hannes Krasser. Enumerating order types for small point sets with applications. In *Proceedings of the seventeenth annual symposium on Computational geometry, Annual Symposium on Computational Geometry*, pages 11–18. Association for Computing Machinery, ACM Press, 2001.

[2] Oswin Aichholzer, Franz Aurenhammer, and Hannes Krasser. Progress on rectilinear crossing numbers. www.igi.tugraz.at/oaich/psfiles/aak-prcn-01.ps.gz, IGI-TU, Austria, 2002.

[3] Oswin Aichholzer, Ferran Hurtado, and Marc Noy. On the number of triangulations every planar point set must have. In *Proceedings of the 13th Canadian Conference Computation Geometry, Annual Symposium on Computational Geometry*, pages 13–16, 2001.

[4] Oswin Aichholzer and Hannes Krasser. The point set order type data base: a collection of applications and results. In *Proceedings of the 13th annual Canadian Conference on Computational Geometry CCCG 2001*, pages 17–20, 2001.

[5] Oswin Aichholzer and Hannes Krasser. Abstract order type extension and new results on the rectilinear crossing number. In *Proceedings of the twenty-first annual symposium on Computational geometry, Annual Symposium on Computation Geometry*, pages 91–98. Association for Computing Machinery, ACM Press, 2005.

[6] M. Ajtai, V. Chvatal, M. Newborn, and E. Szemeredi. Crossing-free subgraphs. *Annals of Discrete Mathematics*, 12:9–12, 1982.

[7] G. Di Battista, R. Eades, P. Tamassia, and I.G. Tollis. Algorithms for drawing graphs: an annotated bibliography. *Computational Geometry: Theory and Applications*, **4**(5):235–282, 1994.

[8] G. Di Battista, R. Eades, P. Tamassia, and I.G. Tollis. *Graph Drawing: Algorithms for Visualization of Graphs*. Prentice Hall, Upper Saddle river, NJ, 1st edition, 1999.

[9] Daniel Bienstock. Some provably hard crossing number problems. *Discrete Computation Geometry*, **6**:443–459, 1991.

[10] Daniel Bienstock and N. Dean. Bounds for rectilinear crossing numbers. *Journal of Graph Theory*, **17**:333–348, 1993.

[11] J. Blazek and M. Koman. A minimal problem concerning complete plane graphs. In *Proc. Theory of Graphs and its Applications*, pages 333–338, New York, 1965. Academic Press.

[12] Alex Brodsky, Stephane Durocher, and Gethner Ellen. The rectilinear crossing number of K_{10} is 62. Technical Report TR-2000-10, University of British Columbia, October 2000. citeseer.ist.psu.edu/article/brodsky00rectilinear.html.

[13] G. Chartrand and L. Lesniak. *Graphs and Digraphs*. Chapman & Hall/CRC, Boca Raton, FL, 3nd. edition, 1996.

[14] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein. *Introduction to Algorithms*. MIT Press with McGraw Hill, 2nd edition, 2002.

[15] E. de Klerk, J. Maharray, D.V. Pasechnik, R.B. Richter, and G. Salaar. Improved bounds for the crossing numbers of $K_{m,n}$ and K_n. http:// citeseer.ist.psu.edu/682027.html, submitted for publication, 2004.

[16] Hristo N. Djidjev and Vrt'o Imrich. Crossing numbers and cutwidths. *Journal of Graph Algorithms and Applications*, **7**:245–251, 2003.

[17] Dragos M. Dvetkovic, Michael Doob, and Horst Sachs. *Spectra of Graphs: Theory and Applications*. Academic Press, 3rd edition, 1997.

[18] J. Edmonds. A combinatorial representation for polyhedral surfaces. *Notices of the American Mathematical Society*, **7**:646, 1960.

[19] Rober B. Eggleton. *Crossing Numbers of Graphs*. PhD thesis, University of Calgary, 1973.

[20] P. Erdos and Guy Richard K. Crossing number problems. *American Mathematical Monthly*, **88**:52–58, 1973.

[21] I. Fáry. On the straight line representations of planar graphs. *Acta Scientiarum Mathematicarum*, **11**:229–233, 1948.

[22] Scott Fortin. The graph isomorphism problem. Technical Report TR96-20, University of Alberta, July 1996. www.cs.ualberta.ca/research/techreports /1996/TR96-20.php.

[23] Judith R. Fredrickson, Bei Yuan, and Frederick C. Harris, Jr. A time saving region restriction for calculating the crossing number of K_n. *Congressus Numerantium*, **168**:145–158, May 2004.

[24] Michael R. Garey and David S. Johnson. *Computers and intractability a guide to the theory of NP-completeness*. W. H. Freeman and Company, New York, 1979.

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

[26] L. Yu. Glebsky and G. Salazar. The crossing number of $C_m \times C_n$ is as conjectured for $n > m(m+1)$. Journal of Graph Theory, to appear.

[27] Martin Grohe. Computing crossing numbers in quadratic time. *Journal of Computer and System Sciences*, **68**(2):285–302, March 2004.

[28] H. Gronau and H. Harborth. Numbers of nonisomorphic drawings for small graphs. *Congressus Numerantium*, **71**:105–114, 1990.

[29] Richard K. Guy. A combinatorial problem. *Nabla (Bulletin of the Malayan Mathematical Society)*, **7**:68–72, 1960.

[30] Richard K. Guy. The decline and fall of Zarankiewicz's theorem. In Frank Harary, editor, *Proof Techniques in Graph Theory (Proc. Second Ann Arbor Graph Theory Conf., Ann Arbor, Mich., 1968)*, pages 63–69. University of Michigan, 1969.

[31] Richard K. Guy. Crossing numbers of graphs. In *Graph Theory and Applications (Proc. of the Conference at Western Michigan University, 1972)*, pages 111–124. Western Michigan University, 1972.

[32] Richard K. Guy, Tom Jenkyns, and Jonathan Schaer. The toroidal crossing number of the complete graph. *Journal of Combinatorial Theory*, **4**:376–390, 1968.

[33] Frank Harary and Anthony Hill. On the number of crossings in a complete graph. In *Proceedings of the Edinburgh Mathematical Society*, volume **13** 2nd Series, pages 333–338, London, 1963. Edinburgh Mathematical Society.

[34] 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 International Conference on Graph Theory, Combinatorics, Algorithms, and Applications*, volume **2**, pages 469–478. Western Michigan University, June 1998.

[35] Linda Humphrey. Efficient generation of minimal graphs using independent path analysis. Master's thesis, University of Nevada, Reno, NV, 2006.

[36] H.F. Jensen. An upper bound for the rectilinear crossing number of the complete graph. *Journal of Combinatorial Theory*, **10** Series B:212–216, 1971.

[37] Daniel J. Kleitman. The crossing number of $K_{5,n}$. *Journal of Combinatorial Theory*, **9**:315–323, 1970.

[38] F. T. Leighton. *Complexity Issues in VLSI.* MIT Press, 1983.

[39] F. T. Leighton. New lower bound techniques for VLSI. *Math. Systems Theory*, **17**:47–70, 1984.

[40] Annegret Liebers. Planarizing graphs - a survey and annotated bibliography. *Journal of Graph Algorithms and Applications*, **5**:1–74, 2001.

[41] Sean Christopher Martin. A parallel queuing system for computationally intensive problems of medium to large beowulf clusters. Master's thesis, University of Nevada, Reno, NV, 2003.

[42] Brendan D. McKay. Practical graph isomorphism. *Congressus Numerantium*, **30**:45–87, 1981.

[43] F. Pach, J. Shahrokhi and M. Szegedy. Applications of crossing numbers. *Algorithmica*, **16**:11–117, 1996.

[44] J. Pach and G. Tóth. Thirteen problems on crossing numbers. *Geombinatorics*, **9**:225–246, 2000.

[45] J. Pach and G. Tóth. Which crossing number is it anyway? *Journal of Combinatorial Theory*, **80** Series B:225–246, 2000.

[46] Janos Pach, Joel Spencer, and Geza Tóth. New bounds on crossing numbers. Technical Report 37, Dimacs, Rutgers University, June 1999. ftp:// dimacs.rutgers.edu/pub/dimacs/TechnicalReports/TechReports/1999/99-37.ps.gz.

[47] Michael J. Pelsmajer, Marcus Schaefer, and Daniel Stefankovic. Odd crossing number is not crossing number. In *Healy, Patrick and Nikolov, Nikola S., Eds, Proceedings Graph Drawing*, volume **13** 2nd Series, pages 386–396, Limerick, Ireland, 2006. Edinburgh Mathematical Society.

[48] R. B. Richter and C. Thomassen. Relations between crossing numbers of complete and complete bipartite graphs. *American Mathematical Monthly*, **104**:131–137, Feb. 1997.

[49] Farhad Shahrokhi, Ondrej Sykora, Laszlo A. Szekely, and Vrt'o Imrich. A new lower bound for the bipartite crossing number with applications. *Theoretical Computer Science*, **245**:281–294, August 2000.

[50] David A. Singer. The rectilinear crossing number of certain graphs. Case Western Reserve University, Cleveland, OH, 1971.

[51] O. Sykora and I. Vrt'o. On VLSI layouts of the star graph and related networks. *Integration, the VLSI Journal*, **17**:83–93, 1994.

[52] Umid Tadjiev. Parallel computation and graphical visualization of the minimum crossing number of a graph. Master's thesis, University of Nevada, Reno, NV, 1998.

[53] Umid Tadjiev and Frederick C. Harris, Jr. Parallel computation of the minimum crossing number of a graph. *Proceedings of the Eighth SIAM Conference on Parallel Processing for Scientific Computing*, 1997.

[54] John T. Thorpe and Frederick C. Harris, Jr. A parallel stochastic optimization algorhtm for finding mappings of the rectilinear minimal crossing problem. *Ars Combinatoria*, **43**:135–148, 1996.

[55] Edwin R. van Dam and Willem H. Haemers. Which graphs are determined by their spectrum? *Linear Algebra and its Applications*, **373**:241–272, 2003.

[56] Imrich Vrt'o. Crossing numbers of graphs: a bibliography. http:// www.ifi.savba.sk/~imrich, ftp://ifi.savba.sk/pub/imrich/crobib.ps.gz.

[57] Douglas B. West. *Introduction to Graph Theory*. Prentice Hall, Upper Saddle River, NJ, 1st. edition, 1996.

[58] D. R. Woodall. Cyclic-order graphs and Zarankiewicz's crossing-number conjecture. *Journal of Graph Theory*, **17**(6):137–145, 1993.

[59] J. Youngs. Minimal embeddings and the genus of a graph. *Journal of Mathematical Mechanics*, **12**:303–315, 1963.

[60] Bei Yuan. A generic queueing system and time saving region restrictions for calculating the crossing number of K_n. Master's thesis, University of Nevada, Reno, NV, 2004.

[61] Bei Yuan, Sean C. Martin, Judith R. Fredrickson, and Frederick C. Harris, Jr. A generic queuing system for computationally intensive problems. *Congressus Numerantium*, **171**:193–206, May 2004.

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