2001

Enumeration of small triangle free Ramsey Graphs

Robert Getschmann

Rochester Institute of Technology
Department of Computer Science
Thesis

Enumeration of Small Triangle Free Ramsey Graphs

by

Robert A. Getschmann

A thesis submitted to
The Faculty of the Computer Science Department
in partial fulfillment of the requirements for the degree of
Master of Science in Computer Science

Approved by:

Dr. Stanisław Radziszowski
Chairman

Dr. Edith Hemaspaandra
Reader

Dr. Peter G. Anderson
Observer

July 31, 2001

Rochester Institute of Technology
Wallace Memorial Library
Thesis Reproduction Permission Statement

Title of Thesis: Enumeration of Small Triangle Free Ramsey Graphs

I, Robert A. Getschmann hereby **grant permission** to the Wallace Library of the Rochester Institute of Technology to reproduce in whole or in part this thesis. Any reproduction will not be for commercial use or profit.

Signature of Author: _____ **Date:** Nov. 30 2001

ABSTRACT

In 1930, a paper by Frank Plumpton Ramsey entitled *"On a Problem of Formal Logic"* appeared in the *Proceedings of the London Mathematical Society.* Although the impetus of this paper was one of mathematical logic, a far–reaching combinatorial result was needed by Ramsey to achieve his objective. This combinatorial result became known as Ramsey's Theorem.

One of the combinatorial structures which was developed during the study of Ramsey's Theorem is that of a Ramsey graph. A Ramsey graph, denoted (k, l, n, e), is defined as an undirected graph that contains no cliques of size k, no independent sets of size l, with order n, and size e.

Knowledge of Ramsey graphs is useful in the improvement of bounds and sometimes the calculation of exact values for various Ramsey number parameter situations.

Straightforward enumeration of (k, l, n, e) Ramsey graphs for larger values of n is intractable with the current computing technology available. In order to produce such graphs, specialized algorithms need to be implemented.

This thesis provides the theoretical background developed by Graver and Yackel [GRA68a], expanded upon by Grinstead and Roberts [GRI82a], and generalized by Radziszowski and Kreher [RAD88a, RAD88b] for the implementation of algorithms utilized for the enumeration of various Ramsey graphs.

An object–oriented graph manipulation package, including the above mentioned Ramsey graph enumeration algorithms, is implemented and documented.

This package is utilized for the enumeration of all $(3,3)$, $(3,4)$, $(3,5)$ and $(3,6)$ graphs. Some $(3,7)$ and $(3,8)$ also are calculated. These results duplicate and verify Ramsey graphs previously enumerated during other investigations. [RAD88a, RAD88b]

In addition to these results, some newly enumerated $(3,8)$ critical graphs, as well as some newly enumerated $(3,9)$ graphs, including a minimum $(3,9,26,52)$–graph are presented.

Table of Contents

1. Introduction 2
1.1. Ramsey's Theorem 2
1.2. Ramsey Numbers 3
1.2.1. Known Ramsey Numbers 5
1.3. Ramsey Graphs 5
1.3.1. Why Ramsey Graphs are Studied 6
1.3.2. Ramsey Graph Enumeration Difficulties 7
1.4. Historical Overview 8
1.5. Thesis Outline 13
1.5.1. Goals 13
1.5.2. Theoretical Overview 14
1.5.3. Software Libraries 14
1.5.4. Software Tools 15
1.5.5. Graph Enumerations 16
2. Graph Theory 17
2.1. Graph Theoretical Definitions 17
3. Ramsey Graphs 26
3.1. Properties of Ramsey Graphs 26
3.2. Bounds for $(3,8)$ Ramsey Graphs 37
3.3. Bounds of $e(3,l,n)$ for $9 \leq l \leq 13$ and $3k-1 \leq n$ 37
3.4. Bounds for Ramsey numbers $R(3,l)$, with $10 \leq l \leq 14$" 38
4. Implementation 40
4.1. Graph Data Formats 40
4.1.1. Graph State Formats 41
4.1.1.1. Graph y−format 41
4.1.1.2. Graph Adjacency Matrix Format 46
4.1.1.3. Graph Incidence Matrix Format 47
4.1.1.4. Graph Adjacency List Format 48
4.1.2. Graph Property Formats 49
4.1.2.1. Clique Format 50
4.1.2.2. Independent Set Format 50
4.1.2.3. Degree Sequence Format 50
4.1.2.4. Isomorphic Map Format 51
4.1.2.5. Distance Matrix 52
4.1.2.6. Column Characteristic Matrix Format 53
4.1.2.7. Row Characteristic Matrix Format 54
4.1.2.8. Characteristic Matrix Format 55
4.1.3. Graph Image Formats 56
4.1.3.1. DVI Format 56

4.1.3.2. GIF Format 57
4.1.3.3. PIC Format 57
4.1.3.4. PS Format 58
4.2. Graph File Formats 58
4.2.1. G−Format 58
4.3. Algorithms 59
4.3.1. ISOMAP Algorithm 60
4.3.2. DELTA Algorithm 63
4.3.3. EXTEND Algorithm 65
4.3.4. EXPAND Algorithm 67
4.3.4.1. Ramsey Graph Degree Sequence Calculation . . 67
4.3.4.2. $v + H_1(v)$ Graph Calculation 68
4.3.4.3. Independent Set Calculation 68
4.3.4.4. Good Independent Set Calculation 69
4.3.4.5. M−Matrix Calculation 70
4.3.4.6. $H_1(v)$ and $H_2(v)$ Graph Attachment 71
4.3.4.7. Resulting Ramsey Graph Checks 71
4.4. Custom Software Packages 73
4.4.1. Package Configuration and Build Process 73
4.4.2. error Package 75
4.4.3. grama Package 76
4.4.4. tool Package 78
4.4.5. ytool Package 78
4.4.6. grama Test Suite 91
4.5. Third Party Software Packages 93
4.5.1. autoson Package 93
4.5.2. gtools Package 93
4.5.3. Maple V Package 94
4.5.4. nauty Package 94
5. Execution 95
5.1. Enumeration of $(3,3)$, $(3,4)$, $(3,5)$, and $(3,6)$ Graphs" 95
5.2. Enumeration of $(3,7)$, $(3,8)$, and $(3,9)$ Graphs" 99
6. Results 102
6.1. $(3,3)$, $(3,4)$, $(3,5)$, and $(3,6)$ − Ramsey Graphs 102
6.2. $(3,7)$ Ramsey Graphs 106
6.2.1. $(3,7)$ Ramsey Graph Summary 106
6.2.2. Critical $(3,7)$ − Ramsey Graphs 107
6.3. $(3,8)$ Ramsey Graphs 115
6.3.1. $(3,8)$ Ramsey Graph Summary 115
6.3.2. Critical $(3,8)$ − Ramsey Graphs 116
6.4. $(3,9)$ Ramsey Graphs 128
6.4.1. $(3,9,26,52)$ Ramsey Graph Summary 128

6.4.2. $(3, 9, 26, 52)$ Ramsey Graph 129
7. Appendix A grama Test Suite Example 131
8. Appendix B - Glossary 136
9. References 145

Chapter 1. Introduction

1.1. Ramsey's Theorem

In 1930, a paper by Frank Plumpton Ramsey entitled *"On a Problem of Formal Logic"* appeared in the *Proceedings of the London Mathematical Society.* Although the primary concern of this paper was one of mathematical logic, a far–reaching combinatorial result was needed by Ramsey to achieve his objective. This result, which is a profound generalization of the pigeon–hole principle (Dirichlet's Box Principle), has become known as *Ramsey's Theorem.* [WIN88a]

Ramsey was aware of the combinatorial result which he had discovered as is evident from the following quote.

> *"But in the course of this investigation it is necessary to use certain theorems on combinatorics which have an independent interest and are most conveniently set out by themselves beforehand."* [WIN88a]

The discovery that Ramsey made was that a set of objects be partitioned into subsets such that the majority of the elements are contained within one of the subsets.

In order to present Ramsey's theorem a preliminary definition is required.

Definition 1.1.1.

Let S be an n–set and let $P_r(S)$ be the set of all r–subsets of S. Let

$$P_r(S) = A_1 \bigcup A_2 \bigcup \cdots \bigcup A_t. \tag{1.1.1}$$

be an arbitrary ordered partition of $P_r(S)$ in t components A_1, A_2, ..., A_t. Let q_1, q_2, ..., q_t be integers such that

$$1 \leq r \leq q_1, q_2, \cdots, q_t. \tag{1.1.2}$$

If there exists a q_i–subset of S with all of its r–subsets in A_i, we call the subset a (q_i, A_i)–subset of S. [RYS63a]

□

Given this definition, the formal statement of Ramsey's theorem follows.

Theorem 1.1.1.

Let the given integers $q_1, q_2, \ldots, q_t$ and r satisfy inequality 1.1.2. There exists a minimal positive integer $N(q_1, q_2, \ldots, q_t, r)$ such that the following property is valid for all integers $n \geq N(q_1, q_2, \ldots, q_t, r)$. Let S be an $n-$set and allow for an arbitrary ordered partition of $P_r(S)$ into t components $A_1, A_2, \ldots, A_t$, as given in Definition 1.1.1. S will then contain a $(q_i, A_i)-$subset for some $i = 1, 2, \ldots, t$. [RYS63a]

□

There was an early application of Ramsey's Theorem in the 1930's by the Hungarian mathematicians Paul Erdős and George Szekeres. Ramsey's theorem was applied to a problem in combinatorial geometry. The following statement describes the specific problem examined. [ERD35a]

> *"From 5 points of the plane of which no three lie on the same straight line it is always possible to select 4 points determining a convex quadrilateral."* [ERD35a, WIN88a]

Following the investigation of this specific problem, Erdős and Szekeres provided the following generalized problem statement.

> *"Can we find for a given n, a number $N(n)$ such that from any set containing at least N points, in general position, it is possible to select n points forming a convex polygon?"* [WIN88a]

Erdős and Szekeres attribute the solution of the first problem, and the formulation of the more general problem statement to Esther Klein, a member of the "Budapest Circle of Gifted Young Mathematicians." The solution of the more general problem by Erdős and Szekeres is generally considered to be a rediscovery of the theory described by Ramsey. [WIN88a]

The principles behind Ramsey's Theorem permeate many different fields of mathematics including set theory, geometry, and graph theory. Throughout this thesis only the graph theoretical approach is examined.

1.2. Ramsey Numbers

The discovery of the combinatorial puzzle known as Ramsey's Theorem, and the subsequent *rediscovery* by Erdős and Szekeres provided a plethora of problems to which mathematicians could apply their talents. Such studies have led to a standardized notation and set of definitions to be used when describing problems relating to the theorem. One such object is that of a Ramsey number. The following statement provides a formal definition for a *t−color Ramsey number.*

Definition 1.2.1.

> *Given the positive integers s, t, and* $k_1, k_2, \cdots k_t$*, the Ramsey Number* $R(k_1, k_2, \cdots k_t)$ *is defined as the smallest integer n such that for any t−coloring of the s−faces of* K_n *with colors* $\gamma_1, \gamma_2, \cdots \gamma_t$ *there is either a* $\gamma_1 - K_{k_1}$ *or a* $\gamma_2 - K_{k_2}, \cdots$*, or a* $\gamma_t - K_{k_t}$ *sub−graph (i.e. for some i with* $1 \leq i \leq t$*, there is a complete* K_{k_i} *sub−graph of* K_n *all of whose s−faces have color* γ_i*.)*
>
> □

From the above definition, the $R(k_1, k_2, \cdots k_t)$ Ramsey number parameter situations for $t = 2, 3, 4, \ldots$ can be defined where the values of k_i may be arbitrarily chosen from the set of positive integers.

Ramsey number problem instances for which $t = 2$ are referred to as a two−color Ramsey numbers. The following statement provides a formal definition for the specific *two−color Ramsey number* situation.

Definition 1.2.2.

> *Given the positive integers k and l, a two−color Ramsey number* $R(k, l)$ *is defined as the smallest integer n, such that any graph on n vertices contains either a clique of size k or an independent set of size l.*
>
> □

As with the multi−color Ramsey number situation, for a two−color Ramsey number $R(k, l) = n$, a graph G of order n is guaranteed to exist.

Given that this particular problem situation involves only two colors, two-color Ramsey numbers are typically represented without *any* colored edges, but with edges and non-edges. The resulting graph is therefore *not* a complete graph. One color is represented by the edges of the graph while the second color is represented by the lack of edges. This makes the representation of such instances easier to visualize and adheres to popular graph diagramming techniques. This notational convenience is utilized throughout this thesis.

This thesis deals explicitly with two−color Ramsey numbers and two−color Ramsey graphs. Throughout the remainder of this document any reference to Ramsey numbers implies the above two−color Ramsey number definition.

1.2.1. Known Ramsey Numbers

Since the proof of the existence of a value for any Ramsey number parameter situation, values as well as bounds for some situations have been calculated.

Table 1.2.1.1 presents all of the currently known values and bounds for various two−color Ramsey number parameter situations. Some of these situations are trivial and can be proven by simple construction. Others have been established via the calculation of various Ramsey graphs or the proof of the non−existence of certain Ramsey graph parameter situations. [RAD01a] A current summary of the known bounds and values can be found at the *Electronic Journal of Combinatorics* site (http://www.combinatorics.org/).

Note the position of the number(s) in each of the boxes. Empty boxes with $k \leq l$ indicate that neither nontrivial bounds nor an exact value are known for the given instance. Entries that contain a single number indicate the exact value for the situation. Entries that contain two numbers provide the currently known bounds for the instance. The *top* number indicates the lower bound and the *bottom* number indicates the upper bound. Some entries contain a number and a *dash* character. For such an entry the numerical value indicates the single known lower or upper bound (given by its position), while the dash indicates that the remaining bound is unknown.

1.3. Ramsey Graphs

Since 1930 a great number of researchers have studied the combinatorial puzzle set forth by Ramsey. One of the structures which developed from this study was that of a Ramsey graph.

Ramsey graphs are combinatorial structures which meet various criteria for specified parameters. A two−color Ramsey graph is an undirected graph which contains no cliques of a given size and no independent sets of a given size.

The notation $(k, l, n, e)-graph$ denotes a Ramsey graph that contains no cliques of size k, no independent sets of size l, has an order of n, and a size of e. If neither the graph size nor the graph order is specified the notation is generally shortened to (k, l, n) or (k, l) respectively. From this definition it is evident that the clique number for a (k, l) Ramsey graph is less than k and that the independence number of the graph is less than l.

k/l	3	4	5	6	7	8	9	10	11	12	13	14	15
3	6	9	14	18	23	28	36	40 43	46 51	51 60	58 69	66 78	73 89
4		18	25	35 41	49 61	55 84	69 115	80 149	96 191	128 238	131 291	136 349	145 417
5			43 49	58 87	80 143	95 216	121 316	141 442	153 –	181 –	193 –	221 –	242 –
6				102 165	109 298	122 495	153 780	167 1171	203 –	230 –	242 –	284 –	374 –
7					205 540	– 1031	– 1713	– 2826		312 –			
8						282 1870	– 3583	– 6090					
9							565 6588	– 12677					
10								798 23581					

Table 1.2.1.1. – Values and Bounds for Two Color Ramsey Numbers.

Knowledge of various Ramsey graph structural properties is useful in the calculation of such graphs. Some such properties include valid Ramsey graph degree sequences, minimum and maximum vertex degree values, and minimum and maximum Ramsey graph sizes. The minimum size for a (k, l, n) graph is denoted by $e(k, l, n)$, while the maximum size is denoted $E(k, l, n)$. The usefulness of such properties is detailed in Chapter 3 which presents the theoretical background for some Ramsey graph enumeration techniques.

1.3.1. Why Ramsey Graphs are Studied

Based on the informal definition of a Ramsey graph from the preceding section, a two–color *Ramsey number* $R(k, l)$ is defined as the smallest integer n such that no (k, l, n)*–graph* exists. Therefore if at least one (k, l, n) Ramsey graph is known to exist, but no $(k, l, n+1)$ Ramsey graph exists, then the value of $R(k, l) = n + 1$. [RAM30a]

From the definition of a Ramsey graph it should be obvious that the existence of such graphs can be useful in the improvement of the known bounds, and sometimes the calculation of the exact value of various Ramsey numbers.

For instance, currently there is no known exact value for the Ramsey number $R(3, 10)$. However the bounds for this parameter situation are known to be $40 \leq R(3, 10) \leq 43$. Knowledge of various $(3, 8)$ and $(3, 9)$ Ramsey graphs may be valuable in the improvement of these bounds and possibly in the calculation of the exact value for this problem instance.

In addition to the above mentioned usefulness of Ramsey graphs, the study and implementation of optimized algorithms for the enumeration of these combinatorial structures is extremely challenging and interesting in its own right.

1.3.2. Ramsey Graph Enumeration Difficulties

For *smaller* parameter situations, all Ramsey graphs can be enumerated using straightforward brute force methods. Such methods involve enumerating all graphs with a given order and size, removing isomorphic graphs from contention, and finally eliminating graphs containing a clique or an independent set of a specified size for the particular parameter situation.

The $(3,3)$ Ramsey graph instance is small enough such that the brute force technique described above can be utilized. The following pseudo−code segment demonstrates how such techniques could be implemented for the enumeration of all $(3,3,n,e)$ Ramsey graphs, where $1 \leq n \leq 5$ and $0 \leq e \leq n(n-1)/2$.

```
for ( i = 1; i <= n; ++i ) {
        for ( j = 0; j <= e; ++j ) {
                Calculate all Valid Degree Sequences for a Graph of Order i and Size j
                Realize all Degree Sequences into Graphs
                Examine Graphs and Remove Isomorphic Graphs
                Examine Remaining Graphs and Remove those with Cliques of Size 3
                Examine Remaining Graphs and Remove those with Independent Sets of Size 3
        }
}
```

Enumeration of Ramsey graphs for these small parameter situations is trivial. However, as the parameters for the graphs grow even *slightly*, the graph enumeration process becomes intractable.

As an example, the Ramsey number $R(3,8)$ has been calculated as having a value of 28, and the Ramsey number $R(3,9)$ a value of 36. Given the relationship between Ramsey number values and Ramsey graph orders, $(3,8)-$graphs with orders of n with $0 < n < 28$ exist, as well as $(3,9)-$graphs with orders of n with $0 < n < 36$.

The task of calculating $(3,8)$ and $(3,9)$ Ramsey graphs using the straightforward enumeration techniques provided in the above pseudo−code is intractable with current computational technology.

It is a well established fact that for a graph of order n, there exist $2^{\binom{n}{2}}$ *labeled* graphs. Therefore, for a graph order of 10 there are $2^{\binom{10}{2}}$ or $35,184,372,088,832$ labeled graphs.

The enumeration of all labeled graphs of order 27 would produce $2^{\binom{27}{2}}$ or 2^{351} graphs, while the enumeration of all labeled graphs of order 35 would produce $2^{\binom{35}{2}}$ or 2^{595} graphs.

The number of labeled graphs which exist for these orders is obviously huge. Eliminating all isomorphic graphs and examining the remaining graphs for cliques and/or independent sets of a given size for the parameter situation under investigation would be intractable.

The calculation of various $(3,8)$ and $(3,9)$ graphs, for larger orders, such as 27 and 35 respectively, is possible only if techniques which need not examine all graphs for a given order and size are implemented. These techniques involve the construction of $(3,l)$ Ramsey graphs from previously constructed smaller Ramsey graphs, specifically $(3,l-1)$ Ramsey graphs.

Research leading to such techniques for the calculation of Ramsey graphs is an interesting and challenging field of study. The purpose of this thesis is the implementation of such techniques to be used for the calculation of various Ramsey graphs.

Chapter 3 of this thesis details the theoretical groundwork and background needed to comprehend the algorithms implemented for the construction of Ramsey graphs from smaller previously calculated Ramsey graphs.

1.4. Historical Overview

This section provides a brief summary of some of the various Ramsey graph and Ramsey number problems researched during the last century.

Given that the impetus of this thesis involves two−color, triangle−free Ramsey graphs, the majority of the historical record provided is related to Ramsey graph and Ramsey number problems with similar parameter situations.

In his 1930 paper, Ramsey provided the equation for the asymptotic bounds of the combinatorial puzzle which he had discovered. Various comments made by Ramsey suggest that he was aware of the fact that the bounds calculated by this equation were unnecessarily *loose*, and that improvements could be made. Ramsey may well have made such improvements, were it not for his premature death in 1930 at age 26. [WIN88a]

In 1935 Erdős and Szekeres published a paper entitled *"On a Combinatorial Problem in Geometry"*. This paper presented a combinatorial problem involving the partitioning of elements into sets. [GOU88a, WIN88a]

This paper established that for any two values k and l, the asymptotic value for a Ramsey number is bounded by $R(k,l) \leq \binom{k+l-2}{k-1}$. This inequality provided a

dramatic improvement over the bounds initially calculated by Ramsey. [ERD35a, GOU88a, WIN88a]

Regarding Ramsey numbers $R(k,l)$, there is special interest in the case where $k = l$. In the 1947 paper *"Some Remarks on the Theory of Graphs"*, Erdős and Szekeres improved upon the asymptotic bounds for the general $R(k,l)$ number case by showing that $R(k,k) \geq ck^{k/2}[1/e\sqrt{2} + o(1)]$, for some constant c. This was the first significant improvement in asymptotic bounds since the 1935 Erdős and Szekeres paper. [ERD47a, WIN88a]

While research continued in an attempt to improve the asymptotic bounds for the general two−color Ramsey number problem, other research concentrated on calculating values and bounds for specific Ramsey numbers.

Determination of exact Ramsey number values has in general been unfruitful. This is evident by size of Table 1.2.1.1. and the number of exact values it contains. However, during the last two decades of the twentieth century research involving calculations for specific Ramsey numbers has improved due to the application of algorithms implemented and executed by computer.

The first widespread Ramsey number calculation involved the instance $R(3,3)$. This problem instance was known to have a value less than or equal to 6. The task of calculating the value $R(3,3) \leq 6$ was set forth as the second problem in Part I of the William Lowell Putnam Mathematical Competition in March 1953. It was stated as follows.

> *"Six points are in general position in space (no three in a line, no four in a plane). The fifteen line segments joining them in pairs are drawn, and then painted, some segments red some blue. Prove that some triangle has all its sides the same color."* [GAR78a, WIN88a, RAD01a]

This problem is typically referred to as the *party problem.* It is usually encountered by undergraduate computer science and mathematics students in a *discrete mathematics* course as an example of the pigeon−hole principle.

The $R(3,3)$ number party problem was also presented in the June/July 1958 issue of Mathematical Monthly and was stated as follows.

> *"Prove that at a gathering of any six people, some three of them are either mutual acquaintances or complete strangers to one another."* [WIN88a]

Knowledge of various Ramsey graphs can be useful in the calculation of Ramsey numbers as well as improvement in known Ramsey number bounds. Given this, a great deal of research has concentrated on the enumeration of various Ramsey graphs as well as the analysis of the properties of these graphs.

One such property which has received a great deal of attention is that of the minimum and maximum graph size. Denoted $e(k,l,n)$ and $E(k,l,n)$ respectively, knowledge of these values can be useful in determining properties of larger objects such as $(k,l+1)$ graphs.

In 1955, Greenwood and Gleason published a paper entitled *"Combinatorial Relations and Chromatic Graphs"*. The authors provided the results of their work in determining some specific Ramsey numbers. Their research presented a proof for the Ramsey number $R(3,3)=6$, as well as value for the two–color Ramsey numbers $R(3,5)=14$, and $R(4,4)=18$. For multi–color cases, they determined that $R(3,3,3)=17$ and that $R(3,3,3,3)\leq 66$. [WIN88a]

In the same 1955 paper, Greenwood and Gleason also provided an analysis of the size of various triangle–free Ramsey graphs. Specifically they investigated the values of $e(3,l,n)$ for $l\leq 5$. [GRE55a]

In 1966, Kalbfleisch completed his Ph.D. thesis at the University of Waterloo. In this thesis entitled *"Chromatic Graphs and Ramsey's Theorem"*, he presented constructions of all (k,l,n)–graphs for the parameter situations $(3,3,5)$, $(3,4,7)$, $(3,4,8)$, $(3,5,12)$, $(3,5,13)$, and $(3,6,17)$. He partially analyzed the Ramsey graph situation $(3,6,16,32)$, and established the Ramsey number $R(3,6)=18$. [KAL66a, WIN88a]

In their 1968 paper, *"Some Graph Theoretic Results Associated with Ramsey's Theorem"*, Graver and Yackel developed a system for enumerating Ramsey graphs from *known smaller* Ramsey graphs. Specifically (k,l,n)–graphs could be *built* from existing $(k,l-1,n-d-1)$–graphs, where d is the degree of a specific vertex within a (k,l,n)–graph. Thus, given existing smaller Ramsey graphs, Ramsey graphs for larger parameter situations could be calculated.

The techniques developed utilized systems of linear equations reflecting properties of the graphs under inspection. Two of the system's equations reflected the order and size of the graph under consideration. The system's final equation was an inequality which provided a minimum possible value for the graph's size. This inequality was based upon knowledge of certain smaller Ramsey graphs.

In the same paper, the authors also determined the uniqueness of the minimum graph for the parameter situation $(3,5,11)$, and calculated various values of $e(3,6,n)$ and $e(3,7,n)$. Although the value of $R(3,6)=18$ had already been established by Kéry in 1964 and Kalbfleisch in 1966, the authors used various techniques which they had developed to reestablish this fact. From the computations performed they were also able to establish the Ramsey number $R(3,7)=23$. [GRA68a, KER64a, KAL66a, WIN88a]

Graver and Yackel improved the asymptotic bounds for $R(k,l)$ with $k\geq 3$ by deriving the inequality $R(k,l)\leq cl^{k-1}\log\log l/\log l$, for $k\geq 3$. The asymptotic

bounds for triangle−free Ramsey numbers were thus improved to $R(3,l) = o(y^2)$. [GRA68a]

The theorems established by Graver and Yackel for building Ramsey graphs from smaller existing Ramsey graphs formed the starting point for future graph enumerations by Grinstead and Roberts, as well as Radziszowski and Kreher. These theorems are also the basis for the Ramsey graph enumeration algorithms implemented in this thesis. [GRA68a]

While at Dartmouth College, Grinstead and Roberts discovered additional values of $e(k,l,n)$ utilizing the previously described Ramsey graph enumeration theorems established by Graver and Yackel. In their 1982 paper *"On the Ramsey Numbers* $R(3,8)$ *and* $R(3,9)$*"*, Grinstead and Roberts constructed all $(3,l,n,e)$−graphs for the parameter situations $(3,5,11,16)$, $(3,6,15,26)$, and $(3,6,16,32)$. The number of graphs enumerated were 6, 7, and 5 respectively. For the $(3,7)$−graph situation the authors enumerated 15 $(3,7,20,44)$−graphs, 4 $(3,7,21,51)$−graphs, and a unique $(3,7,22,60)$−graph. [GRI82a]

In addition to these graph enumerations, the authors were able to establish the minimum size and bounds for the following Ramsey graphs instances: $e(3,7,19) = 37$, $e(3,7,21) = 51$, and $e(3,7,27) \geq 81$. All of these calculations were improvements upon the values computed by Graver and Yackel. The authors utilized a Honeywell Level 66 computer for the execution of their algorithms. [GRI82a]

Their results also included an improvement in the bounds for the parameter situation $28 \leq R(3,8) \leq 29$, and the calculation of the Ramsey number $R(3,9) = 36$. [GRI82a]

In the 1988 paper *"On* $(3,k)$ *Ramsey Graphs: Theoretical and Computational Results"*, Radziszowski and Kreher expanded upon work performed by Grinstead and Roberts. The authors generalized the algorithms presented by Grinstead and Roberts in [GRI82a].

A catalogue of all $(3,3)$, $(3,4)$, $(3,5)$, and $(3,6)$ Ramsey graphs was created using a vertex addition algorithm. The first step of the process was the execution of a brute force method for the enumeration of all $(3,l,n)$−graphs with $n < l$. These graphs formed a set of *base* graphs. The number of such graphs is small and therefore the computations and checks of resulting graphs are feasible. For a range of calculated degree values d, a d−vertex was attached to various independent sets within the previously enumerated base graphs. This method was utilized to build larger $(3,l)$ which comprised the catalogue.

Since a complete catalog of $(3,6)$−graphs existed, the generalized algorithms of Grinstead and Roberts were implemented and utilized for the enumeration of all minimum $(3,7,n)$−graphs with $16 \leq n \leq 21$, all $(3,7,n,e)$−graphs with

$16 \leq n \leq 21$ and $e = e(3,7,n)+1$, as well as all $(3,7,22,e)-$graphs with $60 \leq e \leq 66$. [RAD88a]

To accomplish this, the authors utilized linear programming techniques to implement an algorithm (referred to as the DELTA algorithm) for the calculation of all *possible* valid Ramsey graph degree sequences. The authors implemented the EXPAND algorithm to execute the generalized graph enumeration process. This algorithm utilized the degree sequences produced by the DELTA algorithm as well as the complete catalogue of $(3,6)-$graphs to produce the described $(3,7)-$graphs. [RAD88a]

In the same paper, Radziszowski and Kreher analyzed various structural properties of $(3,l)$ graphs, such as graph size. From this they were able to derive an equation for the minimum number of edges in any $(3,l,n)-$graph for $n \leq 3(l-1)$. In other words a partial formula for the function $e(3,l,n)$ was established. [RAD88a, RAD91a]

A second 1988 Radziszowski and Kreher paper, *"Upper Bounds for Some Ramsey Numbers R(3,k)"*, utilized the same DELTA and EXPAND algorithms from the author's prior 1988 paper. This second paper provided results for the enumeration of various triangle−free Ramsey graphs including 396 $(3,8,25,65)-$graphs, 62 $(3,8,26,73)-$graphs, and 4 $(3,8,27,85)-$graphs. The paper also included an improvement in the equation for the partial function $e(3,l,n)$. [RAD88b]

The various $(3,8)$ Ramsey graph enumerations by Radziszowski and Kreher led to the improvement in the upper bounds for the following Ramsey number situations: $R(3,10) \leq 43$, $R(3,11) \leq 51$, and $R(3,12) \leq 60$. [RAD88b]

In 1993, Chung provided an explicit construction for the parameter situation $R(3,4k+1)$ and showed it to have a value of at least $6R(3,k+1)-5$ for all $k \geq 1$. [CHU93a]

In 1994 Piwakowski improved the lower bounds for the Ramsey numbers $R(3,13) \geq 58$ and $R(5,8) \geq 94$. [PIW94a, PIW93a]

In 1995, Kim obtained a breakthrough by proving that a given Ramsey number $R(3,l)$ has an order of magnitude of exactly $\Theta(l^2/\log l)$. This was an improvement over the result established by Graver and Yackel. [GRA68a, KIM95a]

In the same year Brendan McKay and Stanisław Radziszowski established the Ramsey number $R(4,5) = 25$. [MCK95a]

Since the results reported by Radziszowski and Kreher, no known additional calculations involving the enumeration of $(3,7)$, $(3,8)$ or $(3,9)$ Ramsey graphs have been performed. [RAD88a, RAD88b]

The John Winn book *"Asymptotic Bounds for Classical Ramsey Numbers"*, although somewhat dated (1988), contains a wealth of information regarding

various asymptotic bounds for Ramsey numbers. [WIN88a]

Stanisław Radziszowski maintains a periodically updated *living document* which contains the current known bounds and values for various Ramsey numbers. This document is maintained at the web based *Electronic Journal of Combinatorics* can be accessed via the URL *http://www.combinatorics.org/*. [RAD01a]

1.5. Thesis Outline

This section provides a brief overview of what is to be presented in the following chapters of this document.

1.5.1. Goals

The goals of this thesis include the following.

Theoretical Overview

- Provide an overview of necessary general graph theoretical definitions and theorems.
- Provide a theoretical explanation of the techniques needed for Ramsey graph enumerations, including definitions and theorems specific to Ramsey graphs.

Software Libraries

- Implement an ANSI C UNIX® library for generic graph manipulation.
- Implement an ANSI C UNIX library for Ramsey graph manipulation.

Software Tools

- Implementation of a set of UNIX command line utilities for the manipulation, enumeration, and graphical display of generic graphs and Ramsey graphs.

Graph Enumerations

- Enumeration of all $(3,3)$, $(3,4)$, $(3,5)$, and $(3,6)$ Ramsey graphs.
- Enumeration of all minimum $(3,7,n)$ Ramsey graphs with $n = 16..21$, all $(3,7,n,e)$ Ramsey graphs with $n = 16..21$ and $e = e(3,7,n)+1$, and all critical $(3,7,22)$ Ramsey graphs.
- Enumeration of some minimum $(3,8,n)$ Ramsey graphs with $n = 18..26$, and some critical $(3,8,27)$ Ramsey graphs.
- Enumeration of some $(3,9,n)$ Ramsey graphs with $n = 26..31$.

1.5.2. Theoretical Overview

All of the algorithms implemented by the author are based on the work of previous researchers. As described in Section §1.4. (Historical Overview), Graver and Yackel initially established the logic for the graph enumeration process. Grinstead and Roberts expanded upon this work and utilized various computer implemented algorithms for some specific graph enumerations. Radziszowski and Kreher followed with an implementation of a generalized version of the algorithm as well as the calculation of further Ramsey graphs.

The theoretical section of this thesis presents a set of standard graph definitions and theorems which are used throughout this document. Definitions and theorems specific to Ramsey graphs and Ramsey numbers are also provided.

These definitions and theorems are followed with a formal description of the specific Ramsey graph enumeration principles utilized. Intermixed within the formal presentation are comments on how such procedures are implemented algorithmically and utilized for the graph enumeration aspect of this thesis.

1.5.3. Software Libraries

The enumeration of Ramsey graphs requires various general graph manipulation algorithms and algorithms specific to Ramsey graphs. Algorithms for the classical manipulation of graphs as well as specific Ramsey graph operations are implemented using an *object oriented framework*.

A set of objects and methods are implemented as libraries using the ANSI C programming language. The SWIG package from David Beazley is used to create an interface to these libraries. This allows the objects and methods provided to be utilized directly from Perl and Python as well as other scripting languages. These libraries provide a set of building blocks for the implementation of UNIX text and graphic based graph manipulation tools described in the following section. [BEA97a]

The graph objects implemented reflect the classical algebraic graph structures. These include an undirected non−weighted graph object, adjacency, incidence, distance, and characteristic matrix objects, class and class count vector objects, clique and independent set objects, as well as a graph isomorphism mapping object.

Algorithms for common graph related manipulations such as brute force enumeration, union, composition, and intersection are implemented. Routines for the calculation of a graph's distance matrix, characteristic matrix, class and class count vectors are also provided.

Given that Ramsey graph structures are defined based upon the lack of any cliques and lack of any independent sets of a specified size, such structures are

needed. Object methods for the enumeration of all cliques or independent sets contained within a graph are provided.

The graph enumeration methods attempt to calculate all graphs for a specified parameter situation. Given this, isomorphic graphs will be produced. Algorithms for the determination of isomorphic graphs as well as the mapping of the vertices between these graphs are implemented.

The primary structure utilized to represent a graph is that of the adjacency matrix. It is convenient to be able to express a graph's structure in other formats. Given this, methods for the conversion between algebraic graph structures, such as adjacency matrices and incidence matrices are provided.

In addition to the classical graph manipulation methods, algorithms specific to the enumeration of Ramsey graphs are implemented. These include implementations of the generalized DELTA and EXPAND algorithms as presented in the paper [RAD88a].

It is often useful to be able to visualize a graph as a geometrical figure. The graph manipulation library provides routines for the generation of colored images in DVI, PIC (a groff pre–processor language), Postscript, and GIF formats. The image generation routines make no attempt to maximize the graph's aesthetic appeal, such as displaying a graph with the least number of edge crossings.

Chapter 4 of this document provides a detailed description of the objects and methods provided in the graph manipulation library.

1.5.4. Software Tools

A set of UNIX based command line tools for the manipulation of generic graphs and Ramsey graphs is needed to achieve the Ramsey graph enumeration goals of this thesis. A set of such tools is implemented using the ANSI C programming language, as well as the Perl scripting language. The software libraries described in the previous section provide a foundation for the development of such programs. All of the command line tools are designed and implemented using these libraries as a base.

The command line tools implement much of the same same functionality provided by the graph library, albeit in command line form. This allows for easy use of the graph manipulation algorithms from any standard UNIX shell interface.

The tools are designed to read input and write output from and to standard input and standard output respectively, or from and to files. Such input/output manipulation was designed with the explicit goal of allowing the construction of UNIX command pipelines. This feature combined with the tools provided allows a user to type a single command pipeline which enumerates all labeled graphs of a given order, eliminates isomorphisms, and generates Postscript images of the

resulting graphs.

The full array of generic graph and Ramsey graph manipulation tools implemented for this project is presented in Chapter 4 of this document.

1.5.5. Graph Enumerations

This thesis deals explicitly with $(3, l)$ Ramsey graphs. Therefore, all of the graphs calculated are 3−clique or triangle−free.

All $(3,3)$, $(3,4)$, $(3,5)$, and $(3,6)$ graphs are enumerated. These calculations duplicate and verify the results presented in [RAD88a].

All $(3,7,n,e)$−graphs for $n =$ 16, 17, 18, 19, 20, and 21 with $e = e(3,7,n)$ and $e = e(3,7,n)+1$, as well as all critical $(3,7,22,e)$ graphs with $e =$ 60, 61, 62, 63, 64, 65, and 66 are enumerated. These calculations also duplicate and verify the results presented in [RAD88a].

Some $(3,8,n)$−graphs for $n =$ 20, 21, 22, 23, 24, 25, and 26 with $e = e(3,8,n)$ and $e = e(3,8,n)+1$, as well as some previously unknown critical $(3,8,27,e)$ graphs with $e = 85, 86, 87, 88, 89, 90, 91$, and 92 are enumerated.

Some $(3,9,n)$−graphs for $n =$ 26, 27, 28, 29, 30, and 31 are calculated including a newly realized minimum $(3,9,26,52)$ graph. In [RAD88b] the lower bound for the size of a $(3,9,26)$ was presented. The calculation of a graph for this parameter situation substantiates this lower bound as the actual minimum size for the given situation.

Chapter 2. Graph Theory

This chapter provides a summary of commonly used graph terms, definitions, and theorems. Given the variance in graph theoretical terminology, an attempt has been made to use the most widespread terms and definitions. Appendix B provides a complete glossary of graph and Ramsey theory related definitions.

2.1. Graph Theoretical Definitions

Given the topic of this thesis, the most appropriate first definition is that of a graph.

Definition 2.1.1.

An undirected non−weighted graph G is defined by the pair $G = (V(G), E(G))$, for which $V(G)$ is the set of vertices and $E(G)$ is the set of edges. Each element $e \in E(G)$ is an unordered pair (x, y), for which $x, y \in V(G)$. The elements x and y are the endpoints of the edge e.

□

Many variants of the simple graph definition provided above exist. Such additional graph variants include directed graphs, multi−graphs, oriented graphs, pseudo−graphs, and any combination thereof. For the purpose of this thesis, these graph types are not utilized. Therefore for the remainder of this thesis, any reference to a graph unless otherwise noted implies a simple graph with no directed edges (arcs), no multiple edges, no loops, and with non−weighted edges.

A formal method of describing the properties of a graph, such as the number of vertices and edges contained within a graph is required.

Definition 2.1.2.

For a graph G, the order of G is defined as the cardinality of the set $V(G)$. Hence the order of a graph is simply the number of vertices within a graph.

□

Definition 2.1.3.

For a graph G, the size of G is defined as the cardinality of the set $E(G)$. Hence the size of a graph is simply the number of edges within a graph.

□

For the purposes of this thesis both connected and unconnected graphs are considered. If a graph G is *connected*, then for each pair of vertices $x, y \in V(G)$ at least one path connecting x and y exists. If a graph G is *not connected*, then for at least one pair of vertices $x, y \in V(G)$, no path connecting x and y exists.

Each vertex in the set $V(G)$ for a graph G has an associated degree. For the given graph definition presented, this value is simply the number of edges with which the given vertex is incident.

The following definitions present the terms used to denote the smallest and largest degrees present in a given graph.

Definition 2.1.4.

For a graph G, the minimum degree of G, or $\delta(G)$, is defined as the smallest value in the set of degrees of $V(G)$.

□

Definition 2.1.5.

For a graph G, the maximum degree of G, or $\Delta(G)$, is defined as the largest value in the set of degrees of $V(G)$.

□

If the value of $\delta(G) = \Delta(G)$ then the graph G is said to be *regular*.

From a given graph, G, various other graphs can be derived. One such graph is that of a complement.

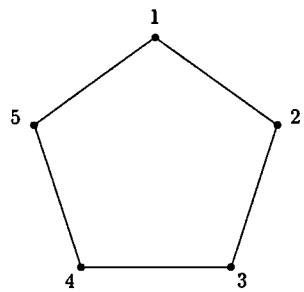
Definition 2.1.6.

The complement of a graph $G = (V(G), E(G))$ of order n, is the graph of order n for which $V(\bar{G}) = V(G)$ and $e \in E(\bar{G})$ iff $e \notin E(G)$.

□

Example 2.1.1.

The following graph G is of order 5, size 5, with $\delta(G) = \Delta(G) = 2$. The vertex set $V(G) = \{1,2,3,4,5\}$, and the edge set $E(G) = \{(1,2),(1,5),(2,3),(3,4),(4,5)\}$. Note that this graph is a cycle of length 5. Such a graph is generally referred to as C_5. The graph C_5 is also self−complementary. This implies that taking the complement of C_5, denoted $\overline{C}_5$, yields C_5 (i.e. the same graph).



□

Various algebraic structures are utilized for the representation of the state of a graph. Such structures include, but are not limited to adjacency matrices, adjacency lists, incidence matrices, and incidence lists. The primary structure utilized throughout this thesis for the representation of a graph's structure is that of an adjacency matrix.

Definition 2.1.7.

An adjacency matrix for an undirected graph, G, of order n is an $n \times n$ symmetric matrix. The elements of an adjacency matrix may have either a value of 0 or a value of 1. For any two vertices i and j the corresponding adjacency matrix value $(i,j) = 0$ indicates that the vertices i and j are not adjacent. A value of $(i,j) = 1$ indicates that i and j are adjacent.

□

Example 2.1.2.

The following is the adjacency matrix for the graph C_5 from Example 2.1.1.

n	1	2	3	4	5
1	0	1	0	0	1
2	1	0	1	0	0
3	0	1	0	1	0
4	0	0	1	0	1
5	1	0	0	1	0

□

This thesis deals explicitly with two−color Ramsey graphs and numbers. As stated earlier this necessitates the use of only undirected graphs. Therefore all of the adjacency matrices utilized are symmetric about the $(1,1)-(n,n)$ diagonal. All of the structural information conveyed in the upper right triangle (i.e. entries above the diagonal) is equivalent to the information contained within the lower left triangle (i.e. entries below the diagonal). Since loops between vertices are not allowed all (i,j) values of the adjacency matrix for which $i = j$ will have a value of 0.

Knowledge of a graph's degrees is useful in the implementation of various graph enumeration algorithms. Given this, a formal structure which maintains various graph vertex degree information is required.

Definition 2.1.8.

A degree sequence for a graph G is an ordered list of the degrees of the elements of the set $V(G)$. These degrees are enumerated in descending order. The sum of the degrees for any graph G is equal to $2\,|E(G)|$. A proof of this is provided by the "First Theorem of Graph Theory".

□

Example 2.1.3.

The following is the degree sequence for the graph C_5 presented in Example 2.1.1.

2 2 2 2 2

□

The degree of any vertex may be gleaned from the corresponding graph's adjacency matrix. For an undirected graph G with adjacency matrix A, the sum of the elements of *row* i is equivalent to the degree of vertex i in G. Note that since only undirected graphs are considered, the sum of the elements of *column* i in A is also equal to the degree of vertex i in G.

The enumeration of Ramsey graphs requires the calculation of graphs without completely connected and completely disconnected sub−graphs for given sizes. Definitions for these structures are provided formally as a clique and an independent set.

Definition 2.1.9.

A set of vertices for a graph G is referred to as a clique if every pair of vertices in the set are adjacent. The clique number $C(G)$ refers to the size of the largest clique contained in a given graph G.

□

Example 2.1.4.

The following is a list of all of the cliques of size 2 in the graph C_5 displayed in Example 2.1.1.

{1,2} {1,5} {2,3} {3,4} {4,5}

□

This thesis deals explicitly with $(3, l)$ Ramsey graphs. The largest clique which can exist in the graphs examined is of size 2. Throughout the thesis cliques of size 3 will also be referred to as *triangles.*

Definition 2.1.10.

A set of vertices for a graph G is referred to as an independent set if no pair of vertices in the set are adjacent. The independence number $I(G)$ refers to the size of the largest independent set contained in a given graph G.

□

Example 2.1.5.

The following is a list of all of the independent sets of size 2 in the graph C_5 displayed in Example 2.1.1.

$$\{1,3\}\ \{1,4\}\ \{2,4\}\ \{2,5\}\ \{3,5\}$$

□

Given the definition of clique and independence number, there is an interesting relationship between a graph and its complement. Specifically, for any undirected graph G, $C(G) = I(\bar{G})$ and $I(G) = C(\bar{G})$.

During any graph enumeration process, graphs which have the same structure but are labeled differently may be produced. A demonstration of this is evident from Example 2.1.1. where it was noted that the graph C_5 and $\bar{C}_5$ are the same graph. Even though the labeling of the vertices in these two graphs may be different, the structure of both is the same.

Definition 2.1.11.

Two graphs are considered isomorphic if there exists a one-to-one and onto relation between the vertices and the edge adjacency mappings of the two graphs.

□

Definition 2.1.12.

Labeled graphs of order n are all graphs which can be enumerated on n vertices without isomorphisms being eliminated. The number of such graphs is $2^{\binom{n}{2}}$.

□

Throughout this thesis counts of the number of graphs which meet various criteria are provided. Unless otherwise specified all such counts are of the number of non−isomorphic graphs.

Isomorphism is an equivalence relation on graphs. Obviously for two graphs to be isomorphic the graphs must share certain *invariant* properties.

Some graph invariants include graph order, graph size, and the graph degree sequences. However, these invariants alone are not enough to prove or disprove isomorphism. A small complete set of simple invariants for graph isomorphism is currently not known.

Given the number of labeled graphs which are enumerated, an isomorphism algorithm is needed to eliminate these duplicate structures. These structures provide

a set of graph invariants for the isomorphism algorithm implemented. This vertex mapping algorithm, which is designated the ISOMAP algorithm, uses vertex classification as a means of determining a possible graph isomorphism and its mapping. The following structures are utilized for the vertex mapping and isomorphism algorithm.

Definition 2.1.13.

The distance matrix for a graph G is defined as the $n \times n$ matrix $D = [d_{ij}]$, where d_{ij} is a positive integer representing the length of the shortest path from vertex v_i to vertex v_j. d_{ii} is defined as 0. For a pair of vertices where no path exists the distance is defined as ∞. [GOU88a]

□

Since only undirected loop−less graphs are considered in this thesis, all distance matrices for a graph of order n will be symmetric about the $(1,1)$ $-(n,n)$ diagonal.

Given a graph distance matrix as defined above, data relating the number of vertices at a given distance from a chosen vertex i may be determined. In addition to this, the number of vertices from which a chosen vertex i is a given distance may also be calculated. Such information is useful in the determination of the existence of an isomorphism between any two given graphs.

Definition 2.1.14.

The column characteristic matrix for a graph G is defined as the $n \times (n-1)$ matrix $X_c = [c_{ij}]$, where c_{ij} is the number of vertices from which v_i is at distance j. [GOU88a]

□

Definition 2.1.15.

The row characteristic matrix for a graph, G, is defined as the $n \times (n-1)$ matrix $X_r = [r_{ij}]$ where r_{ij} is the number of vertices at a distance j from vertex v_i. [GOU88a]

□

The column characteristic matrix and the row characteristic matrices are both derived from the distance matrix. These two matrices provide characteristics of a graph which are used to partition the vertices into classes. These classes are used to form the group of graph invariants which are necessary for the isomorphism mapping algorithm.

Although this thesis deals explicitly with undirected graphs, the algorithms and software are designed to be utilized for undirected as well as directed graphs.

The column and row characteristic matrix calculations for an undirected graph produce the exact same matrix in all situations. For a directed graph the column and row characteristic matrix calculations may produce different matrices. Therefore both column and row matrix definitions are provided and both matrix calculations are demonstrated in all examples.

Definition 2.1.16.

The characteristic matrix for a graph G is defined as the $n \times n$ matrix $X = [x_{ij}]$ calculated from the composition of the respective column characteristic matrix, $X_c = [c_{ij}]$, and the row characteristic matrix, $X_r = [r_{ij}]$. For $X = [x_{ij}]$, $x_{ij} = c_{ij} << 8 + r_{ij}$. [GOU88a]

□

Note that for the purpose of this thesis all of the values contained within the column and row characteristic matrices are relatively small. All such values are well under the upper limit provided by 8 bits. The shift value of 8 bits for the composition function is adequate.

Each row of the matrix defines what is known as a *vertex class.* Each identical row of the characteristic matrix is assigned to the same class. The minimum number of classes which a graph of order n may exhibit is 1 and the maximum is n. For the graph C_5 presented in Example 2.1.1. only one class exists, given that all vertices are of degree 2 and that all of the values of each row of the column and row characteristic matrices are identical.

Definition 2.1.17.

The class vector for a graph G of order n is defined as an n length vector $C = [c_i]$, for which c_i is the value of the class of vertex i of graph G. [GOU88a]

□

Definition 2.1.18.

The class count vector for a graph G of order n is defined as an n length vector $CC = [cc_i]$, for which cc_i is the value of the number of vertices of class i of graph G. [GOU88a]

□

This chapter provides a summary of the most commonly used graph theoretical structures utilized throughout this document. Knowledge of such definitions, theorems, and structures is required for the development of the Ramsey graph

enumeration theory presented in Chapter 3, as well as the description of the algorithm implementation presented in Chapter 4.

Chapter 3. Ramsey Graphs

This chapter presents a theoretical overview of the Ramsey graph enumeration process. The initial theorems developed for this specific process were first described by Graver and Yackel. Grinstead and Roberts utilized various computer implemented algorithms to enumerate Ramsey graphs for specific scenarios. Radziszowski and Kreher continued the use of these same algorithms and computed a complete catalogue of $(3, l)$ Ramsey graphs for $l \leq 6$ as well as some $(3, 7)$ and $(3, 8)$ graphs. [GRA68a, GRI82a, RAD88a, RAD88b]

In this chapter, a formal definition of a Ramsey graph is provided. Various structural aspects of Ramsey graphs are investigated. Examples of these properties are presented and also demonstrated pictorially.

The definitions and theorems presented in this chapter provide the background necessary for the comprehension of the Ramsey graph enumeration algorithms documented in Chapter 4.

3.1. Properties of Ramsey Graphs

The desired result of the enumeration of Ramsey graphs is the calculation of exact values or the improvement of bounds for various Ramsey number situations. Given this a formal definition of a Ramsey graph is required.

Definition 3.1.1.

A Ramsey graph G is defined as an undirected non−weighted graph with $C(G) < k$ and $I(G) < l$, and is denoted (k, l)−graph. If G has n vertices and e edges, then G may be denoted as a (k, l, n)-graph, or a (k, l, n, e)-graph, respectively. [RAD88a]

□

Definition 3.1.1. provides a general description of all Ramsey graphs. The *largest* Ramsey graphs for a given (k, l) parameter situation are called critical graphs. These graphs are of considerable importance given that the existence of a (k, l, n) graph eliminates any value n' with $n' \leq n$ as a possible solution for the situation $R(k, l)$.

Definition 3.1.2.

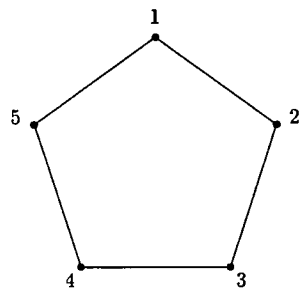
A critical Ramsey graph, denoted (k, l, n), is defined as a (k, l) Ramsey graph for which the graph's order n is equal to the value $R(k, l) - 1$. [RAD88a]

□

Example 3.1.1.

Pictured below is a valid $(3,3,5)-$*graph. It is the only valid Ramsey graph with no cliques of size* 3*, no independent sets of size* 3*, with order 5. The existence of this graph along with the lack of the existence of any* $(3,3,6)-$*graphs leads to the establishment of the value of the elementary Ramsey number* $R(3,3)=6$*.*

Because the order of this graph is one less than the value of the corresponding Ramsey number (i.e. the order is 5 *while* $R(3,3)=6$*) this graph is a critical Ramsey graph. Note that this graph is the only critical* $(3,3)-$*graph.*



□

One of the interesting properties of Ramsey graphs is the relationship between the graph and its complement.

Lemma 3.1.2.

Let G *be a Ramsey graph and* $\overline{G}$ *be the complement of the graph* G*. Given this and the definition of graph complement,* $C(\overline{G})=I(G)$ *and* $I(\overline{G})=C(G)$*, and therefore* G *is an* $(k,l)-$*graph if and only if* $\overline{G}$ *is a* $(l,k)-$*graph.* [GRA68a]

□

Example 3.1.2.

In Example 3.1.1. the unique critical $(3,3,5)-$*graph was presented. This graph also has the property of being the graph* C_5*. The graph* C_5 *is self-complementary, indicating that* $\overline{C}_5$ *is structurally equivalent to* C_5*. The complement of the* $(3,3,5)-$*graph therefore yields the isomorphically equivalent* $(3,3,5)-$*graph.*

□

One of the properties examined by Graver and Yackel was that of the partitioning of a Ramsey graph into sub−graphs. Given a Ramsey graph G and a selected

vertex v, all of the vertices attached to v in G form a sub−graph, as do all of the vertices not attached to v in G. This single vertex along with the sub−graph of its neighbors and the sub−graph of its non−neighbors form an important partitioning.

Definition 3.1.3.

If G is a $(k,l)-$graph then any given vertex $v \in V(G)$ may be chosen as the preferred vertex. [GRA68a, GRI82a]

□

The above statement provides a definition for the *preferred* vertex. Note that this vertex may be chosen from any vertex in the given Ramsey graph. Given the selection of the preferred vertex, the remaining two previously described induced sub−graphs are automatically defined.

Definition 3.1.4.

If G is a $(k,l)-$graph, then for any given preferred vertex v, the sub−graph $H_1(v)$ is defined as the graph induced from the neighbors of the vertex v. [GRA68a, GRI82a]

□

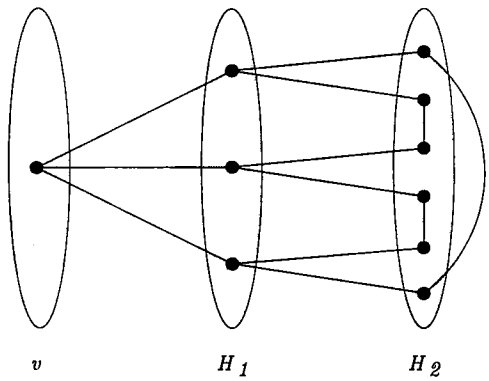
Definition 3.1.5.

If G is a $(k,l)-$graph, then for any given preferred vertex v, the sub−graph $H_2(v)$ is defined as the graph induced from the non−neighbors of the vertex v. [GRA68a, GRI82a]

□

Example 3.1.3.

The following graph is of the unique $(3,5,10,12)-$*graph. The vertex* v*, and the two sub−graphs* $H_1(v)$*, and* $H_2(v)$ *are displayed. Note that the* $H_1(v)$ *and* $H_2(v)$ *sub−graphs can only be determined after the preferred vertex* v *has been selected.*



□

In partitioning a Ramsey graph into the three sub−graphs defined above, various structural properties may be readily observed. One such property is provided by the following theorem.

Theorem 3.1.1.

If G *is a* $(k,l)-$*graph and* v *is a vertex in* G*, then* $H_1(v)$ *is a* $(k-1,l)$*-graph and* $H_2(v)$ *is a* $(k,l-1)$*-graph.* [GRA68a, GRI82a]

□

In the graph G, the vertex v is attached to all of the vertices in $H_1(v)$. Any clique of size x in $H_1(v)$ becomes a clique of size $x+1$ when v and H_1 are joined.

From the definition of the sub−graph $H_2(v)$, any independent set of size y in $H_2(v)$ becomes and independent set of size $y+1$ when v is added.

For all $(3,l)-$graphs, the $H_1(v)$ sub−graph will have a size of 0. A size of 1 or greater would imply that an edge existed between at least two vertices of the $H_1(v)$ sub−graph. Since this same edge would also be present in the graph G, by the definition of the preferred vertex v a triangle would be formed among v and two vertices of $H_1(v)$. This is impossible given that a $(3,l)-$graph is triangle free.

Example 3.1.4.

In Example 3.1.3. a pictorial representation of the unique $(3,5,10,12)-$*graph is presented. In this example the vertex* v *and the* $H_1(v)$ *and* $H_2(v)$ *sub−graphs are visually segregated. It is easily verifiable that the* $H_1(v)$ *sub−graph in Example 3.1.3. is a* $(2,5,3)-$*graph, and therefore contains no cliques of size* 2 *and no independent sets of size* 5. *In the same example the* $H_2(v)$ *sub−graph is a* $(3,4,6)-$*graph with no cliques of size* 3 *and no independent sets of size* 4.

The above definitions, theorems, and examples provide important information regarding the structure of Ramsey graphs. The fact that Ramsey graphs can be partitioned into *smaller* Ramsey graphs can be utilized. Given knowledge of $(k-1,l,n)$ and $(k,l-1,n)$ Ramsey graphs, an algorithmic process can be developed to enumerate $(k,l,n+n')-$graphs, where n' ranges over a set of pre−determined values.

For the $(3,l,n)-$graph case it was noted above that the $H_1(v)$ sub−graph contains no edges. Therefore, for a given $(3,l,n)-$graph and a preferred vertex v, the order and size of the $H_1(v)$ sub−graph are defined by the degree of v and 0 respectively.

The determination of the $H_2(v)$ sub−graph is somewhat more difficult. Knowledge of $(3,l-1)-$graphs is needed. If a set of such graphs exists (i.e. has already been enumerated), a set of parameters can be utilized as inputs to an enumeration algorithm in an attempt to attach an existing $(3,l-1)-$graph to a $(2,l)-$graph and a single vertex v. This is a reverse of the partitioning process of a Ramsey graph into the vertex v, and sub−graphs $H_1(v)$ and $H_2(v)$ described earlier. The resulting graph may possibly be a valid $(3,l)-$graph. Such techniques provide a feasible method for Ramsey graph enumeration as opposed to brute force methods.

Knowledge of other Ramsey graph structural parameters is required for the implementation of the enumeration method. One such datum is that of the *possible* valid degree sequences of Ramsey graphs.

Given the definition of a Ramsey number and the relationship between a Ramsey graph and its complement's clique and independence numbers, bounds for the minimum and maximum degree of any vertex of a Ramsey graph can be determined based upon known *smaller* Ramsey number values. [GRA68a]

Lemma 3.1.5.

If G is a $(k,l)-$graph of order n then $(n-1)-[R(k,l-1)-1]$ is the minimum possible degree for a vertex of G, and $R(k-1,l)-1$ is the maximum possible degree for a vertex of G. [GRA68a]

□

Proof.

Let $d(v)$ be the degree of the vertex v and let $H_1(v)$ be the sub−graph of G spanned by the vertices of G which are joined to v by an edge. Clearly $I(H_1) \leq I(G) \leq l$, since any set of points independent in $H_1(v)$ is also independent in G. If K is a complete sub−graph of $H_1(v)$ of order n, the graph spanned by K and v is a complete sub−graph of G on n points. Therefore

$$C(H_1) < C(G) - 1 < k - 1$$

Therefore $H_1(v)$ is an $(k-1,l)-$graph. Given this it is obvious that $d(v) \leq R(k-1,l)$. Using the above argument and Lemma 3.1.2. the value $\overline{d}(v)$, which is the degree of v in $\overline{G}$, is less than or equal to $R(k,l-1)$. However $d(v)+\overline{d}(v) = n-1$ leading to the following statement.

$$d(v) \geq (n-1) - [R(k,l-1)-1]$$

[GRA68a]

□

Knowledge of the possible minimum and maximum valid degrees for a Ramsey graph is useful for the implementation of Ramsey degree sequence enumeration algorithms.

The theoretical description of the enumeration algorithms makes use of the number of vertices of a given degree in a given Ramsey graph. A formal method for denoting the vertex degrees and the counts of such degrees is required.

Definition 3.1.6.

Given a $(3,l)-$graph G, v_i is defined as $v_i = l-1-i$, and s_i is defined as the number of vertices in G of degree v_i. [GRI82a]

□

For a $(3,l)-$graph, $H_1(v)$ is an independent set for each vertex v. Therefore in definition 3.1.6. EF where $i \geq 0$, the maximum possible degree in a $(3,l)-$graph is $l-1$. The subscript i is the difference between the degree of the vertex and the maximum possible degree in G. The value of v_i depends on l as well on i.

The Ramsey graph enumeration algorithms implemented require that $(3, l-1)-$graphs are *known* for the enumeration of $(3, l)-$graphs. The process of joining the preferred vertex v to the sub−graph $H_1(v)$ is trivial. There are no edges in the sub−graph $H_1(v)$ and v is attached to every vertex in $H_1(v)$. Therefore $v + H_1(v)$ is simply a *star* graph.

The $v + H_1(v)$ sub−graph must be joined with the $H_2(v)$ sub−graph. Determination of how the vertices of $v + H_1(v)$ should be attached to the vertices of $H_2(v)$ requires further structural knowledge of the $H_2(v)$ sub−graph, and Ramsey graphs in general.

Definition 3.1.7.

If a vertex v is preferred in a (k, l)-graph, $Z(v)$ is defined to be the sum of the degrees of the neighbors of v. If $Z(v) = s$, we state that v has Z-sum s. [GRI82a]

□

The $Z-$sum is simply the sum of the degrees of vertices in G which are members of the $H_1(v)$ sub−graph. As has been stated, the $H_2(v)$ sub−graph is a $(k, l-1)$ Ramsey graph. If the $(k, l-1)-$graph is a minimum graph for the $(k, l-1)$ parameter situation, the vertex v is denoted by the term *full.*

Definition 3.1.8.

A $d-$degree vertex v in a $(k, l, n)-$graph is denoted "full" if $|H_2(v)| = e(k, l-1, n-d-1)$, where $|H_2(v)|$ denotes the size of the $H_2(v)$ sub−graph. [GRI82a]

□

Knowledge of the number of edges in the $H_2(v)$ graph is therefore useful information. Specifically evaluation of the (k, l) Ramsey graphs which contain a minimum or maximum number of edges for the parameter situation under consideration is a valuable resource.

Definition 3.1.9.

If G is a $(3, l, n, e)-$graph with $e = e(3, l, n)$, then G is called a minimum $(3, l, n)-$graph. Let G be a $(3, l, n)-$graph with the property that if any edge is removed, the resulting graph contains an independent set of size l. In this case we call G a minimal $(3, l, n)-$graph. [RAD88a]

□

Graver and Yackel developed a method for determining the minimum number of edges in a Ramsey graph. This method requires the knowledge of smaller

parameter situation Ramsey graph sizes.
The following theorem was presented in [GRA68a].

Theorem 3.1.2.

Let G be a $(k,l,n,e)-$graph, and define

$$\Delta = ne - \sum_{i\geq 0} \{e(3,l-1,n-v_i-1) + v_i^2\} s_i \qquad (3.1.1)$$

then $\Delta \geq 0$ and G has at least $n-\Delta$ full vertices. [GRA68a, GRI82a]
□

Proof.

For all i and j, define

$$\beta_{ij} = \begin{cases} |\{w: vw \in E \text{ and } deg(w) = v_j\}| & \text{if } deg(v) = v_i \\ 0 & \text{Otherwise} \end{cases} \qquad (3.1.2)$$

The value of $\sum_v \beta_{ij}(v)$ is the number of edges between vertices of degree v_i and v_j, and therefore $\sum_v \beta_{ij}(v) = \sum_v \beta_{ji}(v)$. If v is of degree v_i, and is preferred, then the size "e" of a Ramsey graph is given by the following.

$$e = |H_2(v)| + (v_i)^2 + \sum_{j\geq 0} (i-j)\beta_{ij}(v) \qquad (3.1.3)$$

If all of the vertices of G are summed the following equation holds. Note that n is the order and e the size of the Ramsey graph for the desired situation.

$$ne = \sum_v |H_2(v)| + (v_i)^2 s_i + \sum_{i\geq 0} \sum_v \sum_{j\geq 0} (i-j)\beta_{ij}(v) \qquad (3.1.4)$$

In all cases the following holds.

$$|H_2(v)| \geq e(3, l-1, n-v_i-1) \tag{3.1.5}$$

Thus the value of $e(3, l-1, n-v_i-1)$ *for a given Ramsey graph* $(3, l-1, n-v_i-1)$ *provides a lower bound for the size of the* $H_2(v)$ *sub–graph. If* v *is a full vertex the previous equation is an equality. If* i *and* j *are fixed, then the sum* $\sum_v \beta_{ij}(v)$ *occurs in Equation 3.1.5. with a coefficient of* $(i-j)$*, and the sum* $\sum_v \beta_{ji}(v)$ *occurs with a coefficient of* $(j-i)$*. These two sums cancel. By substituting the value of* $H_2(v)$ *from Equation 3.1.5. into Equation 3.1.4. and simplifying the following equation is derived.*

$$\begin{aligned} ne &\geq \sum_{i\geq 0} e(3, l-1, n-v_i-1)s_i + (v_i)^2 s_i \\ ne &\geq \sum_{i\geq 0} \{e(3, l-1, n-v_i-1) + (v_i)^2\} s_i \end{aligned} \tag{3.1.6}$$

This is equivalent to the statement that $\Delta \geq 0$*, and it is easily shown that each vertex which is not full contributes at least 1 to* Δ*. Therefore there must be at least* $(n-\Delta)$ *full vertices in the given Ramsey graph.* [GRA68a, GRI82a]

□

Given this, the number of edges which connect the $H_1(v)$ and $H_2(v)$ graphs need to be counted. The following statement provides information regarding the number of such edges.

Theorem 3.1.3.

In any graph G*, we have* $\sum_{i\geq 0} s_i v_i^2 = \sum_v Z(v)$. [GRI82a]

□

Proof.

$$\begin{aligned}\sum_{v} Z(v) &= \sum_{v}\sum_{w} deg(w) = \sum_{w}\sum_{v} deg(w) \\ &= \sum_{w} (deg(w))^2 = \sum_{i\geq 0} s_i v_i^2\end{aligned} \tag{3.1.7}$$

[GRI82a]
□

Equation 3.1.6. forms the starting point for the implementation of the algorithms which are used for the enumeration of Ramsey graphs. Having knowledge of smaller Ramsey graphs can be useful in implementing algorithms for the construction of larger parameter situation Ramsey graphs.

Using the above equations and some Ramsey graph degree sequence evaluations, equations which provided lower and upper bounds for Ramsey graph sizes have been determined. Knowledge of the minimum and maximum graph sizes is important for the development of Ramsey graph enumeration algorithms.

The following definition introduces a notation used to represent the minimum number of edges for such graphs.

Definition 3.1.10.

The value given by $e(k,l,n)$ is the minimum number of edges in any $(k,l,n)-$graph. The value given by $E(k,l,n)$ is the maximum number of edges in any $(k,l,n)-$graph. The following partial equations for $e(k,l,n)$ were established in the paper "On (3,k) Ramsey Graphs: Theoretical and Computational Results" and "Upper Bounds for Some Ramsey Numbers $R(3,k)$". [RAD88a, RAD88b]

For $l \geq 1$:

$$e(3,l+1,n) \geq 6n - 13l \quad \textit{for all } l,n \geq 1 \tag{3.1.7}$$

For $l \geq 2$:

$$e(3,l+1,n) = \begin{cases} 0 & \textit{if } n \leq l \\ n-l & \textit{if } l < n \leq 2l \\ 3n-5k & \textit{if } 2l < n \leq 5l/2 \end{cases} \tag{3.1.8}$$

For $l \geq 4$:

$$e(3,l+1,n) = \begin{cases} 0 & \textit{if } n \leq l \\ n-l & \textit{if } l < n \leq 2l \\ 3n-5l & \textit{if } 2l < n \leq 5l/2 \\ 5n-10l & \textit{if } 5l/2 < n \leq 3l \end{cases} \tag{3.1.9}$$

For all $l \geq 1$ and $n \geq 1$:

$$e(3,l+1,n) \geq 6n - 13l \tag{3.1.10}$$

with equality holding when $3l \leq n \leq 13l/4 - sign(l \bmod 4)$.

□

A complete proof for the partial equations presented above is available in [RAD88a].

Various corollaries to these theorems for more constrained parameter situations are also provided in the same paper. For example Radziszowski shows that for $n \leq 5l/2$ that the minimum graphs are unique. For $n \leq l$ the minimum graphs are given by n isolated points. For $l < n \leq 2l$ the minimum graphs are given by $2l - n$ isolated points and $n - l$ isolated edges. While for $2l < n \leq 5l/2$ the minimum graphs contain $n - 2l$ pentagons and $5l - 2n$ isolated edges.

The above definitions and theorems form the starting point for various algorithms which are implemented to perform the construction of Ramsey graphs from previously enumerated Ramsey graphs. The details of these algorithms are provided in Chapter 4 of this document.

3.2. Bounds for (3, 8) Ramsey Graphs

The following tables summarize the current known bounds for the $(3, 8, n)$ Ramsey graph parameter situations with $n \geq 22$. The table provides the bounds which were determined by Grinstead and Roberts and later refined by Radziszowski and Kreher. [GRI82a, RAD88b]

Note that these bounds were established before the value of $R(3, 8)$ was determined to be 28. Therefore the largest order for a $(3, 8)-$graph is 27 and no Ramsey graphs exist for the parameter situation $(3, 8, 28)$. The entry for $n = 28$ in the table is kept only for historical reasons to show bounds which were calculated by the respective authors.

n	Grinstead & Roberts	Radziszowski & Kreher
22		42
23		49
24		56
25		65
26	71-74	73
27	81-87	83-85
28†	90-98	94-98

Table 3.2.1. – Bounds and Values of e(3, 8, n), n ≥ 22.

3.3. Bounds of e(3, l, n) for 9 ≤ l ≤ 13 and 3k − 1 ≤ n

The following table presents the currently known upper and lower bounds for triangle free Ramsey graphs. The independent set free values for these graphs range from 8 through 12 inclusive while the orders for these graphs range from 26 through 68 inclusive. The orignal values were for this table were taken from the paper *"Upper Bounds for Some Ramsey Numbers $R(3, k)$"*. [RAD88b]

Since the publication of this paper the bounds have been improved. The table reflects the improved values.

---

† $R(3, 8) = 28$ and therefore no $(3, 8, 28)-$graphs exist.

Values prefixed by a t were obtained by applying Equation 3.1.10. and are therefore larger than those obtained by applying Equation 3.1.1. only. Values prefixed by an s are obtained by evaluation of possible degree sequences for the given graphs. Straightforward checking shows that no Ramsey graph can exist for the given degree sequences. These values are therefore also larger than those obtained by applying Equation 3.1.1. only.

	k						k				
n	9	10	11	12	13	n	9	10	11	12	13
26	t 52					48			214	172	144
27	59					49			229	184	153
28	67					50			243	196	162
29	75	t 57				51				208	172
30	84	63				52				221	s 182
31	s 93	70				53				233	193
32	103	77	t 62			54				248	204
33	114	85	68			55				262	216
34	125	94	75			56				276	227
35	136	103	81	t 67		57				291	239
36		113	88	t 73		58				306	252
37		123	96	79		59				322	s 266
38		133	104	86	t 72	60					280
39		145	113	93	t 78	61					294
40		156	122	100	84	62					308
41		169	132	108	91	63					324
42		182	143	115	97	64					339
43			153	124	104	65					354
44			165	132	112	66					371
45			177	142	120	67					389
46			189	152	128	68					406
47			201	162	136						

Table 3.3.1. – Bounds for e(3,l,n) for $9 \leq k \leq 13$ and $3k - l \leq n$.

3.4. Bounds for Ramsey numbers R(3,l), with $10 \leq l \leq 14$

The following table provides the currently known lower and upper bounds for the Ramsey numbers $R(3,l)$ with $10 \leq l \leq 14$. These values were originally taken from the paper *"Upper Bounds for Some Ramsey Numbers $R(3,k)$"*. [RAD88b]

The values for these parameter situations have been improved since the original publication of this paper. This table reflects the updated values.

Note that all $R(3,l)$ Ramsey number values for $l < 10$ have been computed and therefore are not presented in this table.

l	Lower Bound	Upper Bound
10	40	43
11	46	51
12	52	59
13	59	69
14	66	78

Table 3.4.1. – Bounds for Ramsey numbers $R(3,l)$, $10 \leq l \leq 14$.

The graph structural properties described in this section can be effectively utilized in the implementation of algorithms for the enumeration of Ramsey graphs. The next chapter describes how such algorithms are implemented as well as the software and hardware platforms utilized for the enumeration process.

Chapter 4. Implementation

This chapter details the various data formats, database formats, and software packages implemented for the enumeration of Ramsey graphs. A description of third party software packages used in addition to the custom software packages for various graph operations is also presented.

All object structures and pseudo–code segments documented in this section are based on the syntax of the ANSI C Programming Language. [KER78a] In the given pseudo–code segments all user defined variables and functions are formatted in *italic* font. All language defined reserved words are formatted in **bold** font.

This chapter is segregated into the following sections.

- Graph Data Formats

  This section details the data formats used to represent graphs and graph properties.

- Graph File Formats

  This section details the database and file formats utilized to maintain archives of graphs and Ramsey graphs.

- Algorithms

  This section describes the data structures and pseudo–code designed and implemented for the enumeration of various graphs and Ramsey graphs.

- Custom Software Packages

  This section contains detailed descriptions of the software packages implemented for the enumeration of various graphs and Ramsey graphs.

- Third Party Software Packages

  This section contains a description of the various third party software packages utilized.

4.1. Graph Data Formats

This section details the formats which are used as input and produced as output by the various software packages implemented for this thesis and the enumeration of Ramsey graphs. All formats are ASCII based and therefore may be edited/viewed with any standard text editor.

All of these formats are generic in the sense that they are not specific to Ramsey graphs. The use of these tools may be applicable to other graph theoretical problems.

4.1.1. Graph State Formats

This section describes the formats which are used to represent the state of a graph. The order of the graph as well as all of the graph adjacencies are maintained in such encodings.

The primary format used to represent graphs and Ramsey graphs throughout this thesis is the *y–format*. By default, all of the software utilities implemented accept as input and produce as output y–format graphs.

Various software library routines and tools have the capability to input other formats representing the state of a graph. These formats include the adjacency matrix, the adjacency list, and the incidence matrix. Software utilities are available for the conversion of these various formats from and to the y–format.

This section contains a description of each of the state formats as well as various pseudo–code listings of routines used for input and output of each format.

4.1.1.1. Graph y–format

All graphs utilized by the thesis software are stored in what is defined as the *y–format*. This is an ASCII encoding of the adjacencies and non–adjacencies present in a given undirected, loop–less, non–weighted graph (i.e. a simple graph).

The y–format for a graph can easily be determined from the graph's adjacency matrix. Since a graph's adjacency matrix is symmetrical, for a given graph G of order n only the adjacency matrix elements (i, j) for which $\{i: 1 \leq i \leq n\}$ and $\{j: i < j \leq n\}$ are encoded in the y–format.

The y–format is composed of the concatenation of a *header* byte and a series of bytes which encode the adjacencies between the nodes of the given graph. The header byte contains the inclusive or of the graph order with the value 01000000_2. The remaining bytes contain the adjacency values for each pair of vertices in the graph. These bytes are referred to as the *body* bytes. Only the lower 6 bits of a byte are used to contain adjacency/non-adjacency values.

The upper two bits of y–format bytes are always set to the value '01'. This restricts the possible ASCII values used for the encoding to those of the printable character range, $64_{10} - 127_{10}$.

The DELETE character has an ASCII value of 127_{10}. This value is not easily displayed as a single character. Typically on UNIX® terminals it is displayed as the character pair ^?. Obviously each of these two characters (^ and ?) are valid

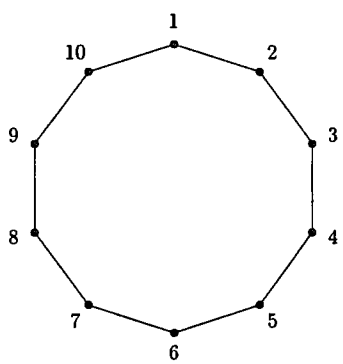
ASCII characters. Given this, a possible ambiguity may develop when displaying a graph's y−format representation in a *paper* document. Given this, the dagger symbol (†) is substituted for the DELETE character in this document when a y−format string is displayed.

Each of the bits which is contained in the lower 6 bits of the *body* bytes represents an adjacency/non−adjacency between two vertices. A 1−bit represents an adjacency while a 0−bit represents a non−adjacency. A graph of order n is represented in a symmetrical matrix of $n \times n$ elements. The number of bits required to represent the adjacencies of a n−order graph is $(n^2 - n)/2$.

Each byte of the y−format body can maintain 6 adjacencies. A total of $\lceil (n^2 - n)/(2)(6) \rceil + 1$ bytes is needed to store the adjacency information for an n−order graph. The +1 is for the header byte. The graph adjacencies are stored in row−major form. For a graph of order n all of the elements of column j are stored followed by all of the elements of column $j + 1$ where $\{1 \leq j < n\}$.

The y−format encoding scheme was designed by Brendan McKay of the Australian National University. His *gtools* package as well as this author's *grama* and *ytool* packages can be used to convert graphs from other formats to the y−format and vice−versa.

Figure 4.1.1.1.1. displays a graph of order 10 composed of a single cycle of length 10, also known as the C_{10} graph.



J	i	D	H	H	D	A	'	H

Figure 4.1.1.1.1. − C_{10} Graph and Corresponding y−format String.

The following figure represents the y-format encoding for the above graph C_{10} with a description of each of the bytes in the encoded format. As can be seen the low order byte of the header byte ('J') has a value of 10 which represents the order of the graph. The first two body bytes ('i' and 'D') have been expanded to show the value of the adjacencies which are maintained by each byte.

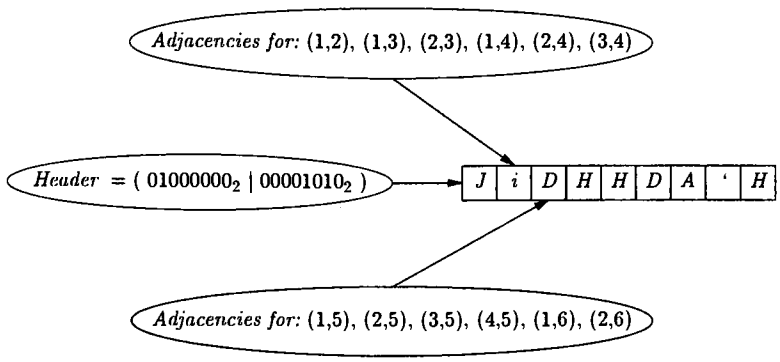


Figure 4.1.1.1.2. – C_{10} Graph y–format Encoding.

Tables 4.1.1.1.1. and 4.1.1.1.2. display expanded data regarding the encoding of the C_{10} graph. The first table shows the details regarding the header byte ('J') which maintains the order of the graph, in this case 10. The second table shows the details for all of the body bytes which maintain the adjacencies between the vertices of the C_{10} graph. The ASCII, hexadecimal, and binary value of each byte is shown, followed by a list of the adjacencies which it maintains. Vertex pairs which are formatted in ***bold–italic*** font designate vertices which are adjacent in the graph C_{10}.

The following is a list of the adjacencies which are represented by each of the bytes (characters) of the y–format string.

y–format Graph Order Encoding			
ASCII	*Hexadecimal*	*Binary*	*Order*
J	$4A_{16}$	0100 1010_2	10

Table 4.1.1.1.1. – C_{10} Graph y–format Header Byte Encoding Detail.

y–format Graph Adjacency Encoding								
ASCII	*Hexadecimal*	*Binary*	*Adjacencies*					
i	69_{16}	$0110\ 1001_2$	***(1,2)***	(1,3)	***(2,3)***	(1,4)	(2,4)	***(3,4)***
D	44_{16}	$0100\ 0100_2$	(1,5)	(2,5)	(3,5)	***(4,5)***	(1,6)	(2,6)
H	48_{16}	$0100\ 1000_2$	(3,6)	(4,6)	***(5,6)***	(1,7)	(2,7)	(3,7)
H	48_{16}	$0100\ 1000_2$	(4,7)	(5,7)	***(6,7)***	(1,8)	(2,8)	(3,8)
D	44_{16}	$0100\ 0100_2$	(4,8)	(5,8)	(6,8)	***(7,8)***	(1,9)	(2,9)
A	41_{16}	$0100\ 0001_2$	(3,9)	(4,9)	(5,9)	(6,9)	(7,9)	***(8,9)***
‘	60_{16}	$0110\ 0000_2$	***(1,10)***	(2,10)	(3,10)	(4,10)	(5,10)	(6,10)
H	48_{16}	$0100\ 1000_2$	(7,10)	(8,10)	***(9,10)***			

Table 4.1.1.1.2. – C_{10} Graph y–format Body Encoding Detail.

Figure 4.1.1.1.3. presents the adjacency matrix for the graph C_{10}. The darker shaded elements within the matrix represent the elements on the $(1,1)$ to (n,n) diagonal where $n = 10$. The unshaded elements of the matrix represent the upper triangle of the symmetrical matrix. These are the adjacencies which are stored in the y–format representation of the graph.

This demonstrates the order in which the *adjacency pairs* from the adjacency matrix are *encoded* into the y–format string. Each arrow starting with the arrow labeled 1 should be followed. The chain of *adjacencies* created provides the order for the encoding. The † marks the starting column while the ‡ marks the terminating column.

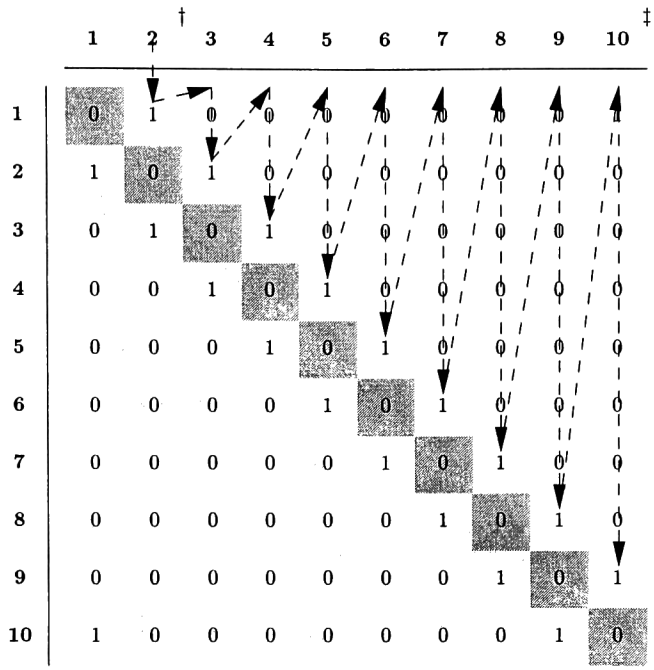


Figure 4.1.1.1.3. – C_{10} Adjacency Matrix.

Listing 4.1.1.1.1. presents a pseudo–code segment which generates the y–format graph representation from the corresponding adjacency matrix. The *order* variable contains the order of the graph for which the y–format has been generated. The *attached()* function returns a value of 1 if the given vertices denoted by the indices i and j are adjacent, and a value of 0 if they are not adjacent. The *yformat* variable is a pointer to a pre–allocated area of memory which is large enough to store the generated y–format object representing the given graph.

```
/* Set the order of the graph */
*yformat++ = ( 0100 | order );
c = 32; v = 0100;
for ( j = 2; j <= order; ++j ) {
        for ( i = 1; i < j; ++i ) {
                if ( attached (i,j) ) v |= c;
                c >>= 1;
                if ( !c ) {
                        *yformat++ = v;
                        c = 32; v = 0100;
                }
        }
}
/* If the last byte was partially filled, set it */
if ( c != 32 ) *yformat++ = v;
*yformat = '\0';
```

Listing 4.1.1.1.1. – Graph Adjacency Matrix to y–format Conversion.

4.1.1.2. Graph Adjacency Matrix Format

Although the y–format is the primary vehicle utilized for the representation of a graph's state, other formats are available. One such format is that of the adjacency matrix.

Routines and command line utilities for the input and output of matrices representing the adjacencies of undirected, unweighted, loop-less graphs have been implemented.

All such routines and utilities accept as input, and produce as output ASCII based strings representing the adjacency matrix. Each such ASCII matrix contains $n \times n$ elements, where n represents the order of the graph. Only 0, 1, and the newline are valid characters for the representation the adjacency matrix. No whitespace may separate the characters. A newline is used to separate the rows of the matrix. The order of the graph is inferred from the number of rows and columns comprising the matrix.

The input/output routines and utilities discussed in Section §4.4.3. and Section §4.4.5., respectively, are capable of reading or writing a stream of input containing multiple adjacency matrices. A blank line is used in both the input and output from such utilities to indicate the delineation of matrices.

If an adjacency matrix is provided as input, the first row is designated as the first vertex of the graph, the second row the second vertex, etc.

The input routines also provide sanity checks. For instance, if the number of columns is not equal to the number of rows, or an illegal character (i.e. not a 0

or a 1) is encountered, the input is considered invalid.

The following is the adjacency matrix used to represent the C_{10} graph. This may be used as input to, or produced as output from the various tools provided.

```
0100000001
1010000000
0101000000
0010100000
0001010000
0000101000
0000010100
0000001010
0000000101
1000000010
```

Listing 4.1.1.2.1. – grama Adjacency Matrix Format.

4.1.1.3. Graph Incidence Matrix Format

In addition to the y–format and adjacency matrix formats, an incidence matrix format is available.

Routines and command line utilities for the input and output of matrices representing the incidence between vertices and edges of undirected, unweighted, loopless graphs have been implemented.

All such routines and utilities accept as input, and produce as output ASCII based strings representing the incidence matrix. Each such ASCII matrix contains $n \times e$ elements, where n represents the order of the graph and e represents the size of the graph. Only 0, 1, and the newline are valid characters for the representation of the incidence matrix. No whitespace may separate the characters of the matrix. A newline is used to separate the rows of the matrix. The order of the graph is inferred from the number of rows comprising the matrix while the size is inferred from the number of columns.

The input/output routines and utilities discussed in Section §4.4.3. and Section §4.4.5., respectively, are capable of reading or writing a stream of input containing multiple incidence matrices. A blank line is used in both the input and output from such utilities to indicate the delineation of matrices.

If an adjacency matrix is provided as input the first row is designated as the first vertex of the graph, the second row the second vertex, etc. The first column is

designated as the first edge, and so on.

The routines also provide sanity checks for incidence matrices which are read. For instance if an illegal character (i.e. not a 0 or a 1) is encountered, the input is considered invalid.

The following is the incidence matrix used to represent the C_{10} graph. This may be used as input to or produced as output from the various tools provided.

```
1100000000
1010000000
0011000000
0001100000
0000110000
0000011000
0000001100
0000000110
0000000011
0100000001
```

Listing 4.1.1.3.1. – grama Incidence Matrix Format.

4.1.1.4. Graph Adjacency List Format

The adjacency list format provides a listing of the adjacencies between the vertices of a graph. The non–adjacencies are not presented. Unlike the adjacency and incidence matrix formats, the valid characters found in this format include all numerals, the colon, the space, and the newline character.

Each line of this format is prefixed with a number followed by a colon. The number before the colon indicates the vertex number. The order of the graph is inferred from the largest such prefix value found. The string following the colon is a whitespace separated list of numbers, each representing the vertices to which the current vertex is adjacent. Each line of input is terminated by a newline character.

The input/output routines and utilities discussed in Section §4.4.3. and Section §4.4.5., respectively, are capable of reading or writing a stream of input containing multiple adjacency lists. A blank line is used in both the input and output from such utilities to indicate the delineation of lists.

Because each line is prefixed with a number indicating the vertex, the rows may be arranged in any order. All output from the various tools provided is produced in increasing order. If *holes* are present in the list, the missing vertices will be registered as legal vertices within the graph and will be isolated points (not attached to any other vertices). For instance assume a single line adjacency list was specified and contained the following.

```
10:
```

Listing 4.1.1.4.1. – grama Adjacency List Input Format.

If this was used as input to various routines a graph of order 10 and size 0 (a completely disconnected graph with 10 vertices) would be created.

The routines also provide sanity checks for inputted adjacency lists. For instance, if a vertex specified as a list member is greater than the largest vertex possible the input is considered invalid.

The following is the adjacency list representing the graph K_5. This may be used as input to, or produced as output from the various tools provided.

```
1: 2 3 4 5
2: 3 4 5
3: 4 5
4: 5
5:
```

Listing 4.1.1.4.2. – grama Adjacency List Output Format.

4.1.2. Graph Property Formats

This section describes the various formats which are used to represent properties of graphs and Ramsey graphs.

The properties expressed via these formats are not adequate to determine the exact graph from which each was derived. For instance one format is that of a the clique. All cliques of a given size for a graph may be calculated and output in a standard format. However this information alone is not enough to realize the graph from which the cliques were calculated.

Because of the fact that these formats represent properties of graphs, each is used only as an output format. The software packages implemented provide routines and utilities for generating each of these property formats for a given graph.

4.1.2.1. Clique Format

The clique vector format contains the enumeration of all of the cliques of a specific size for a given graph. This object is generated as output from the various graph manipulation routines. It cannot be used as input.

All numerals, the space, the semi–colon, and the newline character are valid characters. Each of these cliques is separated within the clique vector by the semi–colon character. Each clique is a white space separated list of vertices which are elements of the given clique. The clique vector is terminated with a newline character.

The following is the clique vector of size 2 for the graph C_{10}.

```
1 2; 1 10; 2 3; 3 4; 4 5; 5 6; 6 7; 7 8; 8 9; 9 10;
```

Listing 4.1.2.1.1. – grama Clique Format.

4.1.2.2. Independent Set Format

The independent set vector format contains the enumeration of all of the independent sets of a specific size for a given graph. This object is generated as output from the various graph manipulation routines. It can not be used as input.

All numerals, the space, the semi–colon, and the newline character are valid characters. Each of these independent sets is separated within the independent set vector by the semi–colon character. Each independent set is a white space separated list of vertices which are elements of the given independent set. The independent set vector is terminated with a newline character.

The following is the independent set vector of size 5 for the graph C_{10}.

```
1 3 5 7 9; 2 4 6 8 10
```

Listing 4.1.2.2.1. – grama Independent Set Format.

4.1.2.3. Degree Sequence Format

The degree sequence vector provides a format which lists the vertex degrees for a given graph. This format can be used as input or produced as output from the various graph manipulation routines and utilities. For example one of the graph

realization utilities takes as input a degree sequence for which all valid graphs are to be enumerated.

All numerals, the space, and the newline character are valid. Each degree of the vector is separated from the other degrees by one or more whitespace characters. The degree sequence vector is terminated with a newline character. The degrees within the sequence may be sorted either in increasing or decreasing order. This is a user configurable feature.

The following is the degree sequence vector for the graph C_{10}.

```
2 2 2 2 2 2 2 2 2 2
```

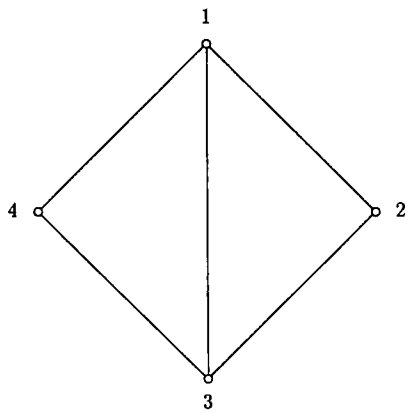
Listing 4.1.2.3.1. – grama Degree Sequence Format.

4.1.2.4. Isomorphic Map Format

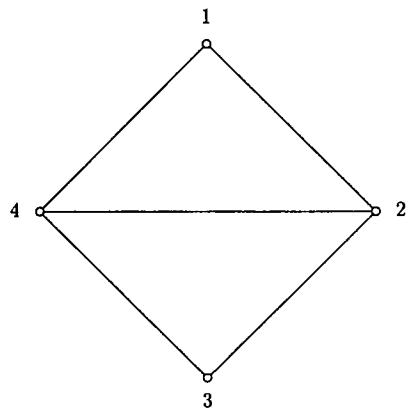
The isomorphic map vector provides a format which lists the mappings of the vertices for two graphs which have been shown to be isomorphic. It is only generated as output from the various graph manipulation routines. It can not be used as input.

All numerals, the space, the comma, the left and right parenthesis, and the newline character are valid. The output contains a set of white space separated vertex pairs. Each pair is contained within a set of left and right parenthesis. Within this pair a comma separates two numbers each of which represent vertex indices. Assuming an isomorphic map exists between two given graphs G_1 and G_2, the first number represents a vertex in G_1 which maps to the second number which is a vertex in G_2.

The following is an example of two graphs followed by one of the possible isomorphic mappings between the two given graphs.



yformat: D}



yformat: Do

```
(1,2) (2,3) (3,4) (4,1)
```

Listing 4.1.2.4.1. – grama Isomorphic Map Format.

4.1.2.5. Distance Matrix

The distance matrix provides a format to convey the minimum distance between vertices of a given graph. It is only generated as output from the various graphs manipulation routines. It cannot be used as input.

All such output tools generate ASCII based strings representing the matrix. Each such ASCII matrix contains $n \times n$ elements, where n represents the order of the graph the matrix represents. All numerals, the space, and the newline character are valid. Each (i, j) element of the matrix is separated from the other matrix elements by one or more whitespace characters. A newline is used to separate the rows of the matrix. The order of the graph is inferred from the number of rows and columns comprising the matrix.

The (i, j) element of the matrix is a number representing the distance from the vertex i to the vertex j in the graph from which the matrix was generated.

The following is the distance matrix for the graph C_{10}. This matrix can be produced as output from the various tools provided.

```
0 1 2 3 4 5 4 3 2 1
1 0 1 2 3 4 5 4 3 2
2 1 0 1 2 3 4 5 4 3
3 2 1 0 1 2 3 4 5 4
4 3 2 1 0 1 2 3 4 5
5 4 3 2 1 0 1 2 3 4
4 5 4 3 2 1 0 1 2 3
3 4 5 4 3 2 1 0 1 2
2 3 4 5 4 3 2 1 0 1
1 2 3 4 5 4 3 2 1 0
```

Listing 4.1.2.5.1. – grama Distance Matrix Format.

4.1.2.6. Column Characteristic Matrix Format

The column characteristic matrix provides a format to convey the number of vertices from which a given vertex is a specific distance. It is only generated as output from the various graph manipulation routines. It can not be used as input.

All such output tools generate ASCII based strings representing the matrix. Each such ASCII matrix contains $n \times n$ elements, where n represents the order of the graph the matrix represents. All numerals, the space, and the newline character are valid. Each (i, j) element of the matrix is separated from the other matrix elements by one or more whitespace characters. A newline is used to separate the rows of the matrix. The order of the graph is inferred from the number of rows and columns comprising the matrix.

The (i, j) element of the matrix is a number representing a count of vertices in the graph from which the vertex i is a distance j.

The following is the column characteristic matrix for the graph C_{10}. This matrix can be produced as output from the various tools provided.

```
2 2 2 2 1 0 0 0 0 0
2 2 2 2 1 0 0 0 0 0
2 2 2 2 1 0 0 0 0 0
2 2 2 2 1 0 0 0 0 0
2 2 2 2 1 0 0 0 0 0
2 2 2 2 1 0 0 0 0 0
2 2 2 2 1 0 0 0 0 0
2 2 2 2 1 0 0 0 0 0
2 2 2 2 1 0 0 0 0 0
2 2 2 2 1 0 0 0 0 0
```

Listing 4.1.2.6.1. – grama Column Characteristic Matrix Format.

4.1.2.7. Row Characteristic Matrix Format

The row characteristic matrix provides a format to convey the number of vertices to which a given vertex is a specific distance. It is only generated as output from the various graph manipulation routines. It can not be used as input.

All such output tools generate ASCII based strings representing the matrix. Each such ASCII matrix contains $n \times n$ elements, where n represents the order of the graph the matrix represents. All numerals, the space, and the newline character are valid. Each (i, j) element of the matrix is separated from the other matrix elements by one or more whitespace characters. A newline is used to separate the rows of the matrix. The order of the graph is inferred from the number of rows and columns comprising the matrix.

The (i, j) element of the matrix is a number representing a count of vertices in the graph to which the vertex i is a distance j.

The following is the row characteristic matrix for the graph C_{10}. This matrix can be produced as output from the various tools provided.

```
2 2 2 2 1 0 0 0 0 0
2 2 2 2 1 0 0 0 0 0
2 2 2 2 1 0 0 0 0 0
2 2 2 2 1 0 0 0 0 0
2 2 2 2 1 0 0 0 0 0
2 2 2 2 1 0 0 0 0 0
2 2 2 2 1 0 0 0 0 0
2 2 2 2 1 0 0 0 0 0
2 2 2 2 1 0 0 0 0 0
2 2 2 2 1 0 0 0 0 0
```

Listing 4.1.2.7.1. – grama Row Characteristic Matrix Format.

4.1.2.8. Characteristic Matrix Format

The characteristic matrix provides a format to convey the number of vertices from which a given vertex is a specific distance composed with the value to which the same vertex is a specific distance. It is only generated as output from the various graph manipulation routines. It cannot be used as input.

All such output tools generate ASCII based strings representing the matrix. Each such ASCII matrix contains $n \times n$ elements, where n represents the order of the graph the matrix represents. All numerals, the space, and the newline character are valid. Each (i, j) element of the matrix is separated from the other matrix elements by one or more whitespace characters. A newline is used to separate the rows of the matrix. The order of the graph is inferred from the number of rows and columns comprising the matrix.

The (i, j) element of the matrix is a number which is composed from the corresponding row and column characteristic matrix values. The following C programming language statement demonstrates how the composition is performed.

```
(cm->matrix)[i][j] = ((rcm.matrix)[i][j]<<8)+(ccm.matrix)[i][j];
```

Listing 4.1.2.8.1. – C Code Characteristic Matrix Composition.

The above listing which demonstrates the creation of a graph characteristic matrix utilizes both the column and row characteristic matrix. It should be noted that both the column and row characteristic matrices are needed when calculating the characteristic matrix for a directed graph. For an undirected graph this is not the case. For an undirected graph the column and row characteristic matrices are equivalent.

The following is the characteristic matrix for the graph C_{10}. This matrix can be produced as output from the various tools provided.

```
514 514 514 514 257   0   0   0   0   0
514 514 514 514 257   0   0   0   0   0
514 514 514 514 257   0   0   0   0   0
514 514 514 514 257   0   0   0   0   0
514 514 514 514 257   0   0   0   0   0
514 514 514 514 257   0   0   0   0   0
514 514 514 514 257   0   0   0   0   0
514 514 514 514 257   0   0   0   0   0
514 514 514 514 257   0   0   0   0   0
514 514 514 514 257   0   0   0   0   0
```

Listing 4.1.2.8.2. – grama Characteristic Matrix Format.

4.1.3. Graph Image Formats

This section describes the formats which are used to represent an image of a graph. These formats do provide the state of a graph in that each expresses the graph's order as well as all of the adjacencies and non–adjacencies between the vertices.

Each of these formats are only used as output formats and not input formats. The software packages implemented provide routines and utilities for generating each of these image formats for a given graph.

This section contains a description of each of the image formats. Section §4.4.5.10. contains examples of the generation of the various image formats using packages implemented for this thesis.

4.1.3.1. DVI Format

DVI (or Device Independent) is a popular image representation format. The various packages implemented for this thesis are capable of producing image files

representing graphs in the DVI language. The technique used is very simple. The groff text formatting program is capable of producing DVI as its output. The DVI generating routines implemented by the author simply create a DVI language version of the graph which is then processed by the groff program to produce a DVI version. As with the DVI format the graph images generated are simple and no attempt is made to produce images which contain minimum edge crossing etc.

4.1.3.2. GIF Format

The GIF (Graphic Interchange Format) is a popular graphical format for representing images. The GIF was created by Compuserve and originally became the de facto standard for representing images on bulletin board systems. Various packages implemented for this thesis are capable of producing images in the GIF format. The *gd* package of C routines created and maintained by Thomas Boutell has been integrated into the packages written by the author. The gd package is used to generate images representing graphs as well as graphs with colored edges, the colored edges being used to highlight cliques or independent sets of a given size. As with the other image formats produced by the packages implemented by the author no attempt is made to produce graph images which contain minimum edge crossings.

4.1.3.3. PIC Format

PIC is a package for formatting images. It was originally written as a pre–processor for the [gt]roff formatting tool. Various routines and programs implemented by the author generate PIC code which represents a given graph. This generated code may then be included in [gt]roff documents and further processed to create a nicely formatted document.

The following is a sample of the first 3 lines of a PIC image file representing a graph.

```
.PS 6.5 9.0
circlerad = 0.025
circle at 0.000000, 1.000000 fill
circle at 0.230616, 0.973045 fill
circle at 0.448799, 0.893633 fill
...
```

Listing 4.1.3.3.1. – PIC Graph Encoding Format.

4.1.3.4. PS Format

PS (or Postscript) is a popular image representation format. The various packages implemented for this thesis are capable of producing image files representing graphs in the Postscript language. The technique used is very simple. The groff text formatting program is capable of producing Postscript as its output. The Postscript generating routines implemented by the author simply create a PIC language version of the graph which is then processed by the groff program to produce a Postscript version.

The following is a sample of the first 3 lines of a Postscript image file representing a graph. The version of Postscript generated as well as the generating program are clearly indicated.

```
%!PS-Adobe-3.0
%%Creator: groff version 1.16.1
%%CreationDate: Wed Dec 27 12:57:47 2000
...
```

Listing 4.1.3.4.1. – PostScript Graph Encoding Format.

4.2. Graph File Formats

This section contains a description of the file formats used to organize the graphs and Ramsey graphs used throughout the experiments performed for this thesis.

4.2.1. G–Format

Although the programs implemented for and used by this thesis can generate graphs in various formats including adjacency and incidence matrices, the de facto standard is the y–format.

By default all graph enumeration programs created for this thesis generate graphs in the y–format. All such graph produced are stored in flat files, the names of which have a standard format. This format is used to convey certain information about the properties of the graphs contained within. The values of the following properties are contained within the names of the graphs: maximum clique free size, maximum independent set free size, order, size, delta (minimum degree).

The following format is used for the graph file names: *[u]g-k.l.n.e.d.y*. Each of the letters *k*, *l*, *n*, *e*, and *d* is substituted with a given value. The *k* indicates the maximum clique free size, the *l* the maximum independent set free size, the *n* the graph order, the *e* the graph size, and the *d* the graph delta.

A file name prefix of *g-* indicates that all *isomorphic* graphs with the given parameters are contained within the file. A prefix of *ug-* indicates that only *non−isomorphic* graphs with the given parameters are contained within the file. The *.y* suffix indicates that the graphs contained therein are all in the y−format. The newline character is used as a record separator for each of the y−format graphs contained within the file.

As an example a file name *g-3.3.5.5.2.y* would indicate that the graphs contained within are 3−clique free, 3−independent free, of order 5, size 5, and that the graphs have a δ (minimum degree) of 2. A file of the name *g-3.3.x.x.x.y* would indicate that the graphs contained within are 3−clique free and 3−independent free but that the order, size, and δ may vary.

4.3. Algorithms

This section contains a description of all of the non−trivial algorithms implemented for the enumeration of graphs and Ramsey graphs. The following algorithms are reviewed.

- ISOMAP Algorithm

  The ISOMAP algorithm is used for the determination of a possible isomorphism and the subsequent mapping of vertices between the two given graphs. This algorithm is dependent upon the knowledge of a graph's distance matrix, characteristic matrix (column and row matrices as well), class vector, and class count vector. These sub−algorithms are also described.

- DELTA Algorithm

  The DELTA algorithm is used for the calculation of all graph feasible degree sequences for $(3, l, n)$ Ramsey graphs. The algorithm is based upon the Ramsey graph edge calculation equations provided in Section §3.1.

- EXTEND Algorithm

  The EXTEND algorithm is used for the enumeration of $(3, l, n)$ Ramsey graphs from already known $(3, l, n-1)$ Ramsey graphs. This algorithm is based on the original description from [RAD88a].

- EXPAND Algorithm

  The EXPAND algorithm is used for the enumeration of $(3, l, n)$ Ramsey graphs from already known $(3, l-1, n-d-1)$ Ramsey graphs. This algorithm is based on the EXPAND algorithm described in [GRA68a], [GRI82a], and [RAD88a].

4.3.1. ISOMAP Algorithm

The ISOMAP algorithm is an implementation of the Schmidt and Druffel isomorphism mapping algorithm. [GOU88a] This algorithm is executed on two graphs. The algorithm outputs a value indicating whether or not the two graphs are isomorphic. If the graphs are isomorphic a mapping between the vertices of the two graphs is also calculated. It is a fast backtracking algorithm which uses vertex classification as an invariant and is applicable to undirected and directed graphs.

Vertex classification is a technique which partitions the vertices of a given graph according to some graph property that is invariant under isomorphism. Assume the vertex set V of a graph can be partitioned into n classes $p_1, p_2, \cdots, p_n$. There are $(p_1)!(p_2)!\cdots(p_n)!$ possible mappings which must be attempted. If the size of p_i is small, the number of cases that must be examined is reduced. The Schmidt and Druffel algorithm partitions the vertices using the relationship of a single vertex to all other vertices. The classification of a vertex v is based on the number of vertices at a given distance from v.

The first step in the vertex classification is calculation of the *distance matrix* for the two graphs on which the algorithm is being run. Figure 4.3.1.1. presents the pseudo–code used for the calculation of the *distance matrix.*

```
/* Calculate the distance matrix */
for ( i = 1 ; i <= order ; ++i ) {
        for ( done = 0, d = 0, D_ij = 0 ; !done ; ++d ) {
                if ( D_ij == d ) {
                        for ( k = 1 ; k <= order ; ++k ) {
                                if ( Adjacent(i,j) && D_ij == -1 ) {
                                        D_ij = d+1; done = 0;
                                }
                        }
                }
        }
}
```

Listing 4.3.1.1. – Distance Matrix Calculation.

The distance matrix for each of the graphs is then used to calculate the column and row characteristic matrices.

The column characteristic matrix and the row characteristic matrix are used to form the characteristic matrix. This matrix denoted X is used for the initial partitioning of the vertices of each graph.

Vertices which have identical rows in the characteristic matrix are assigned to the same *vertex class.* The maximum number of vertex classes for a graph of order n

is n (in this case each vertex is assigned its own class). The vector $C = [k_i]$ is used to denote the vector which maintains the class information. The element k_i denotes the class of the vertex v_i. All vertices with the same class are placed in the same set in the partition of the graph vertices. The vector $CC = [k_i]$ is used to denote the vector which maintains the class count information. The element k_i denotes the count of the vertices in class i. It is necessary but not sufficient that a pair of vertices have the same class for an isomorphic mapping between the vertices to exist.

The following listing provides the pseudo–code used for the calculation of the column characteristic, row characteristic, and characteristic matrices. In this listing the distance matrix for the given graph is denoted by D. The column and row characteristic matrices are denoted by C and R respectively, with the characteristic matrix being denoted by X.

These algorithms assume that the distance matrix for the graph has already been calculated. It should be understood that the column and row characteristic matrices are initialized to *"0"* before the calculation algorithms are executed.

```
/* Calculate the row characteristic matrix */
for ( i = 1; i <= order; ++i ) {
        for ( j = 1; j <= order; ++j ) { if ( k = D_ij ) ++R_i,k; }
}
/* Calculate the column characteristic matrix */
for ( i = 1; i <= order; ++i ) {
        for ( j = 1; j <= order; ++j ) { if ( k = D_ij ) ++C_i,k; }
}
/* Calculate the characteristic matrix */
for ( i = 1; i <= order; ++i ) {
        for ( j = 1; j <= order; ++j ) { X_ij = (R_ij << 8) + C_ij }
}
```

Listing 4.3.1.2. – Characteristic Matrix Calculation.

The characteristic matrix is then used to generate the class vector for the given graph. The characteristic matrix is assumed to already be calculated and is denoted by the variable X. The variable CV is used to denote the class vector.

```
/* Calculate the Class Vector */
for ( i = 2, CV_1 = c = 1; i <= order; ++i ) {
        for ( j = 1; j < i; ++j ) {
                for ( s = k = 1; k <= order; ++k ) {
                        if ( X_ik != X_jk ) { s = 0; break; }
                }
                if ( s ) { break; }
        }
        CV_i = ( s ? CV_j : ++c );
}
```

Listing 4.3.1.3. – Class Vector Calculation.

At this point in the algorithm, the class vector has been calculated and the various characteristic matrices are no longer needed and can be discarded. Note that the distance matrix may not be discarded as it will be needed at later stages in the algorithm. The class count vector can easily be calculated from the class vector. This operation is trivial and a pseudo–code listing is not shown.

After the class count vectors have been computed, an element by element comparison is performed. If the vectors for the two graphs are not equivalent no isomorphic mapping between the graphs exists.

The vectors being equal is a necessary but not a sufficient requirement for the isomorphic mapping between the two graphs. Assuming that the class count vectors are equivalent, the backtracking section of the Schmidt Druffel algorithm is initiated. This portion of the algorithm determines if the graphs are isomorphic and if so provides a vertex to vertex mapping between the two graphs.

The following is a high level pseudo–code description of the algorithm which is used for mapping the vertices from the first graph to the second.

```
/* Each Level Represents a Vertex which is to be Mapped. */
for ( level = 1; level > 0 && level <= order; ) {
        /* Determine the First Unmapped Vertex in G2 with the Same Class. */
        for ( v1 = level, c = CV1[v1]; level <= order; ) {
                /* Determine if the class of v1 is the same as v2 */
                if ( c == CV2[i] ) {
                        /* Check if v2 has already been mapped. */
                        if ( AlreadyMapped(v2) ) { continue; }
                        /* Recalculate the Class Vectors based on the current mapping. */
                        NCV1 = ClassVector ( CV1, D1 ) ; NCV2 = ClassVector ( CV2, D2 ) ;
                        /* Recalculate and Compare the Class Count Vectors. */
                        CCV1 = ClassCountVector ( NCV1 ); CCV2 = ClassCountVector ( NCV2 );
                        /* No Vertex Mapping Found at this Level. */
                        if ( CCV1 != CCV2 ) { continue; }
                        /* Vertex Mapping Found at this Level. */
                        else { Map( v1, v2 ); goto nextLevel; }
                }
        }
        /* No Mapping Found at this Level, Backtrack. */
        previousLevel: --level; continue;
        /* Mapping Found at this Level. */
        nextLevel: ++level; continue;
}
```

Listing 4.3.1.4. – Graph Isomorphism Algorithm.

4.3.2. DELTA Algorithm

The DELTA algorithm implements a function which enumerates all valid graphical degree sequences for given parameters. The degree sequences generated are valid degree sequences for triangle free Ramsey graphs which contain no independent sets of a given size, with a specified graph order as well as a specified graph size.

The DELTA algorithm is denoted *DELTA(l,n,e)*, where l is the size of the independent sets to be avoided in the graph, n is the order of the sequence, and e is the size. This function finds all degree sequences n_i where $i \leq l$ and satisfies the following equations.

Let $f(v) = |H_2(v)| - e(k, l-1, n-deg(v)-1)$, where v is the preferred vertex and $|H_2(v)|$ denotes the size of the graph $H_2(v)$. Given this $f(v) \geq 0$. If $f(v) = 0$ the vertex v is designated *full*.

If G is a $(3,l)-$graph, n_i is defined as the number of $i-$vertices in G. The definitions of n and e follow.

$$n = \sum_{i=0}^{l} n_i \qquad (4.3.2.1)$$

$$2e = \sum_{i=0}^{l} in_i \tag{4.3.2.2}$$

The sum $\sum_{v \in V} f(v)$ can be evaluated resulting in the following.

$$\sum_{v \in V} f(v) = ne - \sum_{i \geq 0}^{l-1} n_i(e(3, l-1, n-i-1) + i^2) \geq 0 \tag{4.3.2.3}$$

The right hand side of the above equation is denoted $\Delta(G, l, n, e)$.

$$\sum_{v \in V} f(v) = \Delta(G, l, n, e) \tag{4.3.2.4}$$

Given this there are at least $n - \Delta(G, l, n, e)$ full vertices in the graph G.

Example 4.3.2.1.

The following graphical degree sequences are generated by the DELTA algorithm for the parameters $l = 7, n = 16$, and $e = 20$. The value of Δ from Equation 4.3.2.4. is also provided.

S: 5 3 3 3 3 3 2 2 2 2 2 2 2 2 2 2, $\Delta = 0$
S: 4 4 3 3 3 3 2 2 2 2 2 2 2 2 2 2, $\Delta = 2$
S: 4 3 3 3 3 3 3 2 2 2 2 2 2 2 2 2, $\Delta = 5$
S: 3 3 3 3 3 3 3 3 3 3 2 2 2 2 1 1, $\Delta = 2$
S: 3 3 3 3 3 3 3 3 2 2 2 2 2 2 2 2, $\Delta = 8$

□

These degree sequences are used by the EXPAND algorithm described in Section §3.1. All of the sequences are graphical and are valid sequences for triangle free Ramsey graphs which contain no independent sets of size 7, have order 16 and size 20.

The degree sequences are determined to be graphical by using the Erdős and Gallai "System of Inequalities". This system is described by the following theorem. [GOU88a]

Theorem 4.3.2.1.

A non–increasing sequence of non–negative integers S: $d_1, d_2, \cdots, d_p$ is graphical if, and only if the following holds:

$$\sum_{i=1}^{p} d_i \text{ is even} \tag{4.3.2.5}$$

and for each integer k *where* $1 \leq k \leq p-1$*:*

$$\sum_{i=1}^{k} d_i \leq k(k-1) + \sum_{i=k+1}^{p} \min\{k, d_i\}. \tag{4.3.2.6}$$

[GOU88a]
□

Example 4.3.2.2.

This example applies the Erdős and Gallai "System of Inequalities" to the sequence S: 5 5 5 5 2 2 2 to prove the non−graphical nature of the sequence.

1. For $k = 1, d_1 = 5 \leq 1(0) + \sum_{i=2}^{7} \min\{1, d_i\} = 6.$
2. For $k = 2, d_1 + d_2 = 10 \leq 2(1) + \sum_{i=3}^{7} \min\{2, d_i\} = 2 + 10 = 12.$
3. For $k = 3, \sum_{i=1}^{3} d_i = 15 \leq 3(2) + \sum_{i=4}^{7} \min\{3, d_i\} = 6 + 9 = 15.$
4. For $k = 4, \sum_{i=1}^{4} d_i = 20 > 4(3) + \sum_{i=5}^{7} \min\{4, d_i\} = 12 + 6 = 18.$

Step 4 shows that the value of $\sum_{i=1}^{k} d_i$ *is greater than* $k(k-1) + \sum_{i=k+1}^{p} \min\{k, d_i\}$*, thus terminating with a negative result the algorithm which verifies that a degree sequence is graphical.* [GOU88a]
□

4.3.3. EXTEND Algorithm

The EXTEND program as originally described in [RAD88a] has been implemented by this author. Note that in the orignal paper the algorithm is not named

EXTEND. This name has been chosen by the author to describe this algorithm.

This algorithm is used for the the enumeration of all $(3, l)$ graphs with $3 \leq l \leq 6$. The algorithm constructs $(3, l)-$graphs from $(3, l-1)-$graphs. The $(3, l-1)-$graphs with $n < l$ must already be enumerated. $(3, l, n)$ graphs with $n < l$ are small and all non−isomorphic graphs may easily be enumerated by straightforward techniques. By using the standard UNIX time command (man time(1)) to measure program running time it is easily shown on a FreeBSD Pentium 400 MHz machine that all non−isomorphic $(3, 6, n)-$graphs with $n < 6$ can be enumerated in less than 0.15 seconds of wall clock time.

The algorithm can be used for the enumeration of Ramsey graph with larger values of l, such as 7. However, the number of $(3, l)$ Ramsey graphs for values of $l > 6$ are extremely large and therefore the use of this algorithm for such parameter situations becomes intractable.

The following is the pseudo−code listing for the algorithm which is defined as the *EXTEND* algorithm.

```
Status extend(g, k) {
    /* Loop over all possible values for the graph order */
    for ( n = k; k < R(3,k); ++n ) {
        /* Loop over all possible values for the graph size. */
        for ( e = e(3,k,n); e ≤ ⌊n(k−1)/2⌋; ++e ) {
            /* Loop over all possible values for minimal degree. */
            for ( d = 0; d ≤ min(k−1, ⌊n(k−1)/2⌋; ++d ) {
                sprintf (s,"g-3.%d.%d.%d.y",k,n−1,e−d,d̄);
                f = fopen (s, "r");
                /* Process all of the (3, k, n−1, d−e, dbar) graphs */
                while ( g = readGraphYFormat ( f ) )
                    /* Calculate all independent sets of size k. */
                    is = IndependentSet ( g, k );
                    /* Add a vertex to the current graph. */
                    v = ++order;
                    /* Attach vertex v to all of the independent sets in s. */
                    graphVertexAttach (v, is );
                }
                fclose (f);
            }
        }
    }
}
```

Listing 4.3.3.1. − EXTEND Algorithm.

4.3.4. EXPAND Algorithm

The exhaustive construction of all $(3,7)$, $(3,8)$, and $(3,9)$ graphs is not feasible. It is possible to restrict the construction to some $(3,7)$, $(3,8)$, and $(3,9)$ graphs. The EXPAND algorithm, as originally described in [GRA68a] and [GRI82a], is used for this process.

The EXPAND algorithm constructs all $(3,l,n,e)-$graphs from smaller existing Ramsey graphs. Specifically $(3,l-1,n-d-1,e-Z(v))-$graphs, with v designating the preferred vertex, are used as *building blocks*. In the following discussion the label G is used to denote the $(3,l,n,e)-$graph and the label $H_2(v)$ will be used to denote the $(3,l-1,n-d-1,e-Z(v))-$graph, where d is the degree of the vertex v and $Z(v)$ is the sum of the neighbors of vertex v.

The following is a list of the tasks which must be performed to execute the EXPAND algorithm for a given parameter situation.

- Ramsey Graph Degree Sequence Calculation
- $v + H_1(v)$ Graph Calculation
- $H_2(v)$ Graph Independent Set Calculation
- M−Matrix Calculation
- Good Independent Set Calculation
- $H_1(v)$ and $H_2(v)$ Graph Attachment
- Clique and Independent Set Checks

The following sections provide a detailed description of each of these tasks.

4.3.4.1. Ramsey Graph Degree Sequence Calculation

The first step in the execution of the EXPAND algorithm is the calculation of valid degree sequences for the Ramsey graph parameter situation under investigation.

This is performed by using the generic degree sequence calculation routine provided as part of the grama package. This routine can enumerate all valid degree sequences for a given graph order and if specified, graph size. Various graph criteria, such as maximum vertex degree, can also be constrained.

The degree sequence enumeration routine uses a simple recursive backtracking algorithm. For the EXPAND algorithm, all degree sequences are calculated including sequences for disconnected graphs. All degree sequences are checked for validity using the Erdős and Gallai "Method of Inequalities". See Theorem 4.3.2.1. and Example 4.3.2.2. for a complete description of this technique.

After all of the generalized degree sequences have been calculated, each is tested for properties which are required for the it to be a valid Ramsey graph degree

sequence. Specifically the calculation of the Ramsey graph parameter situation Δ value as described in Section §4.3.2. is executed. All graphs which have a Δ with $\Delta \geq 0$ may be valid Ramsey graph degree sequences. Knowledge of the value Δ combined with the enumerated degree sequences is used for determining which $(3, l-1, n-d-1)-$graphs are to be utilized in forming the set of $H_2(v)$ sub−graphs for the EXPAND algorithm.

Upon calculation of all valid degree sequences a small set of parameters, derived from the degree sequences is established. These values are utilized by the EXPAND algorithm. The various vertex degree values are utilized as possible values for the degree of the attached vertex v.

4.3.4.2. $\mathbf{v + H_1(v)}$ Graph Calculation

As previously established, any $(3, l)-$graph G can be partitioned into 2 smaller Ramsey graphs. These two sub−graphs are referred to as the $H_1(v)$ sub−graph and the $H_2(v)$ sub−graph. $H_1(v)$ is a $(2, l)-$graph and $H_2(v)$ is a $(3, l-1)-$graph. Note that both these sub−graphs are already calculated Ramsey graphs. The figure in Example 3.1.3. presents a pictorial delineation of the $H_1(v)$ and $H_2(v)$ sub−graphs for the unique $(3, 5, 10, 12)-$graph.

Given the parameter situation under investigation, along with the order of the $H_2(v)$ sub−graph, the order of the $H_1(v)$ sub−graph is easily determined. Since the $H_1(v)$ sub−graph is a $(2, l)$ Ramsey graph, it contains no cliques of size 2 and therefore must have a size of 0.

As per the description of the EXPAND algorithm in Section §3.1. the preferred vertex, v, is attached to all of the vertices which comprise the $H_1(v)$ sub−graph.

Given this data, the size of the $v + H_1(v)$ sub−graph as well as the size of the $H_2(v)$ sub−graph are both known. The sum of the sizes of these two sub−graphs is less than the size of the desired Ramsey graph situation under investigation. Utilizing this fact, the number of edges required to attached the $v + H_1(v)$ sub−graph to the $H_2(v)$ sub−graph is easily calculated.

The combinations which are available to attach the vertices of the $H_1(v)$ sub−graph to the vertices of the $H_2(v)$ sub−graph need to be established.

4.3.4.3. Independent Set Calculation

Given knowledge of the $H_1(v)$ and $H_2(v)$ sub−graphs, the next phase of the EXPAND algorithm is the connection of these graphs.

From the definition of the vertex v and the $H_1(v)$ and $H_2(v)$ sub−graphs, v is not connected to any vertices in the $H_2(v)$ sub−graph. All of the connections to be made must be from vertices in the $H_1(v)$ sub−graph to vertices in the $H_2(v)$ sub−graph. The vertices in $H_1(v)$ must also only be connected to independent

sets in $H_2(v)$. If the vertices of the $H_1(v)$ sub−graph were connected to vertices in the $H_2(v)$ sub−graph which were not members of an independent set, a triangle would be formed. Since the desired Ramsey graphs are all $(3, l)$−graphs this is impossible.

An efficient method of calculating independent sets in the $H_2(v)$ sub−graphs is needed. The pseudo−code in Listing 4.3.4.3.1. presents a simple recursive backtracking algorithm which calculates independent sets for a given graph. This routine is provided as part of the grama package.

```
Status independentSet(ris, is, g, s, l, p, pattern, ignore) {
        n = 0;
        t = g->order - --s;
        if (p) i = l[p-1] + 1; else i = 0;
        for (b = (1 << i); i < t; ++i, b <<= 1) {
                if (b & ignore) continue;
                for (j = 0; j < p; ++j) {
                        if (g->am.matrix[l[j]+1][i+1]) break;
                }
                if (j < p) continue;
                l[p] = i; lp = pattern | b;
                if (!s) {
                        ++n; *is++ = lp;
                }
                else {
                        lis = is + n;
                        independentSet(ris,lis,g,s,l,p+1,lp,ignore);
                }
        }
}
```

Listing 4.3.4.3.1. − Independent Set Calculation.

4.3.4.4. Good Independent Set Calculation

Because the Ramsey graphs to be enumerated must be triangle free, the vertices in the $H_1(v)$ sub−graph must be attached to vertices which are members of various independent sets in the $H_2(v)$ sub−graph. The maximum size of the independent sets in $H_2(v)$ will be $l - 2$ since $H_2(v)$ is a $(3, l - 1)$ graph. [GRI82a]

In the previous section a method for enumerating all possible independent sets for a graph was detailed. A calculation for the order of the $H_1(v)$ sub−graph was also presented. Given this data and knowledge of the number of edges required to connect the $H_1(v)$ and $H_2(v)$ sub−graphs, all possible degree sequences for the $H_1(v)$ sub−graph can be calculated.

As presented in Definition 3.1.7. the Z−sum value, $Z(v)$, is used to designate the sum of the degrees of the vertices for the $H_1(v)$ sub−graph. Given the order of the $H_1(v)$ sub−graph and the value $Z(v)$, the *sequenceGenerate()* routine provided as part of the grama package, can be utilized to enumerate all possible $H_1(v)$ degree sequences. Such degree sequences are designated as the $H_1(v) - H_2(v)$ degree sequence.

Knowledge of the possible degree sequences for the $H_1(v)$ sub−graph dictates the size of the independent sets in the $H_2(v)$ sub−graph which must be known. Not all of the independent sets found in the $H_2(v)$ sub−graph are valid sets for the purpose of connecting the $H_1(v)$ and $H_2(v)$ sub−graphs. The independent sets which are valid are referred to as *good independent sets.* [GRI82a]

The first step in the calculation of the good independent sets is the determination of *where* within a graph G the l−independent sets could possibly be found. If T is a l−independent set in G, and T contains the vertex v, then T contains none of the vertices in the $H_1(v)$ sub−graph. Therefore $T - \{v\}$ is a $(l-1)$−independent set in the $H_2(v)$ sub−graph which is impossible given that $H_2(v)$ is a $(3, l-1)$−graph. If T contains less than 2 vertices in the $H_1(v)$ sub−graph the same scenario occurs. Therefore T must contain at least 2 vertices in $H_1(v)$. [GRI82a]

Assume that T is a l−independent set in G. Assume that T contains exactly l vertices, $w_{i_1}, w_{i_2}, \cdots, w_{i_m}$, in $H_1(v)$ with $m \geq 2$. Let $V_2 = V(H_2(v)))$, then $T - \{w_{i_1}, \cdots, w_{i_m}\}$ is a l−independent set in $V_2 - (S_{i_1} \bigcup S_{i_2} \cdots S_{i_m})$. If when the m sets $S_{i_1}, \cdots S_{i_m}$ are removed from V_2, an $(l-m)$−independent set denoted S remains, then $S \bigcup \{w_{i_1}, \cdots, w_{i_m}\}$ is an l−independent set in G.

Thus, a necessary and sufficient condition for the selection of the sets $\{S_1, S_2, \cdots, S_{k-2}\}$ from a $(3, l)$ graph is that for each sub-selection $\{S_{i_1}, \cdots S_{i_m}\}$, with $m \geq 2$, the set $V_2 - (S_{i_1} \bigcup \cdots \bigcup S_{i_m})$ contains no $(l-m)$−independent sets. [GRI82a]

Two sets S_i and S_j form a *good pair* if $V_2 - (S_i \bigcup S_j)$ has no $(l-2)$−independent sets. It is necessary, but not sufficient, that each pair S_i and S_j be a *good pair.* The property of being a good pair is dependent upon the $H_2(v)$ sub−graph. [GRI82a]

Given that a method for determining good pairs has been established a means of efficiently calculating such pairs is required.

4.3.4.5. M−Matrix Calculation

Knowledge of good independent sets facilitates the attachment of the $H_1(v)$ sub−graph and the $H_2(v)$ sub−graph during execution of the EXPAND algorithm.

M−matrices are utilized for the calculation of pairwise good independent sets. The calculation and use of these matrices increases the efficiency of the EXPAND algorithm. The number of possible independent sets utilized for the $H_1(v)$ sub−graph to $H_2(v)$ sub−graph attachment is reduced.

Definition 4.3.4.5.1.

The M−Matrix for an $H_2(v)$ sub−graph is defined as the $m \times n$ matrix $M_{ij} = [m_{xy}]$, where m is the number of independent sets of size i and n is the number of independent sets of size j. If $x < y$ the value m_{xy} is defined to be 1 if the independent set S_i^x and S_j^y form a good pair, and 0 otherwise. [GRI82a]
□

Given an $H_1(v)$ $H_2(v)$ sub−graph attachment degree sequence of $\{p, q, r\}$, the M−Matrices M_{pq}, M_{pr}, and M_{qr} must all be calculated. The size of each of these matrices is dependent upon the number of independent sets within the $H_2(v)$ sub−graph. Therefore determining the sizes of the M−Matrices necessitates the calculation of the needed $H_2(v)$ independent sets.

As an example, an M−Matrix calculation utilized by the EXPAND algorithm for the attachment of the (3,3,5,5)−graph to the (3,4,8,10)−graph is provided in Example 4.3.4.5.1.

4.3.4.6. $H_1(v)$ and $H_2(v)$ Graph Attachment

As previously stated, enumeration of $(3, l, n, e)$−graphs requires knowledge of $(3, l-1, n-d-1)$−graphs. Given this along with the calculation of valid Ramsey graph degree sequences, good independent set calculation, and M−Matrix calculation the calculations required to enumerate Ramsey graphs are trivial.

Given the $H_1(v)$ sub−graph degree sequence calculation described in Section §4.3.4.1., along with the $H_2(v)$ sub−graph M−Matrix calculation presented in Section §4.3.4.5., a simple backtracking algorithm is utilized to connect the vertices of the $H_1(v)$ sub−graph to the vertices of the independent sets of the $H_2(v)$ sub−graph.

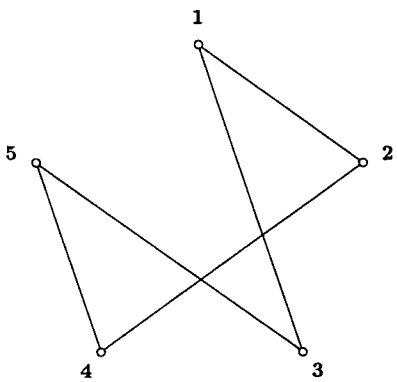
Although the connection techniques described are guaranteed to calculate Ramsey graphs which are triangle free, the graphs may not be free of independent sets for the parameter situation under investigation.

4.3.4.7. Resulting Ramsey Graph Checks

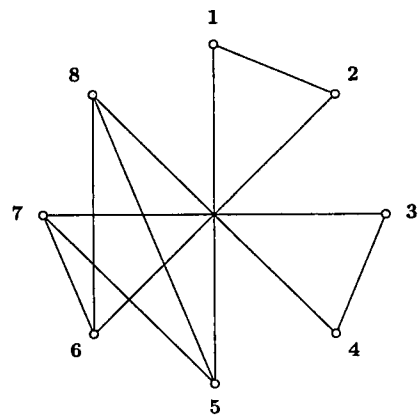
Upon successful attachment of the $H_1(v)$ and $H_2(v)$ sub−graphs, the resulting graph needs to be checked for independent sets of the given size. The attachment process, albeit to good independent sets, does not guarantee that the resulting

Example 4.3.4.5.1.

In the following the leftmost displayed figure is of the unique $(3,3,5,5)-$*graph. When the EXPAND algorithm is executed for the parameter situation* $l = 4$, $n = 8$, *and* $e = 10$ *the rightmost displayed figure, the unique* $(3,4,8,10)-$*graph is produced.*



yformat: ErL



yformat: HaaAXx

n	1	2	3	4	5
1	0	0	1	0	0
2	1	0	0	0	0
3	0	1	0	0	0
4	0	0	0	1	0
5	0	0	0	0	1

Figure 4.3.4.5.1. – M_{12} **for the unique** $(3,3,5,5)-$**graph.**

□

graph is valid for the given Ramsey graph parameter situation sought.

Note that cliques of size 3 need not be checked for. This is given from the fact that both the $H_1(v)$ and $H_2(v)$ sub−graphs are already valid Ramsey graphs. As long as the vertices of the $H_1(v)$ sub−graph are only attached to independent sets in the $H_2(v)$ sub−graph, no triangles will be formed.

After each successful $H_1(v)$ to $H_2(v)$ sub−graph attachment, the resulting graph is checked for independent sets of size l using the grama package *independentSet-FastIndependentSet()* routine.

This process does not eliminate graph isomorphisms in the Ramsey graphs calculated. The removal of such graphs must be performed independently. The grama package routine simply checks the resulting graphs for independent sets of the given size. The elimination of the isomorphisms is performed by the ytool package program *ylabel*. This tool is an adaptation of Brendan McKay's *labely* program.

The elimination of the isomorphic graphs could also be performed by using the author's implementation of the ISOMAP algorithm. The *labely* program was used instead because of its ability to handle very large sets of input graphs efficiently.

4.4. Custom Software Packages

This section contains a overview of all of the software packages which were implemented by the author. The software was designed and implemented to be applicable to generic graph theoretical problems as well as having functionality which is directly applicable to the task of enumerating Ramsey graphs.

All of the packages were designed to be used on various UNIX environments. A generic configuration and build/install process was implemented to assist in the use of the software on various flavors of UNIX. All software packages have been tested on the following platforms: FreeBSD 2.x, 3.x, and 4.x, NetBSD 1.5.x, OpenBSD 2.x, IRIX 5.x and 6.x, Solaris 2.x, and Tru64 UNIX.

4.4.1. Package Configuration and Build Process

The software packages implemented to obtain the goals of this thesis were designed to be used on UNIX environments. However there are a plethora of such environments. To achieve maximum usability of the software on these platforms the GNU autoconf suite was used.

Autoconf provides its user with the ability to create a generic Bourne shell (man sh(1)) script which is executed as the first stage in the package build/installation process. This script examines the system upon which the package is being built and configures the various Makefiles to create a build environment for the package which is appropriate for the current platform.

The use of autoconf for the configuration, build, and installation of the various packages has been integrated into all of the software packages implemented for this thesis.

The following statement demonstrates the execution of the configure script and the first few lines which are output.

```
% ./configure

creating cache ./config.cache
checking architecture... i386-freebsd
checking for ar... ar
checking for env... env
checking for groff... groff
...
```

Listing 4.4.1.1. – grama Package Build Procedure .

The configure script generates Makefiles (see make(1)) from autoconf Makefile templates. These Makefiles are customized to build the grama package for the current UNIX environment being utilized. After successful generation of the Makefiles the make command may be invoked to build all of the sub–packages which comprise the grama package.

```
% make

Making all in grama
Compiling clique.o
Compiling degreeSequence.o
Compiling displayOption.o
Compiling global.o
...
```

Listing 4.4.1.2. – grama Package Compilation Process.

At this point the software may be installed. By default all packages are configured and built to be installed in the build user's home directory.

```
% make install

Making install in grama
Installing clique.h in /home/robg/include
Installing degreeSequence.h in /home/robg/include
Installing displayOption.h in /home/robg/include
Installing graph.h in /home/robg/include
...
```

Listing 4.4.1.3. – grama Package Installation Process.

The configure script is extremely flexible and allows for a great deal of customization of the gram package. Basic configuration options are provided by the autoconf package by default. For example, the script provides options for the specification of an install prefix.

The following invocation of the configure script will configure the package to be installed with a base directory prefix of */usr/local.*

```
% ./configure --prefix=/usr/local
```

In addition to the standard options, the ability to designate specific grama package configuration options is also available. Such options include grama routine parameter checking, grama storage allocation failures, as well as grama debugging message output.

The following invocation of the configure script provides a complete list of all of the command line parameters which are available.

```
%./configure --help
```

The GNU autoconf package which is used to create the configure script can be found at *http://www.gnu.org/software/autoconf/autoconf.html.* Documentation on its use can be found in [MAC96a].

4.4.2. error Package

This section contains a description of the *error* package. The error package is a library of routines which can be either dynamically or statically linked to a program. It is designed to be used in UNIX environments. The package implements the error routines described and implemented by W. Richard Stevens in [STE92a]. The library provides a total of five routines for handling both fatal and non−fatal

program errors.

The following is a summary of the functionality provided by each of the three fatal error routines.

- Output to *stderr* a user provided error message; produce a core image; terminate with a status of 1
- Output to *stderr* a user provided error message; terminate with a status of 1
- Output to *stderr* a user provided error message; return to the calling routine

4.4.3. grama Package

This section contains a description of the *grama* package. The grama package is a library of routines which can be either dynamically or statically linked to a program. It is designed to be used in UNIX environments.

The package implements a group of graph theoretical related objects as well as routines (or methods) to manipulate them. Such objects include, but are not limited to, adjacency, incidence, distance and characteristic matrices, graphs, cliques, independent sets, and sequences.

The macro MAX_ORDER is defined in the header file *grama.h*. This macro by default is defined to have a value of 32.

```
#define MAX_ORDER 32
```

The value of the MAX_ORDER macro determines the maximum order that a graph object may have. This value is critical for all of the objects provided by the grama package. If the MAX_ORDER is defined to be 32, the adjacency matrix object for a graph will be allocated as a 32×32 integer matrix and the degree sequence object will be allocated as a 32 integer vector.

The package may be *built* with various compile time options which specify the level of checking to be performed for object access and manipulation. The following is a list of the features with which the package may be built. These command line flags are passed to the configure script at package build time.

- ––enable–debug

  This feature enables all debug message output. All of the routines which comprise the grama package contain debug messages. By default these messages are not compiled into the package. If this option is enabled the code to output the debug messages is compiled into the package. Note that this does not cause the debug messages to be automatically displayed. Each module of the grama package contains a local variable which must be set to a non–zero value in order for the messages to be displayed. Such a variable may be set at compile time or set using a command line flag to the respective ytool program.

- ––enable–check

  This feature enables all of the checking features. These features include alloccheck, boundcheck, overflowcheck, and paramcheck.

- ––enable–alloccheck

  This feature enables checking of the return value from calls to the *malloc* routine and it various cousins (i.e. calloc, etc.) With this option enabled the return value is checked to verify that the memory was successfully allocated. If the memory allocation failed (i.e. malloc returned a null pointer) the C Library *abort()* routine is invoked to alert the user at which point in the source code the failure occurred. A *core* dump memory image is produced.

- ––enable–boundcheck

  This feature enables checking of the bounds of various package objects. For instance if the *graphGetAdjacent(i,j)* routine is invoked with the i and j variables representing indices of vertices of a graph the values will be checked for the correct range, specifically $1 \leq i, j \leq$ MAX_ORDER.

- ––enable–paramcheck

  This feature enables checking of parameters passed to routines. For instance if the *graphOrder(g)* routine is invoked with the g variable representing a pointer to a grama graph object, the pointer value will be checked to verify that it is not a null pointer.

The grama package is designed with an interface to *SWIG* which allows all of the objects and routines implemented within the package to be used by various UNIX scripting languages including *Guile*, *Perl*, *Python*, and *Tcl/TK*. The SWIG interface to the grama package has only been designed to work with the Perl and Python scripting languages. This does not indicate that Guile and Tcl/Tk could

not be utilized, just that the interface for these languages has not be implemented.

4.4.4. tool Package

The tool package is a library of routines which can be either dynamically or statically linked to a program. It is designed to be used in UNIX environments.

It provides a set of general routines used throughout the implementation of the programs which comprise the ytool Package.

The following is a summary of the functionality provided by the tool package:

- Recursive directory copying and removal
- String manipulation including integer/string conversion, string reversal, string reversal in−place
- Integer comparison functions

4.4.5. ytool Package

This section contains a description of the software tools implemented by the author for the enumeration and manipulation of Ramsey graphs. All of the tools were designed and implemented to be used in UNIX environments.

The programs were designed with the tool philosophy, which deems that programs should be designed to perform a single or a few tasks well. Each of the programs in the y−tool suite has a specific purpose. All of the tools can read input from *standard input* or from command line specified files. All of the tools also can write output to either *standard output* or a command line specified file.

The tools share a set of common flags. The following list describes each of these flags.

- -D

  The -D flag enables debugging mode. If the utility has been built (compiled) with the conditional debugging statements enabled this feature will turn on the given statements. Output regarding program flow will be written to the stderr file handle.

- -h

  The -h flag outputs a list and description of the command line flags available to the utility.

- -s

  The -s flag outputs a brief synopsis of the program containing its function, as well as the input the program expects and the output which it produces.

- -v

  The -v flag outputs the executable version for the given program.

For each of the command line programs described an example is provided demonstrating its use. For each example the command to be invoked as well as the output produced are provided. All commands invocations as well as output produced are presented in `computer modern` font.

Figure 4.4.5.1. presents and diagram of the architectural overview of the *ytool* package. The objects contained within circles represent the command line programs which are provided by the package. The objects contained within rectangles are the libraries provided by the error, grama, and tool packages. The figure demonstrates the dependencies of the command line programs and libraries. The following sub−section provides a high level description of each of the tools.

ycount Program

The *ycount* program generates lists or tables of counts of the number of Ramsey graphs with various properties.

If files are specified as command line arguments the ycount program will process all of the graphs contained within those files. If no command line file arguments are specified the ycount program will process graphs read from standard input.

A command line flag of *-n* will instruct the ycount program to generate a list of the number of graphs categorized by graph order. A command line flag of *-e* will cause the program to generate a list of the number of graphs categorized by graph size.

A command line flag of *-t* will instruct the program to create a table of the number of graphs in which the graph orders are listed on the topmost x−axis and the graph sizes are listed on the leftmost y−axis. The counts of the graphs with the given order and size are contained within the table in the appropriate column and row.

Chapter 6 contains examples of such tables. The counts of all $(3,3)$, $(3,4)$, $(3,5)$, and $(3,6)$ Ramsey graphs are presented.

Example 4.4.5.1.

Assuming that the following files are present in a given directory: g-x.x.1.x.x.y, g-x.x.2.x.x.y, g-x.x.3.x.x.y, and g-x.x.4.x.x.y. These four

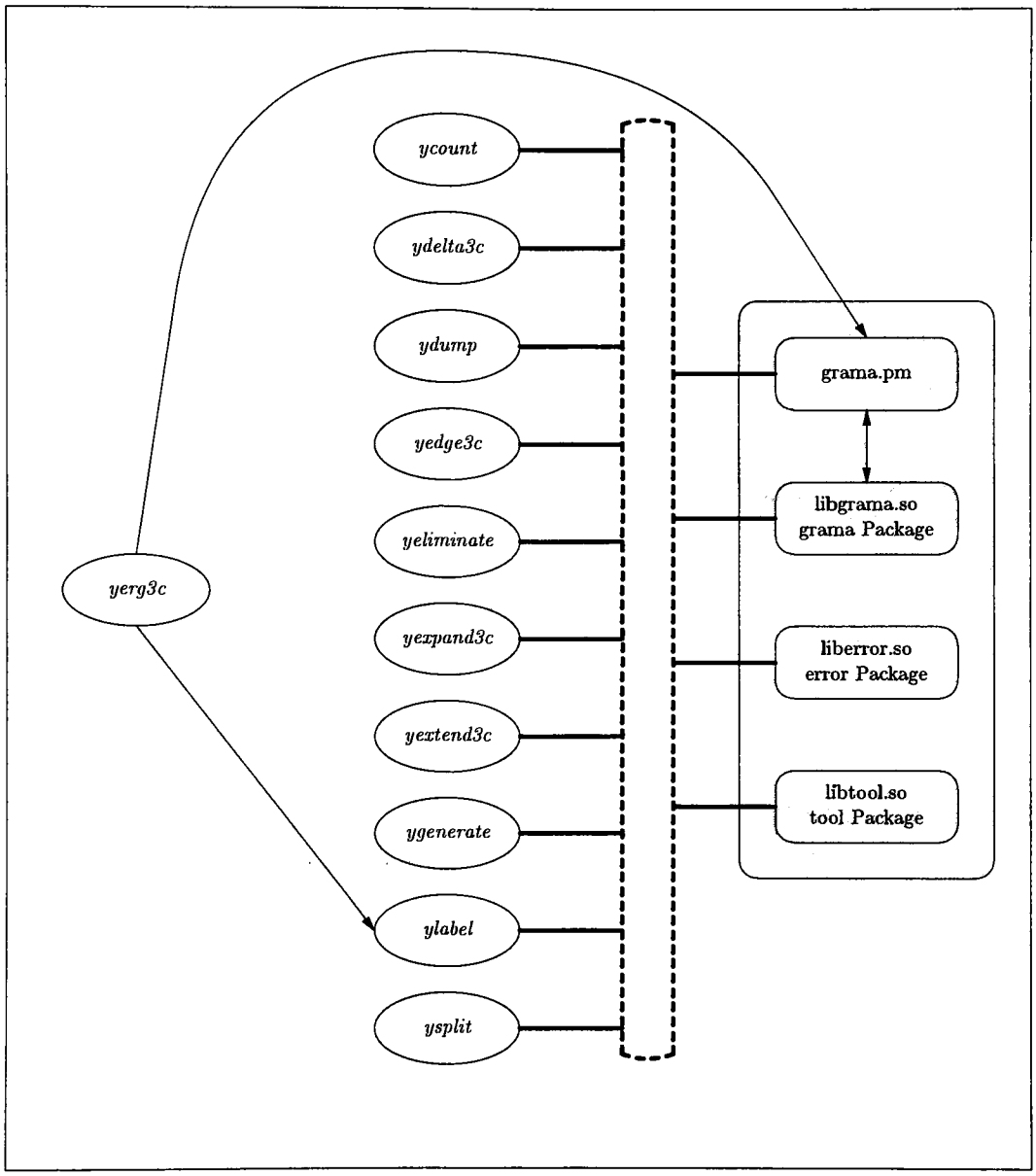


Figure 4.4.5.1. – ytool/grama Package Architctural Overview.

files contain all non–isomorphic graphs of order 1, 2, 3, and 4 respectfully. In this case the clique and independent sets sizes are irrelevant and therefore not present in the graph file name encoding.

The following command statement displays a table which contains the counts of the graphs organized by order and size for the four files described above.

```
% ycount g-x.x.?.x.x.y
```

The output contains an aligned table of the counts of the graphs based upon the graph's order and the graph's size. The bottom row contains a summary of the counts of the total number of graphs with the given order. The rightmost column contains a summary of the count of the total number of graphs with the given size.

	1	2	3	4	
0	1	1	1	1	4
1		1	1	1	3
2			1	2	3
3			1	3	4
4				2	2
5				1	1
6				1	1
	1	2	4	11	18

□

ydelta3c Program

The *ydelta3c* program is a degree sequence generator. It generates valid $(3, l)$ Ramsey graph degree sequences. The program takes no input.

Command line parameters must be provided specifying the parameters for the degree sequences which are to be generated. These parameters include the Ramsey graph order, size, and independent set free size. All valid $(3, l)$ Ramsey graph degree sequences are output. The sequence is specified in descending order and is terminated by a comma. Following the command is the Δ value for the given degree sequence. See Section §3.1. for an explanation of this value. Multiple degree sequences for the specified parameter situation may be output, each record being separated by a newline character.

Example 4.4.5.2.

The following command statement demonstrates the invocation of the ydelta3c program for the generation of valid Ramsey graph degree sequences containing no independent sets of size 7, of order 22, and size 60.

```
% ydelta3c -l 7 -n 22 -e 60

6 6 6 6 6 6 6 6 6 6 5 5 5 5 5 5 5 5 5 5 5 5, d = 38
```

□

ydump Program

The *ydump* program is a format converter. Specifically it converts from and to various graph formats. It can be used to convert from a graph's y−format to various matrix formats. These format include the adjacency matrix, incidence matrix, distance matrix, column characteristic matrix, row characteristic matrix, and characteristic matrix. It can also be used to convert from a graph's y−format to a graphs adjacency list as well as a graphs degree sequence. Various image representations of a y−format graph can also be created including DVI, PIC, and Postscript formats.

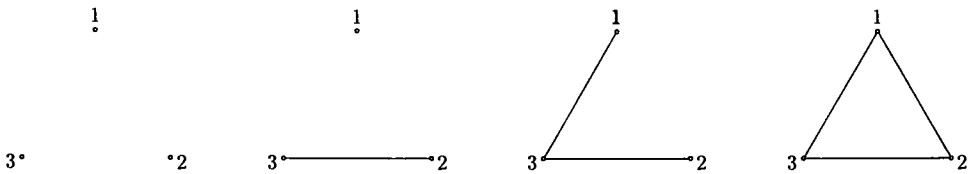
Example 4.4.5.3.

The following command statement demonstrates a method which can be used to generate all non−isomorphic graphs of order 3 from the degree sequence S: 2 2 2.

```
ygenerate -d 3 | ydump -r | ylabel | sort -u \
     | ydump -i pic
```

The first segment of the command pipeline invokes the ygenerate program to enumerate all undirected graphical degree sequence of order 3. The second segment performs the realization of the degree sequences and outputs all of the labeled y−format graphs. These labeled y−format graphs are then input to the ylabel program which performs the graph canonicalization, the output being uniquely sorted to eliminate duplicates.

A total of four non–isomorphic undirected graphs of order 3 are produced. These resulting graphs are input to the last segment of the pipeline. At this point the ydump program generates commands and text formatted for the PIC groff pre–processor. The four following y–format graphs are produced:



C@ CH CX Cx

□

yedge3c Program

The *yedge3c* program calculates the minimum size for a given parameter situation. If the exact lower bound $e(3, l, n)$ cannot be determined, a range of possible values for $e(3, l, n)$ is output.

Example 4.4.5.4.

The following command executes the yedge3c program for the parameters with the independent set free size being 7 *and the graph order being* 16.

```
% yedge3c -l 7 -n 16
```

The following output contains the minimum size of the Ramsey graph for the given parameter situation.

```
l = 7, n = 16, e = 20
```

□

yerg3c Program

The *yerg3c* program is a Perl script which automates an entire run of the yextend3c program. By construction, the yextend3c program attempts to enumerate $(3, l, n, e)$ Ramsey graphs. Only $(3, l)$ Ramsey graphs of order n and size e are enumerated. The yerg3c utility automates the process to enumerate all $(3, l)$ Ramsey graphs over the valid range of n and e for the given situation.

The yerg3c utility requires at least the "-l" flag. This specifies the independent set free size. The yerg3c utility creates a yextend3c process for each valid n and e parameter combination in order to enumerate all $(3, l)$ Ramsey graphs.

The yerg3c utility may be invoked in one of two run modes. These two modes are *local* and *distributed* mode. The local mode executes all of the yextend3c processes on the same workstation upon which the original yerg3c utility was executed. The distributed mode creates an autoson batch job entry for each possible valid $(3, l, n, e)$ scenario. Each of these batch jobs is submitted to the autoson queue.

In the distributed run mode it is assumed that the required autoson processes (specifically aurun0) are running on the proper set of networked hosts. Each host must have access to the NFS share which contains the autoson queue as well as the base graph and destination graph directories. See Section §4.5.1. for a description of autoson, and Section §4.4.5. for a description of the setup requirements needed for yextend3c and autoson.

Example 4.4.5.5.

The following command line executes the yerg3c instructing it which enumerates all (3,6)−graphs. These graphs have a minimum order of 1 and a maximum order of 17. There are 761692 such graphs. Since the "-d" flag was not specified the yextend3c jobs will be executed locally and autoson will not be utilized.

```
% yerg3c -l 6
```

The following output displays the first few lines of the execution run to enumerate all (3,6)−graphs. Note that the "-w" flag specifies the name of the directory which is to be used to store the files containing the graphs enumerated before isomorphic graphs are removed.

```
starting local run-mode for graphs of order: "6"

        yextend3c -e 1 -k 6 -n 6 -w ./.yextend3c6k
        yextend3c -e 2 -k 6 -n 6 -w ./.yextend3c6k
        yextend3c -e 3 -k 6 -n 6 -w ./.yextend3c6k
        yextend3c -e 4 -k 6 -n 6 -w ./.yextend3c6k
        .
        .
        .
```

□

yexpand3c Program

The *yexpand3c* program implements the EXPAND algorithm as described in Section §4.3.4. The utility enumerates all $(3, l, n, e)-$graph from existing $(3, l-1, n-d-1, e-Z(v)-$graphs for the given parameter situation. The utility examines the current directory for files containing base graphs These files must have a G−File name format as specified in Section §4.2.1.

The utility accepts command line parameters which specify the parameter situation for which the EXPAND algorithm is to be invoked. The independent set size (specified by the -l flag), order (specified by the -n flag), and size (given by the -e flag) of the Ramsey graphs sought must be provided.

Because of the long execution time for the EXPAND algorithm it is possible that the process executing the yexpand3c program may be killed given machine reboots, etc. To avoid the loss of computations and the need to re−start the specific parameter situation execution, a feature to allow for the recording of the algorithm's status has been implemented. This feature is implemented with the use of a mark file. The mark file is created in the directory in which the process is executing and has a file name of the form *.g-3.%l.%n.%e.m.* In this format, *%l* indicates the independent set free size, *%n* the order, and *%e* the size for the current parameter situation. By default the yexpand3c will start a new enumeration each time it is executed. If the "-t" flag is specified, the utility will examine the current working directory for a mark file and if found use the data within to *prime* the algorithm to the last known state.

Various command line flags are available to alter the enumeration process. The "-d" flag can be specified to designate the degree of the preferred vertex (i. e. the vertex v of the graph $v + H_1(v)$). If this flag is not specified the entire range of valid degrees for the preferred vertex is attempted. The "-g" flag can be specified to designate a particular graph file which is to be used to obtain the base $(3, l-1, n-d-1, e-Z(v))-$graphs for the algorithm's execution. The "-g" flag can be extremely useful in parameter situations which are large. In such an instance the various base graphs can be configured such that each input file contains a small number of graphs. A yexpand3c process can then be executed for each of the graphs and the parameter situation required. If the autoson distributed computing utility is used the various jobs can be spread over a network of UNIX® workstations. The granularity of the base files (i.e. the number of graphs contained within each) can be controlled. See Section §4.5.1. for a description of autoson and Section §4.4.5. for an example of the setup process used.

A *simulated* execution mode flag has been implemented. This allows for a run to execute without actually attempting to enumerate any Ramsey graphs. The run simply outputs what would have occurred if the run was executed for real. A report is generated as to the degrees for the preferred vertex v which would be

attempted as well as a list of the Ramsey graph files which would be processed.

It should be noted that the yexpand3c utility produces Ramsey graphs which may be isomorphic. The utility does not attempt to remove such graphs. There are significant reasons for this.

The yexpand3c tool is designed to possibly be run on a network of workstations. Because of this independent processes on distinct workstations may be producing the same (isomorphic) graphs. Elimination of isomorphic graphs in such an execution environment would be difficult. Therefore the elimination of the isomorphic graphs for a given parameter situation run must be performed upon its completion.

Because of the possibility of the long execution times for the enumeration tasks, the yexpand3c tool was designed to allow for the processes to be killed and restarted using the mark file feature. This feature is easily implemented if the process can simply produce Ramsey graphs without concern for possible isomorphisms.

Example 4.4.5.6.

The following command line executes the yexpand3c program. It attempts to enumerate (3,4,8,12)−graphs from (3,3,8-3-1,e)−graphs which are stored in G−Format graph files in the working directory of the executing process.

The "-t" flag specifies that the process should be started in resume mode and therefore it attempts to find a mark file left by a previous run for this specific parameter situation.

```
% yexpand3c -l 4 -n 8 -e 12 -t
```

The following is the output of the yexpand3c utility run which is written to the stdout file descriptor. The output indicates that at least one graph was written to the file g-3.4.8.12.3.y. The yexpand3c utility will create this file if it does not exist. If it exists, the file will be appended.

```
yexpand3c: run starting with k=3, l=4, n=8, e=12
yexpand3c: mark file open failed, restarting enumeration

( D(v) = [ *3* ], O(H2) = 4, S(H2) = [ 2 - 6 ] )
        processing: g-3.3.4.2.1.y
        processing: g-3.3.4.3.1.y
                writing: g-3.4.8.12.3.y
        processing: g-3.3.4.4.2.y

yexpand3c: run complete for k=3, l=4, n=8, e=12
```

□

yextend3c Program

The *yextend3c* program implements the EXTEND algorithm as described in Section §4.3.3. The utility attempts to enumerate $(3, l, n, e)-$graphs from existing $(3, l, n - X, e - Y)-$graphs for a given parameter situation. This program is primarily used to enumerate all $(3, l)$ Ramsey graphs with $3 \leq l \leq 6$. The $(3, 6)-$graphs are subsequently used as a set of base graphs for the yexpand3c program described in Section §4.4.5.

The utility uses the G−Format files in the process' current working directory which contain the required base graphs (i.e $(3, l, n - X, e - Y)-$graphs).

The utility accepts command line parameters that specify the parameter situation that the EXTEND algorithm is to be executed upon. The independent set free size (specified by the -l flag), order (specified by the -n flag), and size (specified by the -e flag) of the Ramsey graphs sought must be provided.

Example 4.4.5.7.

The following command line executes the yextend3c program. It attempts to enumerate all possible (3,6,17,42)−graphs from (3,6,17-X,42-Y)−graphs which are stored in G−Format graph files in the current working directory of the executing process.

```
% yextend3c -l 6 -n 17 -e 42
```

□

ygenerate Program

The *ygenerate* tool enumerates various graph theoretic structures with given parameters. The program generates graphs and degree sequences. The order as well as the size of the desired graphs and degree sequences must be specified.

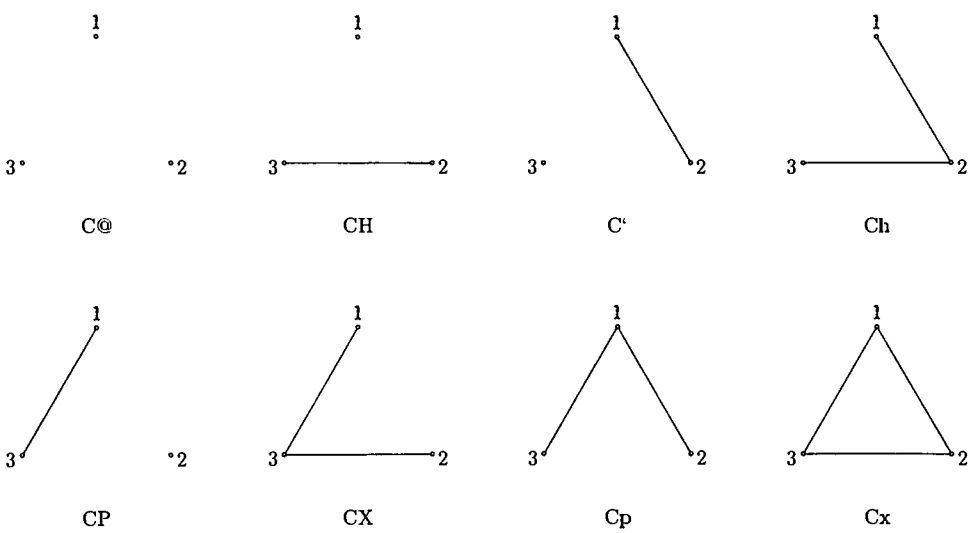
For a given parameter situation all labeled graphs and degree sequences are produced. All graphs which are produced are in the y−format. All degree sequences produced are ASCII strings which contain vertex degrees separated by spaces. Each of the *records* output are separated by a newline.

Example 4.4.5.8.

The following command pipeline generates all graphs of order 3 which have a size of 3. Note that the ygenerate utility enumerates all labeled graphs for the specified parameter situation.

```
% ygenerate -n 3 -e 3 | ydump -i pic
```

The first segment of the pipeline invokes the ygenerate tool to generate all order 3 and size 3 labeled graphs. The generated graphs are written to standard out which is then redirected to be the standard input of the ydump tool which then generates PIC groff pre−processor code.



□

ylabel Program

The *ylabel* program transforms an undirected graph into the graph's corresponding canonical form. The program takes as input the y−format for a undirected graph and outputs the corresponding canonical y−format graph. The code for the program is taken from the *labely* program designed and implemented by Brendan McKay.

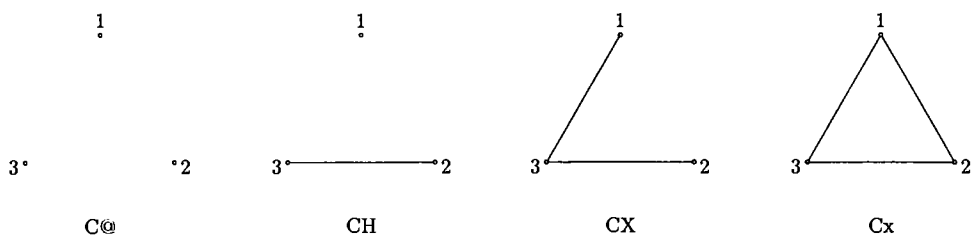
The primary purpose of the *ylabel* program is the elimination of isomorphic graphs. All graphs which are isomorphic will have the same canonical form. If a group of isomorphic y−format graphs is provided as input to the ylabel program, the same y−format output string will be produced for each of the graphs. This output can then be uniquely sorted (for instance by the UNIX command "sort -u") to produce only the non−isomorphic graphs.

Example 4.4.5.9.

The following command pipeline generates all non−isomorphic graphs of order 3 which have a size of 3.

```
% ygenerate -n 3 -e 3 | ylabel | sort -u \
    | ydump -i pic
```

The first segment of the pipeline invokes the ygenerate tool to generate all order 3 and size 3 labeled graphs. The second stage of the pipeline filters the y−format graphs produced through the ylabel program which converts the graphs to their canonical format. The graphs are then uniquely sorted (removing isomorphic graphs which have an identical canonical order). The subsequent output is redirected as input to the ydump tool which then generates PIC groff pre−processor code and text.



□

ysplit Program

The *ysplit* program is used to categorize or partition y−format Ramsey graphs into files based upon various graph criteria such as order, size, δ, etc.

The ysplit program may take as an argument a command line option of "-f <string>" where the <string> parameter specifies the format for the name of the graph files which will maintain the partitioned graphs. The <string> parameter may be straight text or may contain variables which are to be substituted with a value for a parameter related to the graph such as the graph's order, size, etc.

Example 4.4.5.10.

The following command pipeline enumerates all labeled graphs of order 3. A total of 8 graphs are produced. These graphs are then canonically ordered by the ylabel program with isomorphisms removed. A total of 4 graphs are produced. All of these non−isomorphic graphs are then provided as standard input to the split command.

```
% ygenerate -n 4 | ylabel | sort -u | ysplit
```

The following eleven graphs files are produced by the ysplit program, each containing one of the eleven non−isomorphic graphs:

```
% ls g*y

g-1.4.4.0.0.y  g-2.2.4.1.0.y  g-2.3.4.4.1.y
g-2.2.4.2.1.y  g-2.3.4.2.0.y  g-3.2.4.5.2.y
g-2.2.4.3.1.y  g-2.3.4.3.1.y  g-3.2.4.6.3.y
g-2.2.4.4.2.y  g-2.3.4.3.0.y
```

The following command pipeline demonstrates the use of the "-f" command line option to specify the graph file format. The same command pipeline as in the above example is used with the exception of the addition of the "-f" option.

```
% ygenerate -n 4 | ylabel | sort -u \
     | ysplit -f "g-x.x.%n.%e.x"
```

This example produces the same graphs as the above examples. The graphs are simply partitioned differently because of the "-f" option. Note that in the above command line the "%n" and "%e" parameters in the graph file string template will be substituted with the order and size of the graph respectively. The following graph files are produced.

```
% ls g*y

g-x.x.4.0.x.y  g-x.x.4.3.x.y  g-x.x.4.6.x.y
g-x.x.4.1.x.y  g-x.x.4.4.x.y
g-x.x.4.2.x.y  g-x.x.4.5.x.y
```

□

4.4.6. grama Test Suite

The *grama test suite* is a set of 9 *C* programs and 9 Perl scripts which are used for demonstrating the capabilities of the grama package as well as testing the functionality and correctness of the library.

For each of the *C* programs a corresponding Perl script, which performs the same function and computes the same result, is available. The implementation in Perl is provided to demonstrate and test the functionality of the SWIG generated interface to the grama *C* libraries. The correct output for each of the programs/scripts is available as part of the package. A full reference to the SWIG package can be found in [BEA97a].

The test suite consists of a UNIX *Makefile* (see make(1)) which can be used to build the programs/scripts. The Makefile also contains *targets* which can be used to execute each program/script and compare the output to the pre–established "correct output". The Makefile is automatically generated during the package configuration process as described in Section §4.4.1.

The source code listings for the first of the nine test programs implemented in *C* is presented in Appendix A. The following is a brief summary of the function performed by each of the programs/scripts.

ydemo0

> *ydemo0* instantiates a graph G of order 6. Various vertices are attached. The incidence matrix for each graph is then calculated from the adjacency matrix. Note that for the grama package, the adjacency matrix is the default structure utilized for maintaining a graph's vertex connections. The adjacency matrix, as well as the incidence matrix, are written to the output. The distance matrix, column characteristic matrix, row characteristic matrix, characteristic matrix, class vector, and the class count vector for the original graph is calculated and written to the output.

ydemo1

> *ydemo1* instantiates two graphs G and H. The standard graph composition, join, product, and union operations are performed between G and H. Both compositions of the two graphs, G with H and H with G are calculated. The five graphs resulting from these operations are written to the output.

ydemo2

ydemo2 reads degree sequences from standard input until the EOF marker is reached. Each of the degree sequences read is realized and the y−format for all possible labeled graphs is written to the output.

ydemo3

ydemo3 instantiates two graphs, denoted G and H. For each of the two graphs the class vector and the class count vector are calculated and written to output. The class vector for G is calculated relative to the class vector for H, and this relation is written to the output.

ydemo4

ydemo4 reads a clique, an independent set, and two degree sequences from standard input. The ASCII whitespace separated format for these various structures is required as input. The output routine for the each of the respective objects is then invoked and each structure is written to the output.

ydemo5

ydemo5 reads y−format graphs from standard input until the EOF marker is reached. The GIF image representation of each graph is output. Each graph GIF image representation is output to a distinct file using a file name template of *ydemo5.%d.gif* where the variable "%d" is substituted with the index of order in which the graph was read from input.

ydemo6

ydemo6 instantiates a graph G, as well as the graph's 3−clique set, denoted K, and the graph's 3−independent set, denoted L. The GIF image representations of K and L are then each output to a separate file, where the file name is based upon the template *ydemo6.out%d.gif*, with the variable "%d" being an index ranging from 1..2.

ydemo7

ydemo7 instantiates two graphs, G and H, both of order 5. Various vertices are attached in the graphs so that the resulting graphs are each of size 5. The adjacency matrices for both G and H are then written to the output. GIF image representations of both G and H are produced and each output to a file based upon the file name template *ydemo7.out%d.gif*, with the variable "%d" being an index ranging from $1..2$. An isomorphic mapping, denoted M, between the two graphs G and H is calculated and written to the output.

ydemo8

ydemo8 instantiates a graph G. The minimum number of edges for a given $(3,3)$ Ramsey graphs with $n = \{0..9\}$ is calculated and output. Possible Ramsey graph degree sequences for $(3,l,n)$ parameter situations with $l = \{1..8\}$ and $n = 4$ are calculated and written to the output.

4.5. Third Party Software Packages

This section contains a description of all of the third party software packages which were utilized throughout the calculations performed for this thesis. Some of the software packages are *OpenSource* projects under various licenses. Others are marketed products and must be purchased.

4.5.1. autoson Package

The *autoson* package is a tool for scheduling processes across a network of UNIX workstations. It provides a form of distributed batch queue that enables the execution of a stream of processes in a flexible and convenient manner with minimum impact on interactive users.

Autoson can execute independent processes on heterogenous and/or unreliable processors. It does not provide a parallel environment that allows processes to communicate. It simply executes jobs and waits for each job's completion.

See *http://cs.anu.edu.au/people/bdm/autoson/* for a more in depth description of the autoson package. [McK96a] The autoson package was used for the distributed execution of processes to enumerate $(3,3),(3,4),(3,5),(3,6),(3,7),(3,8)$, and $(3,9)$ graphs. An autoson execution environment was implemented for both the EXTEND and EXPAND algorithms.

4.5.2. gtools Package

The *gtools* package is a suite of programs for processing graphs with an emphasis on efficient processing of large amounts of data. One such program is *geng*, which

is a very fast graph generator. The gtools requires that *nauty* version 2.0, described in Section §4.5.4., be installed.

The gtools package provides tools for the conversion of graphs from and to various formats, including the y−format. It provides features which include the removal of isomorphisms and the generation of graph of various orders. These features were used to verify software which was implemented by the author including various components of the grama package.

See *http://cs.anu.edu.au/people/bdm/nauty/* for a more in depth description of the gtools package.

4.5.3. Maple V Package

The *Maple V* package is a computational algebra system which provides features for symbolic, numeric, and graphical mathematical computation. It is an on−going project of the Symbolic Computing Group of the University of Waterloo, Waterloo Canada, and the Institute for Scientific Computing at ETH Zürich. [HEA98a]

See *http://www.maplesoft.com/* for a more in depth description of the Maple V package.

4.5.4. nauty Package

The *nauty* (**no aut**omorphism, **y**es?) package is a set of utilities for computing the automorphism group of a vertex−colored graph. It provides this information in the form of a set of generators, the size of the group, as well as the orbits of the group. It is able to produce a canonically−labeled isomorph of a graph to be used in isomorphism testing.

See *http://cs.anu.edu.au/people/bdm/nauty/* for a more in depth description of the nauty package. [MCK90a]

Chapter 5. Execution

The calculation of the described Ramsey graphs requires a great number of CPU cycles. To minimize the amount of time needed for the calculations and to efficiently use the computing power which was available to the author, a network of distributed UNIX® workstations was utilized.

This chapter details the various execution processes used for the enumeration of Ramsey graphs. Details regarding the initialization of graphs needed for the *priming* of the algorithms, the directory structure utilized for the execution of the distributed processes, as well as the networking facilities required are detailed.

5.1. Enumeration of (3,3), (3,4), (3,5), and (3,6) Graphs

This section details the execution process used for the enumeration of all $(3,3)$, $(3,4)$, $(3,5)$, and $(3,6)$ Ramsey graphs. All of the processes described in this section use the EXTEND algorithm as described in Section §4.3.3.

Graph Priming Tasks

The enumeration of graphs using the EXTEND algorithm requires that $(3,l,n)$ graphs for $n < l$ be already calculated. There graphs are used as input to the EXTEND algorithm and are *extended* to build the larger graphs.

All $(3,l,n)$ Ramsey graphs with $3 \leq l \leq 6$ and $n < l$ are trivial and can be enumerated by calculating all graphs of order n and eliminating those which contain cliques of size 3 and/or independent sets of size l. The following command pipeline was used to enumerate all non−isomorphic $(3,3,2)$−graphs. A similar command line was used to enumerate all of the $(3,l)$ *base* graphs needed.

```
ygenerate -n 2 | ylabel | sort -u | yeliminate -k 3 -l 3 | ysplit
```

This created a total of 3 $(3,3)$−graphs, 6 $(3,4)$−graphs, 13 $(3,5)$−graphs, and 27 $(3,6)$−graphs. The ysplit program takes as input graphs in the y−format. It examines each graph's properties such as clique and independent set free size, order, size, and minimum degree and outputs each graph to a G−Format file, the name of which represents each of these properties. This command line provides a convenient tool for creating the base graphs needed.

Network Facility Requirements

The execution of the yextend3c program in distributed mode requires the use of the autoson package from Brendan McKay. For autoson to be implemented and

used across a network of possibly heterogeneous UNIX workstations the following are required.

- A user account of each of the workstations upon which the yextend3c process is to be executed.
- A common User ID (UID) under which the processes are to be executed.
- An NFS exported filesystem which is mounted on each of the workstations.

The RIT Computer Science Department (RIT–CS) distributed UNIX workstation system provides such an environment. For the execution of the graph enumerations for this thesis the RIT–CS system was used.

YEXTEND Local Mode Execution

Figure 5.1.1. demonstrates the execution of the EXTEND algorithm on a single node. This method of execution does not utilize the autoson package to create the batch environment needed for the distribution of the various jobs.

In the diagram the oval objects represent UNIX processes in execution.

The oval objects represent a single process. The dashed lines between the ovals represent the spawning of a process. The process from which the dashed arrow originates designates the parent process while the destination of the arrow designates the child (or spawned) process.

Each of the single edged boxes represents a file which contains one or more y–format Ramsey graphs. The double edged box represents the holding directory. This directory is used to maintain y–format Ramsey graph files before the isomorphic graphs have been eliminated.

The YEXTEND enumeration process is initiated with the execution of the coordinating yerg3c program. This program requires two command line parameters which specify the value of k and l for a given (k, l) situation. The yerg3c calculates the range of valid orders and sizes for the given graph parameter situation. The yerg3c program spawns a yextend3c process for each such scenario.

The yextend3c program processes all $(3, l, n)$ graphs required for the enumeration of $(3, l, n+1)$ graphs. Upon completion of the YEXTEND algorithm, a ylabel processes is spawned. This process examines all of the y–format files which were calculated. It removes all isomorphic graphs and appends/creates graph files in the current working directory with the newly enumerated graphs. These graph files are represented in the diagram by the rightmost box. The resulting graphs are used as the base graphs for the next iteration of the EXTEND algorithm. This is designated in the figure by the arrow from the rightmost box to the leftmost box (i.e. from the output graph box to the input graph box).

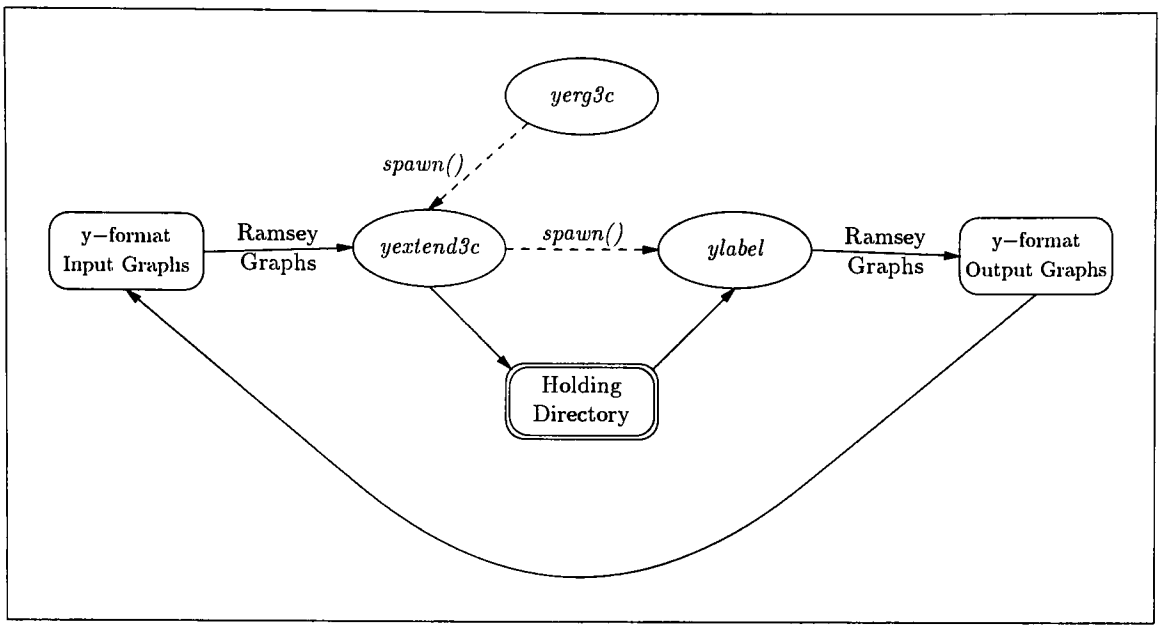


Figure 5.1.1. – YEXTEND Local Execution Mode Dataflow.

YEXTEND Distributed Mode Execution

Figure 5.1.2. demonstrates the execution of the EXTEND algorithm on a collection of possibly heterogeneous UNIX® workstations. This process differs from local mode execution in that the autoson package is used to create a queue of batch jobs which will perform the needed enumeration.

In the diagram the oval objects represent UNIX® processes in execution.

The solid oval object represents a process which executes on a single controlling workstation. The dashed oval objects represent processes which may possibly be executed on multiple workstations simultaneously if the yextend3c process is executed in distributed mode. The dashed lines between the ovals represent the spawning of a process. The process from which the dashed arrow originates designates the parent process while the destination of the arrow designates the child process.

Each of the single edged boxes represents a file which contains one or more y–format Ramsey graphs. The double edged box represents the holding directory. This directory is used to maintain y–format Ramsey graph files before the isomorphic graphs have been eliminated.

The execution process is similar to that described for Local Mode Execution in the previous sub–section. Instead of a single yextend3c process being forked by

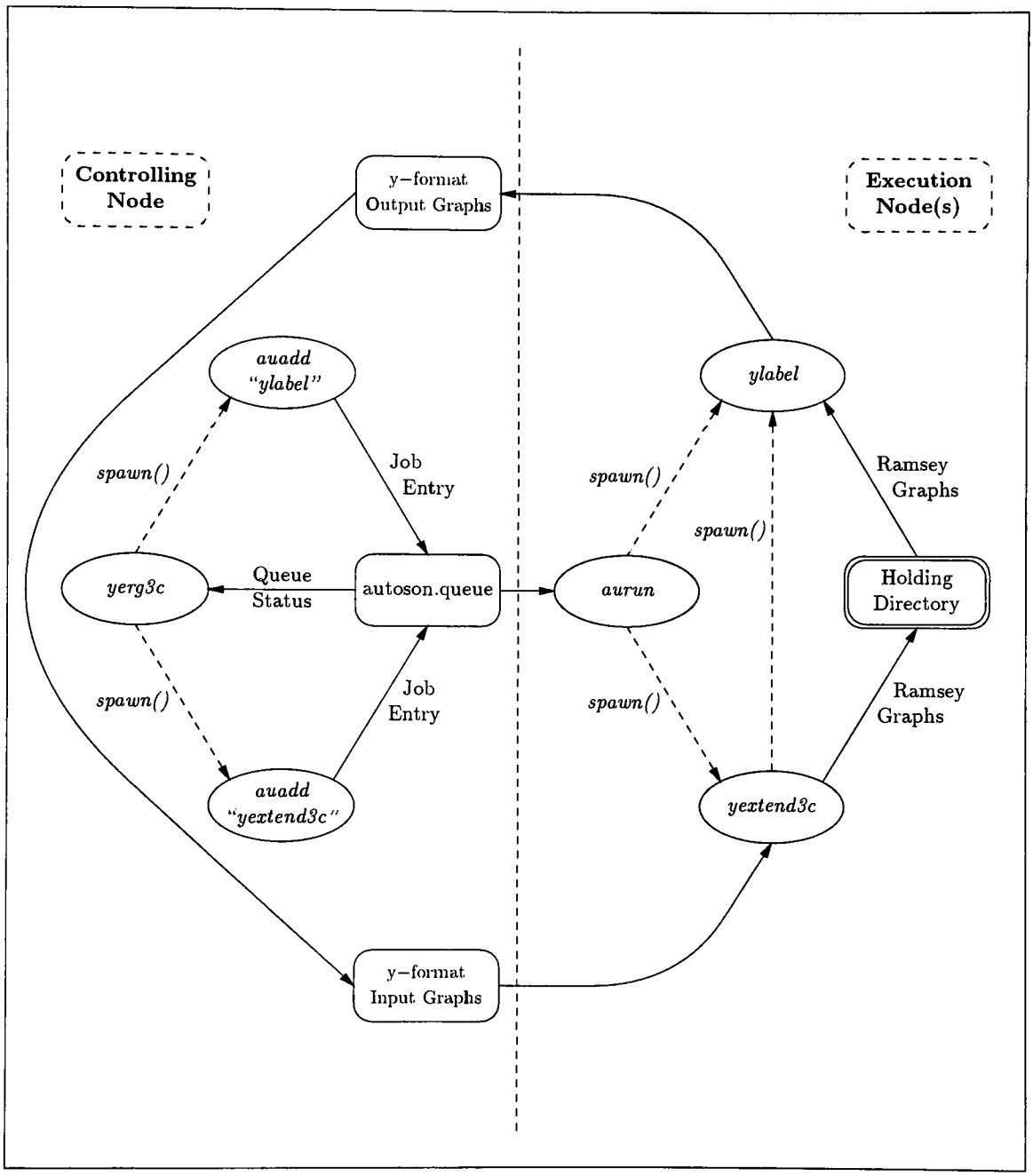


Figure 5.1.2. – YEXTEND Distributed Execution Mode Dataflow.

the yerg3c program, a series of autoson job are created. These jobs are entered into the autoson queue and executed by the available networked workstations.

The yerg3c coordinating program *waits* for the completion of all of the autoson jobs which were queued. Upon completion of the last autoson queued yextend3c job executing across the network, a ylabel processes is spawned. This process examines all of the y−format files which were calculated. It removes all isomorphic graphs and appends/creates graph files in the current working directory with the newly enumerated graphs. These graph files are represented in the diagram by the rightmost box. The resulting graphs are used as the base graphs for the next iteration of the EXTEND algorithm.

5.2. Enumeration of (3,7), (3,8), and (3,9) Graphs

This section details the execution process used for the enumeration of all (3,7), (3,8), and (3,9) Ramsey graphs. All of the processes described in this section use the EXPAND algorithm as described in Section §4.3.4.

Graph Priming Tasks

The graph priming tasks for the YEXPAND differ from those needed for the YEXTEND algorithm described in Section §4.3.3. and §4.4.5. This algorithm utilizes the (3,6) graphs enumerated by the YEXTEND algorithm to produce some (3,7), (3,8), and (3,9) Ramsey graphs.

Network Facility Requirements

The execution of the yexpand3c program in distributed mode requires the use of the autoson package from Brendan McKay. For autoson to be implemented and used across a network of possibly heterogeneous UNIX workstations the following are required.

- A user account of each of the workstations upon which the yexpand3c process is to be executed.
- A common User ID (UID) under which the processes are to be executed.
- An NFS exported filesystem which is mounted on each of the workstations.

The RIT Computer Science Department (RIT−CS) distributed UNIX workstation system provides such an environment. For the execution of the graph enumerations for this thesis the RIT−CS system was used.

YEXPAND Local Mode Execution

The YEXTEND execution process described in the previous section utilized a coordinating script. This script automated the enumeration of all $(3,l)$ graphs for $l \leq 6$. For $l \leq 6$ the enumeration process requires little computational time compared to the enumeration process for $(3,l)$ Ramsey graphs with $l > 7$.

Because the enumeration of $(3, l)$ Ramsey graphs for $l > 7$ is computationally challenging, no coordinating script exists since its run time would be unacceptably long. Therefore the yexpand3c command line program provided by the ytool package requires that all of the parameters for a specific parameter situation be specified.

For each $(3, l, n)$ situation the parameters l and n must be provided to the yexpand3c command line tool. The yexpand3c program attempts to enumerate all valid $(3, l, n)$ Ramsey graphs. The yexpand3c program requires that the $(3, l-1, n-d-1)$ graphs which form the set of possible $H_2(v)$ sub−graphs exist in G−Format files in the working directory.

In local execution mode the yexpand3c program simply requires the two parameters l and n as well as read access to the $(3, l-1, n-d-1)$ Ramsey graphs for the given parameter situation. If the $(3, l-1, n-d-1)$ graphs do not exist, the execution of the yexpand3c will still succeed but the expected output may not be produced.

Because of the long running nature of the YEXPAND algorithm, processes executing the yexpand3c program may be abruptly terminated unexpectedly. Therefore the yexpand3c writes a *mark file* which maintains the process state for the given enumeration parameter situation. The mark file name reflects the parameter situation which is being explored. For instance, if an enumeration was begun with the command *"yexpand3c -l 7 -n 16 -e 20"*, a mark file named *".g-3.7.16.20.m"* would be created in the same directory. This file would contain the state of the enumeration. If the yexpand3c process were to be terminated this file could be utilized to *restart* the enumeration at a point in time slightly previous to the process termination time.

The yexpand3c program does not eliminate isomorphisms between graphs. If the command *"yexpand3c -l 7 -n 16 -e 20"* were executed any resulting Ramsey graphs would be output to the files *"g-3.7.16.20.x.y"* in the same directory. The x character in this file name represents the δ (minimum degree) of all of the graphs contained within. The resulting G−Format graph files may contain graph isomorphisms. Any output files created by the EXPAND process must be filtered through the ylabel program to have any isomorphic graphs removed.

YEXPAND Distributed Mode Execution

As described in the previous section, the YEXPAND process is computationally long compared to the YEXTEND process and therefore no generic automation script is provided. Because of the difference in the running times of the various algorithms, a different tact was used for the distributed execution of the YEXPAND algorithm.

An option provided with the yexpand3c program provides a listing of the processing actions that *would* take place for a given YEXPAND algorithm run. When this option is utilized the enumeration for the specified parameter situation does not take place. One of the pieces of data which can be gleaned from the list of actions which would be performed is that of the $(3, l-1, n-d-1)$ Ramsey graph files which would be processed (i.e. the $H_2(v)$ graphs processed).

Once a list of the graphs which need be processed for the given parameter situation is determined, the granularity of the number of yexpand3c jobs required may be tuned. The yexpand3c program may be executed with two additional command line parameters. These two parameters are *"-d"* and *"-g"*. The *"-d"* parameter specifies the degree of the preferred vertex v which is to be attached to the $H_1(v)$ sub−graph. The ``*-g*" command line option specifies the file from which $(3, l-1, n-d-1)$−graphs are to be processed.

These two options allow for a *set* of yexpand3c jobs to be specified. These jobs may be added to an autoson queue and distributed over a set of computing nodes which share an NFS mounted filesystem.

Section §4.5.1. details methods which can be used to configure the autoson package to achieve this.

Chapter 6. Results

This chapter contains a summary of the Ramsey graphs enumerated by the author. All graphs were enumerated using the algorithms described in Section §3.1. as well as the software documented in Section §4.4.5.

Please note that for all tables the left x−axis contains the orders for the respective graphs while the top y−axis contains the sizes for the respective graphs. The right x−axis contains a count total of the number of graphs for the given order of each row and the bottom y−axis contains a count total of the number of graphs for the given size of each column.

Some pictorial representations of graphs are provided. All such representations were generated in canonical format using the ylabel program. This program is described in Section §4.4.5. The label under each pictorial representation is that of the corresponding graph's y−format string.

A valid character within the scope of the y−format set is a DELETE character (ASCII value 127_{10}). This character is typically displayed on UNIX® terminals as the character pair ^?. This character does not display well in text documents. It is difficult to distinguish between the two separate characters ^ and ?. For the purposes of this document all DELETE characters which are present in graph y−format representations are displayed with the character †.

All graphs are available in various formats upon request of the author.

6.1. (3,3), (3,4), (3,5), and (3,6) – Ramsey Graphs

This section contains four tables which summarize the counts all of non−isomorphic (3,3), (3,4), (3,5), and (3,6) Ramsey graphs enumerated by the author. Note that these are all of the Ramsey graphs which exist for the given parameter situations. These results match those originally cataloged by Radziszowski and Kreher in [RAD88a].

All of these graphs were enumerated using the EXTEND algorithm as originally specified in [RAD88a]. Pseudo code documenting this algorithm is provided in Section §4.3.4. while its implementation is described in Section §4.4.5.

There are a total of 9 (3,3)−graphs, 48 (3,4)−graphs, 1029 (3,5)−graphs, and 761692 (3,6)−graphs. The calculation of these graphs duplicates and verifies the enumeration results reported in [RAD88a].

size	*order n*					*total*
e	1	2	3	4	5	
0	1	1				2
1		1	1			2
2			1	1		2
3				1		1
4				1		1
5					1	1
total	1	2	2	3	1	9

Table 6.1.1. – Number of $(3,3,n,e)$–graphs, g_{33}.

size	*order n*								*total*
e	1	2	3	4	5	6	7	8	
0	1	1	1						3
1		1	1	1					3
2			1	2	1				4
3				2	2	1			5
4				1	3	1			5
5					2	4			6
6					1	4	1		6
7						3	2		5
8						1	3		4
9						1	2		3
10							1	1	2
11								1	1
12								1	1
total	1	2	3	6	9	15	9	3	48

Table 6.1.2. – Number of $(3,4,n,e)$–graphs, g_{34}.

size	*order n*													*total*
e	1	2	3	4	5	6	7	8	9	10	11	12	13	
0	1	1	1	1										4
1		1	1	1	1									4
2			1	2	2	1								6
3				2	3	3	1							9
4				1	4	6	2	1						14
5					2	8	7	1						18
6					1	7	13	5						26
7						4	17	13	1					35
8						2	15	27	3					47
9						1	10	39	11					61
10							4	41	28	1				74
11							1	27	59	2				89
12							1	15	73	10				99
13								6	62	32				100
14								2	33	69				104
15								1	14	86	1			102
16								1	4	65	6			76
17									2	32	19			53
18										12	31			43
19										3	30			33
20										1	13	1		15
21											4	2		6
22											1	5		6
23												2		2
24												2		2
25														0
26													1	1
total	1	2	3	7	13	32	71	179	290	313	105	12	1	1029

Table 6.1.3. – Number of $(3,5,n,e)$–graphs, g_{35}.

size	*order n*																	*total*
e	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	
0	1	1	1	1	1													5
1		1	1	1	1	1												5
2			1	2	2	2	1											8
3				2	3	4	3	1										13
4				1	4	7	7	3	1									23
5					2	9	13	10	2	1								37
6					1	7	20	23	8	1								60
7						4	20	44	26	5								99
8						2	18	63	70	16	1							170
9						1	11	73	142	60	3							290
10							5	63	234	175	16							493
11							1	40	284	451	64	1						841
12							1	21	267	864	265	4						1422
13								9	185	1255	900	20						2369
14								3	106	1344	2353	119						3925
15								2	47	1114	4444	644	1					6252
16								1	22	707	6134	2693	4					9561
17									8	377	6239	7968	45					14637
18									3	167	4823	16445	375					21813
19									1	71	2885	23986	2402					29345
20									1	28	1405	25267	10176	1				36878
21										13	565	19704	27975	16				48273
22										4	206	11672	51188	177				63247
23										2	64	5404	64221	1588				71279
24										1	20	2016	56809	8494				67340
25										1	6	630	36312	27013	1			63963
26											2	169	17208	53157	7			70543
27												41	6189	67224	101			73555
28												8	1729	56478	822			59037
29												1	377	32235	3998			36611
30													66	12784	10910			23760
31													8	3550	17552			21110
32													1	699	16896	5		17601
33														94	9957	39		10090
34														9	3587	200		3796
35														1	794	547		1342
36															100	803		903
37															7	634		641
38																275		275
39																62		62
40																11	2	13
41																	3	3
42																	2	2
total	1	2	3	7	14	37	100	356	1407	6657	30395	116792	275086	263520	64732	2576	7	761692

Table 6.1.4. – Number of $(3, 6, n, e)$–graphs, g_{36}.

6.2. (3,7) Ramsey Graphs

6.2.1. (3,7) Ramsey Graph Summary

This section contains a summary of the non−isomorphic $(3,7)$ Ramsey graphs enumerated by the author.

All of these graphs were enumerated using the EXPAND algorithm as originally specified in [GRA68a] and further detailed in [GRI82a], [RAD88a], and [RAD88b]. The theoretical framework for this enumeration algorithm is specified in Section §3.1. Pseudo code documenting this algorithm is provided in Section §4.3.4. and its implementation is described in Section §4.4.5.

The complete catalogue of $(3,6)$ graphs, which is documented in the previous section, forms the pool of $H_1(v)$ input graphs utilized for the EXPAND algorithm.

All $(3,7,n)$−graph enumerations and counts for $n = 16..21$ with $e = e(3,7,n)$ and $e = e(3,7,n)+1$, as well as the 717 $(3,7,21,53)$−graphs, and all 191 critical $(3,7,22)$−graphs duplicate and verify the results reported by Radziszowski and Kreher in [RAD88b].

The $(3,7,n)$−graphs for $n = 16..21$ with $e = e(3,7,n)+2$ and $e = e(3,7,n)+2$, with the exception of the 717 $(3,7,21,53)$−graphs, are believed to be newly enumerated graphs (i.e. not enumerated during any other investigations).

Table 6.2.1. provides a summary of all $(3,7)$−graphs enumerated. Note that the values for the graph order in the left x−axis are discontiguous. Graph counts enumerated by the author which were previously unknown are prefixed with a † character.

size	*order n*							*total*
e	16	17	18	19	20	21	22	
20	2							2
21	15							15
22	†201							201
23	†2965							2965
25		2						2
26		30						30
27		†642						642
28		†13334						13334
30			1					1
31			15					15
32			†382					382
33			†8652					8652
37				11				11
38				417				417
39				†10447				10447
40				†172534				172534
44					15			15
45					479			479
46					†10119			10119
47					†132965			132965
51						4		4
52						70		70
53						717		717
54						†5167		5167
60							1	1
61							6	6
62							30	30
63							60	60
64							59	59
65							25	25
66							10	10
total	3183	14008	9050	183409	143578	5958	359186	191

Table 6.2.1.1. – Number of $(3,7,n,e)$–graphs, g_{37}.

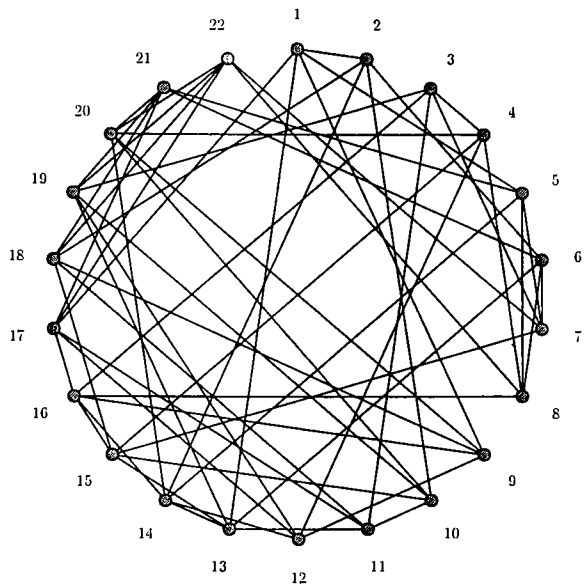
6.2.2. Critical (3,7) – Ramsey Graphs

This section contains pictorial representations of some non–isomorphic critical $(3,7)$ graphs. The value of the Ramsey number $R(3,7) = 23$. All Ramsey graphs on 22 vertices are considered critical graphs.

There are a total of 191 critical $(3,7,22)$–graphs. The sizes of the critical graphs are $e = 60, 61, 62, 63, 64, 65$, and 66 with the counts for each $(3,7,22,e)$ situation being $1, 6, 30, 60, 59, 25$, and 10 respectively.

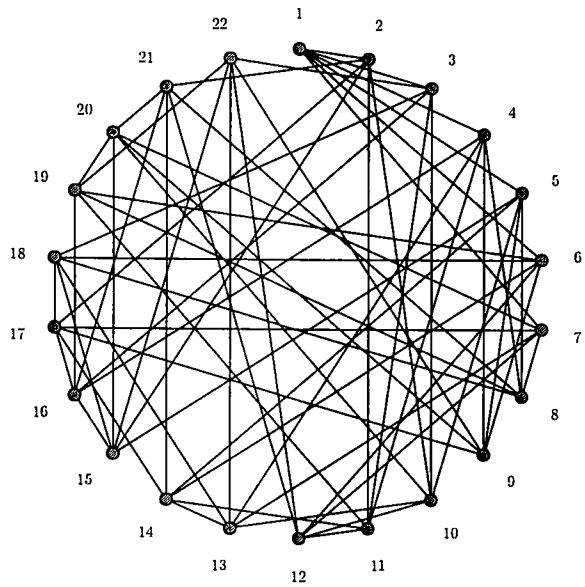
This section contains pictorial representations for all of the minimum and maximum critical $(3,7)$ graphs. A total of 1 $(3,7,22,60)$ and 10 $(3,7,22,66)$ graphs are presented.

(3,7,22,60) – Ramsey Graph, 1 of 1

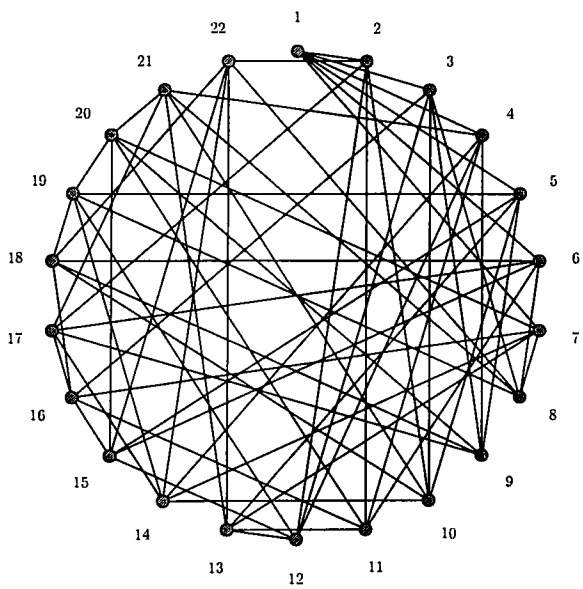


VaaAXz@PA@bDaBRAbRQCC`CE@ha@l@ab@L@O@pCp

(3,7,22,66) – Ramsey Graphs, 1 and 2 of 10

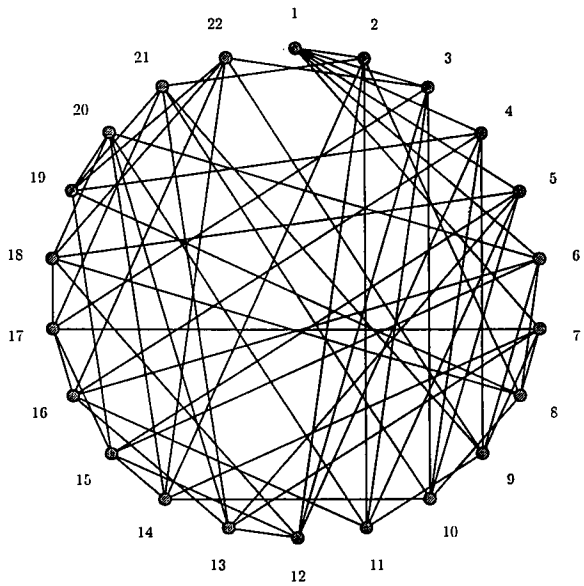


VtbD@t\ZC`@sDdCBi@A`APhTeBhJP`EaEAEQHIh`

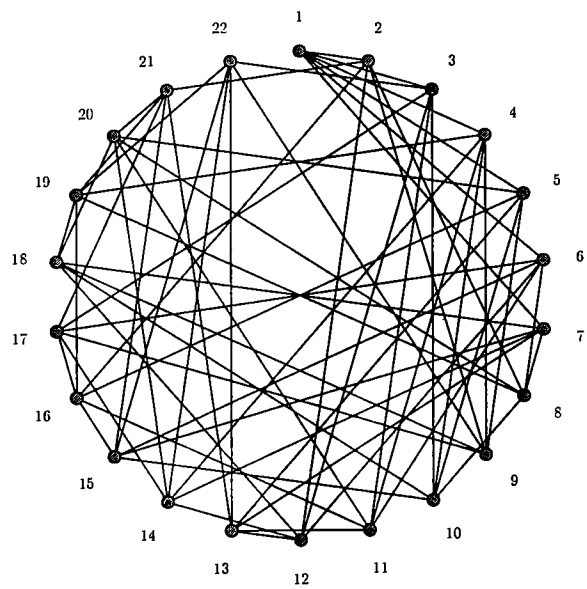


VtbDALxZBhN@BcDdA`aDQQHTDpPRJHEQDQRIPPy@

(3,7,22,66) – Ramsey Graphs, 3 and 4 of 10

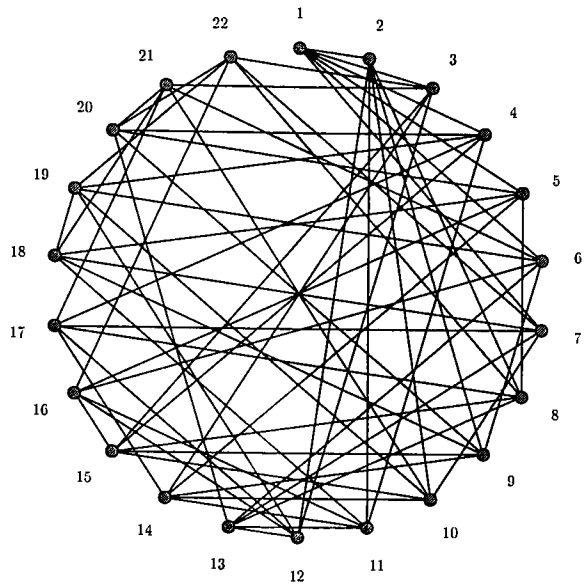


VtbDBLVNSQN@BaPdA`hhTH`LIDHbA@HVM@rJHHU`

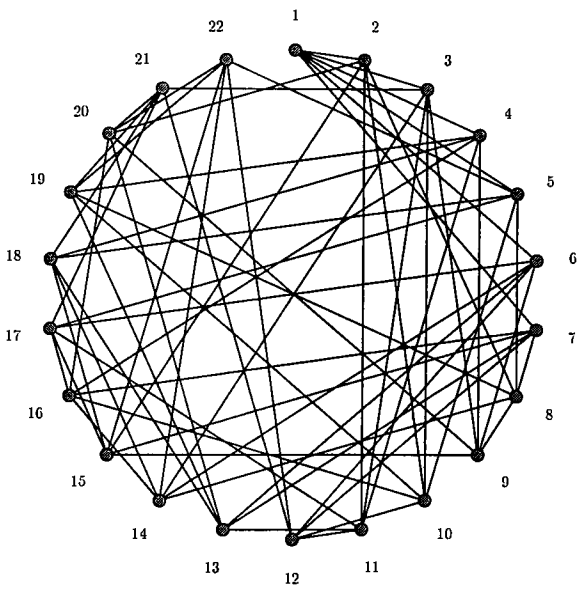


VtbDBMXNQhL`BcDa@rBPQIHTBt@b@hRRI@BkHHx`

(3,7,22,66) – Ramsey Graphs, 5 and 6 of 10

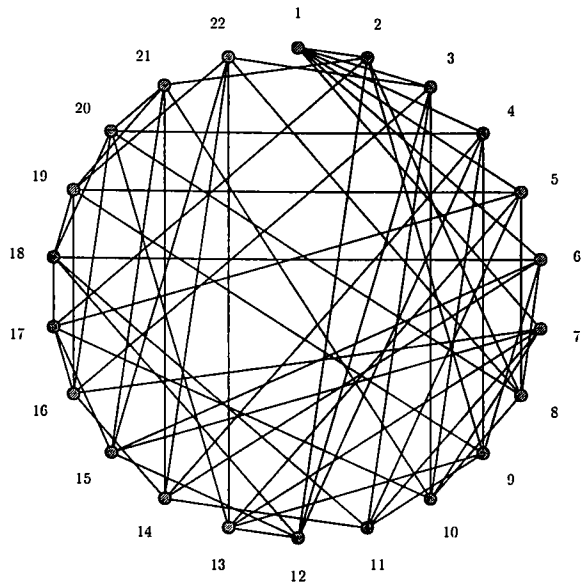


VtbDBQDPb`L@AsJNFN@XYDqPJh@hhHqD@dPMHpDp

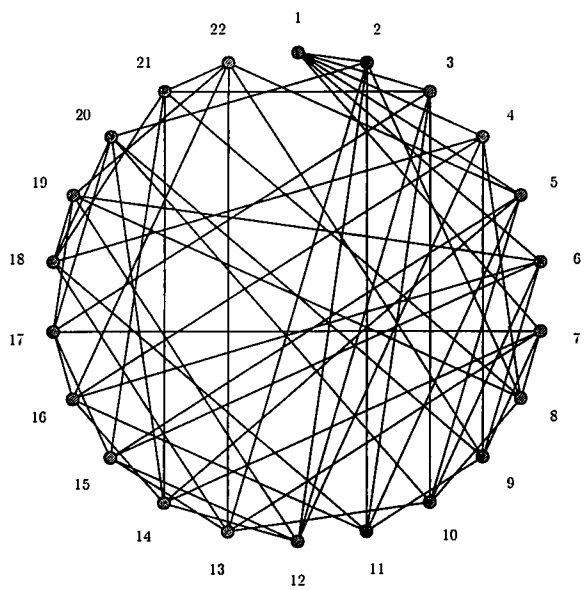


VtbDBTuZC`@sAbIPHT@dbCBLXC`bdBAD``DOBAXp

(3,7,22,66) – Ramsey Graphs, 7 and 8 of 10

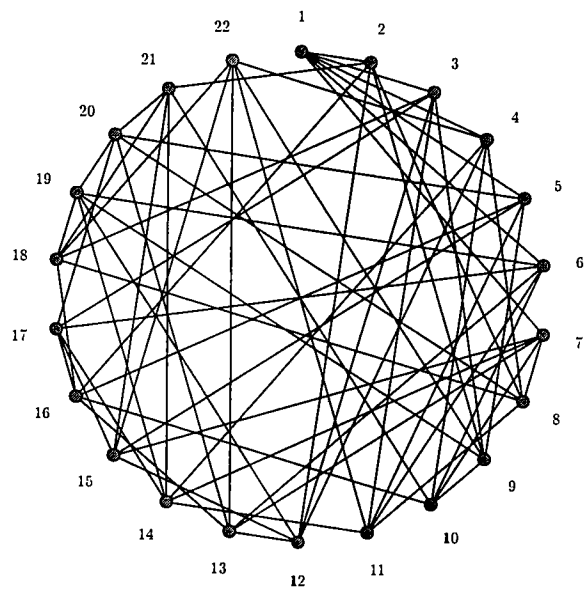


VtbDBYTLqUN@BiEB@paDBRDLDLHQ@hbDi@QcHPx`

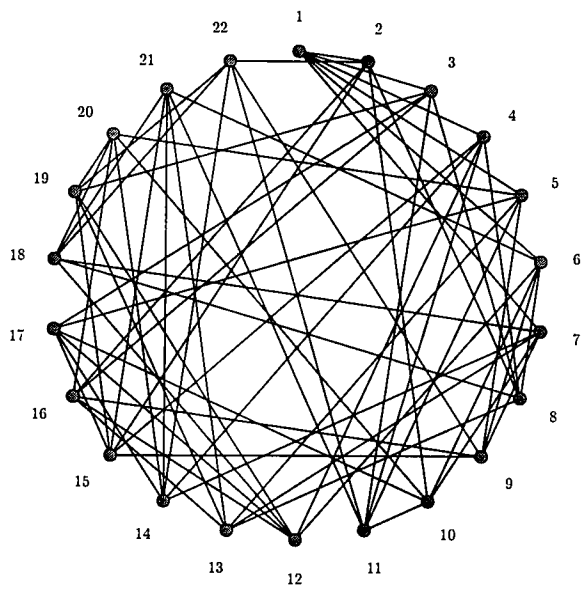


VtbDBdVKSQN@PdBaA`phRH`LPJ@JHZAbXaAdBHdh

(3,7,22,66) – Ramsey Graphs, 9 and 10 of 10



VtbDBehEiZN@AaDbAPbPdI@\a@PIJHRDU@QeDJi@



VtbDChNQapb`BpD`JDCALJEXCDA@M@PblDKdPJQ`

6.3. (3,8) Ramsey Graphs

6.3.1. (3,8) Ramsey Graph Summary

This section contains a summary of the non−isomorphic $(3,8)$ Ramsey graphs enumerated by the author.

All of these graphs were enumerated using the EXPAND algorithm as originally specified in [GRA68a] and further detailed in [GRI82a], [RAD88a], and [RAD88b]. The theoretical framework for this enumeration algorithm is specified in Section §3.1. Pseudo code documenting this algorithm is provided in Section §4.3.4. and its implementation is described in Section §4.4.5.

The catalogue of $(3,7)$ graphs, which is documented in the previous section, forms the pool of $H_1(v)$ input graphs utilized for the EXPAND algorithm.

The 396 minimum $(3,8,25,65)$-graph, 62 minimum $(3,8,26,73)-$graph, and 4 $(3,8,27,85)-$graph enumerations duplicate and verify the calculations reported by Radziszowski and Kreher in [RAD88b].

For $(3,8,n)$ with $n = 20,21,22,23,$ and 24 and $e = e(3,8,n)$ a total of $3,1,21,102,$ and 51 previously unenumerated graphs were calculated. 101 $(3,8,23,49)-$graphs were reported enumerated by Radziszowski (private communication). This authors calculations increase this count by 1. All $(3,8,n)-$graphs for $n = 18..26$ with $e = e(3,8,n)+1$ are believed to be newly enumerated graphs (i.e. not enumerated during any other investigations).

Table 6.3.1. provides a summary of all $(3,8)-$graphs enumerated. Note that the values for the graph order in the left x−axis are discontiguous. Graph counts enumerated by the author which were previously unknown are prefixed with a † character.

size	*order n*								*total*
e	20	21	22	23	24	25	26	27	
30	†3								3
31	†60								60
35		†1							1
36		†20							20
42			†21						21
43			†1521						1521
49				†102					102
50				†8204					8204
56					†51				51
57					†2071				2071
65						396			396
66						†19163			19163
73							62		62
74							†1458		1458
85								4	4
86								†74	74
87								†301	301
88								†334	334
89								†204	204
90								†228	228
91								†93	93
92								†18	18
total	63	21	1542	8306	2122	19559	1520	1256	34389

Table 6.3.1.1. – Number of $(3,8,n,e)$–graphs, g_{38}.

6.3.2. Critical $(3,8)$ – Ramsey Graphs

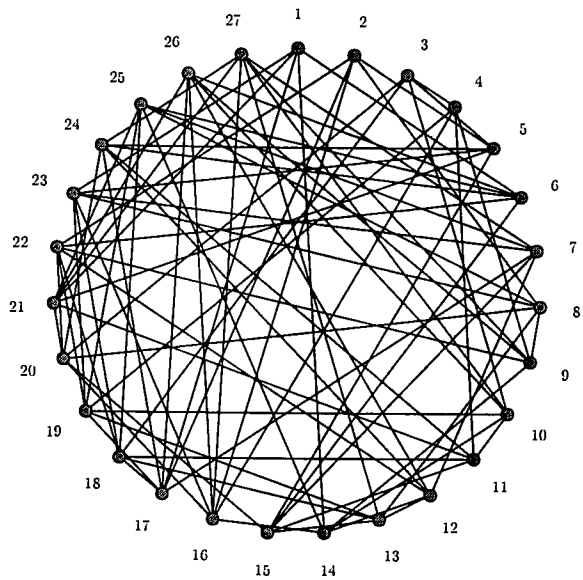
This section contains pictorial representations of some non–isomorphic critical $(3,8)$ graphs. The value of the Ramsey number $R(3,8) = 28$. All Ramsey graphs on 27 vertices are considered critical graphs.

There are a total of 1256 critical $(3,8,27)$–graphs. The sizes of the critical graphs are $e = 85, 86, 87, 88, 89, 90, 91$, and 92 with the counts for each $(3,8,27,e)$ situation being $4, 7, 301, 334, 204, 228, 93$, and 18 respectively. However more graphs for these parameter situations may exist.

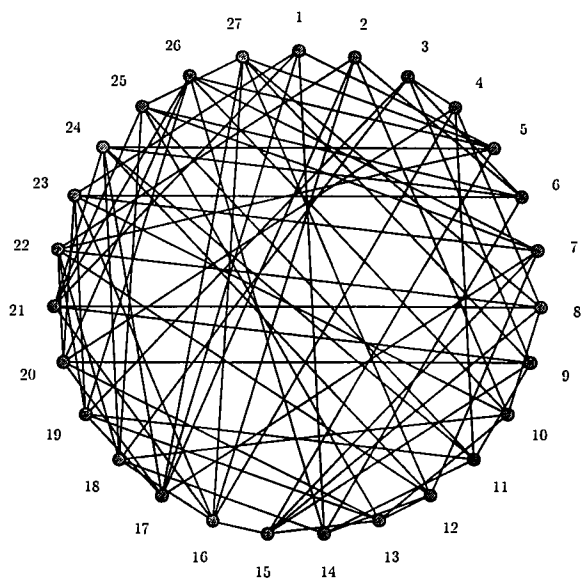
The exact value for $e(3,8,27)$ has yet to be established. The bounds for this value are currently $83 \leq e(3,8,27) \leq 85$.

This section contains pictorial representations for all of the minimum and maximum critical $(3,8)$ graphs which were enumerated. A total of 4 $(3,7,27,85)$ and 19 $(3,8,27,92)$ graphs are presented.

(3,8,27,85) – Ramsey Graphs, 1 and 2 of 4

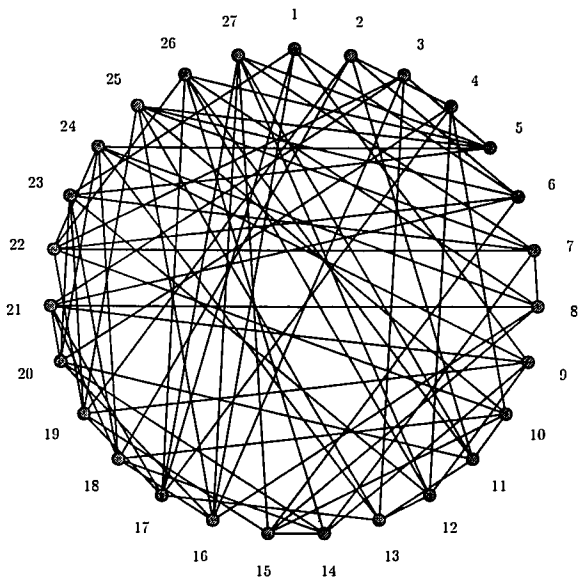


[AZ@B@QH@``P@Q`MAqR`Bp`@`JH`dQBAbHHAQIAtV@lAaaHApQhAMFH@VRD`

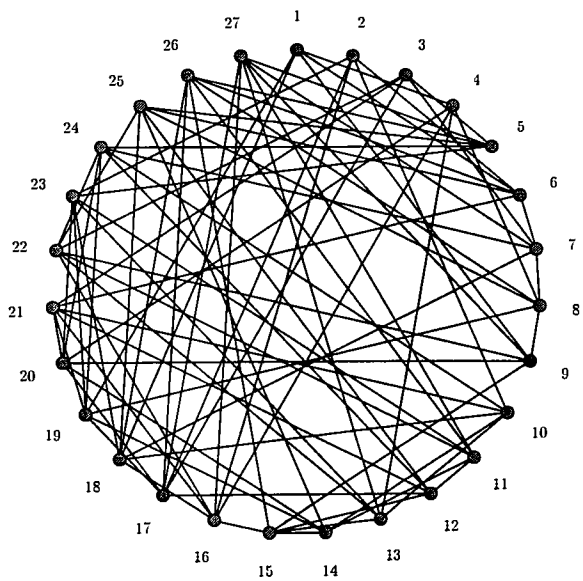


[Aa`@``PPa@P@EaCA\R`Ap`@`QQ@TPaDRA`PRQAtL`fAaadAbaHBdPxALc@P

(3,8,27,85) – Ramsey Graphs, 3 and 4 of 4

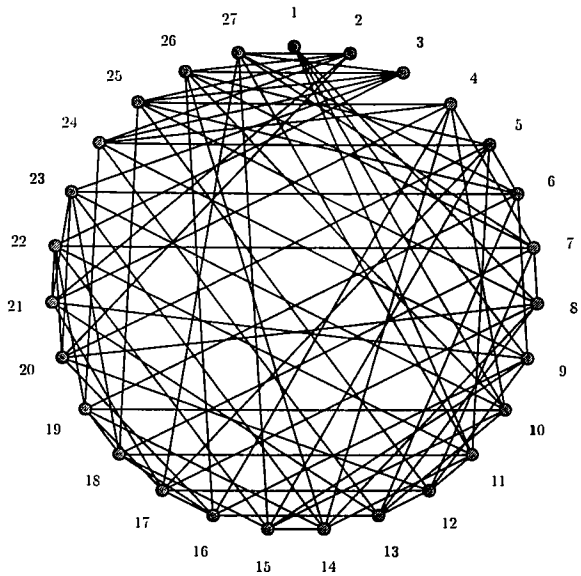


[BK@@F@DB@bDHC@h@jMB@d@a@QRAAP`R`e`MId@dTHNALPjCdd`BSRDAQg@@

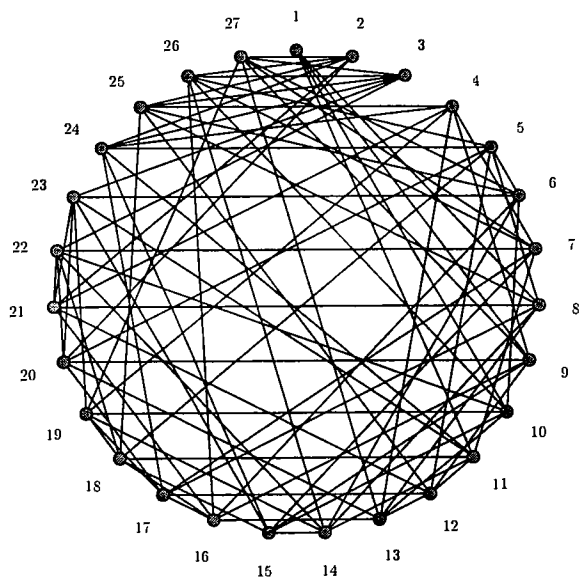


[DKAHfAP@@@ADD@F@Dz`AXABBPTBBPaBPdTSHJdBPTNAR`ZCQQDCRK@AUS@@

(3,8,27,92) – Ramsey Graphs, 1 and 2 of 18

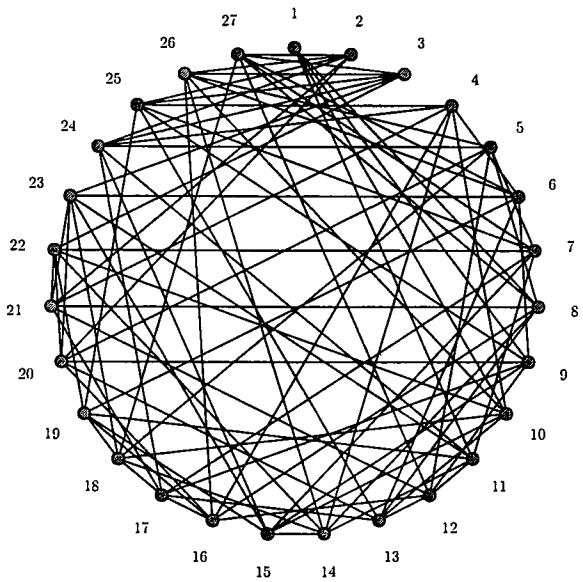


[@@HdFDdDP@UAUDRAFHaDBQDBHpHbZRHaPbHHbHyHbKODPP]QA@ZdT@LqE@@

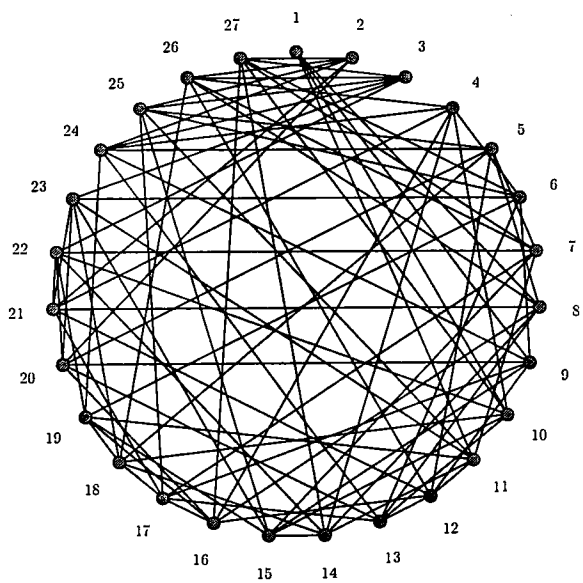


[@@HdRPaDDAEDUAR@VHID@qDHHp`bZQDQQDPHdQYHQGOAE@]QA@ZdT@LtPP@

(3,8,27,92) – Ramsey Graphs, 3 and 4 of 18

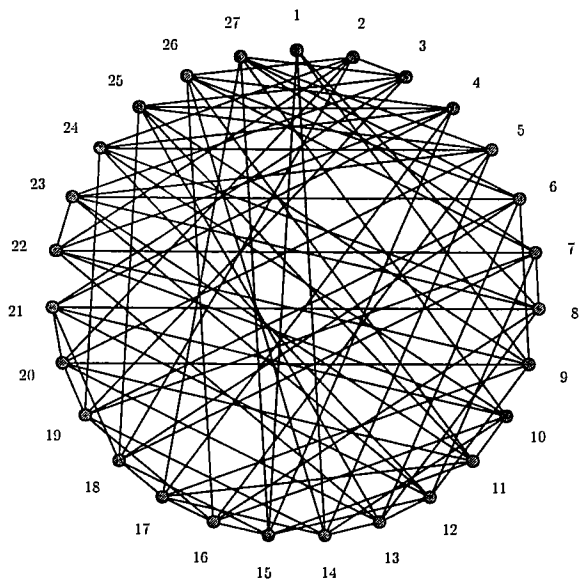


[@@PTRDdDDAEAUDR@VHbH@hdHQHHQjQDaQDHHdPyHQKOAE@\h``[DT@Lx``@

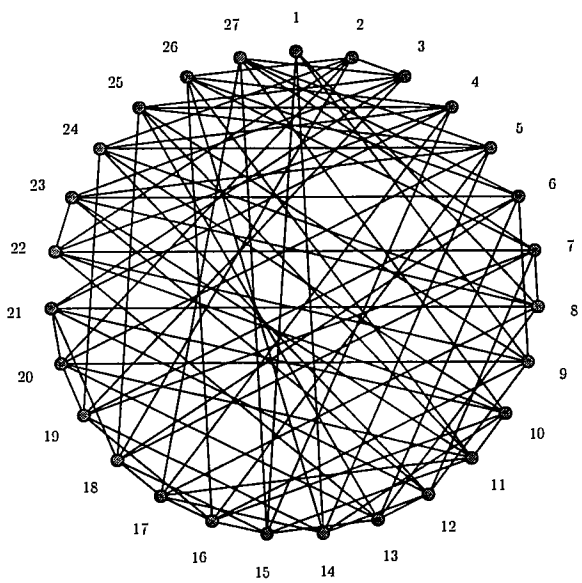


[@@PTRDdDDAEAUDR@VHbH@hdHQHHQjQDaQDHHdPyHQKODPP[BJ@\qA@LrJ@@

(3,8,27,92) – Ramsey Graphs, 5 and 6 of 18

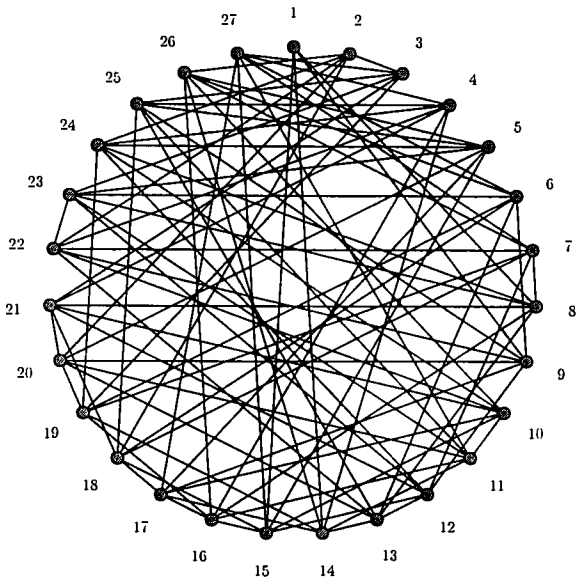


[H@@DFD@@@`TAQbDrA`RaDJUF@iLBbQTDqTETl`AZX@kJ`PNJa@UbT@ErE@@

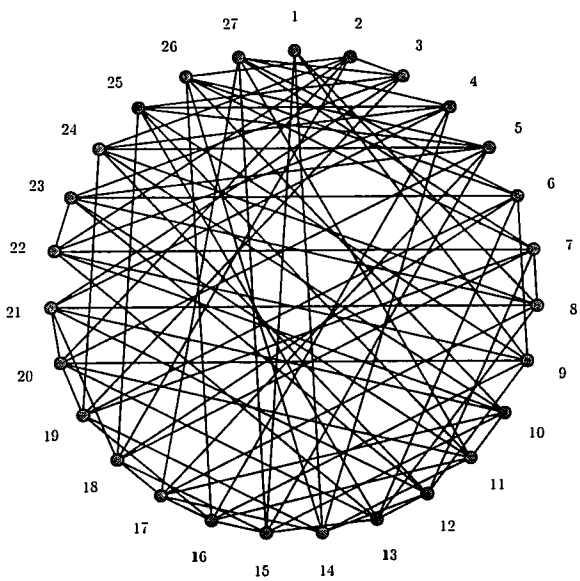


[H@@DFD@@@`TAQbERAPQaDRUF@iLBbQTDqTETl`AZX@kJ`PNJa@UbT@ErE@@

(3,8,27,92) – Ramsey Graphs, 7 and 8 of 18

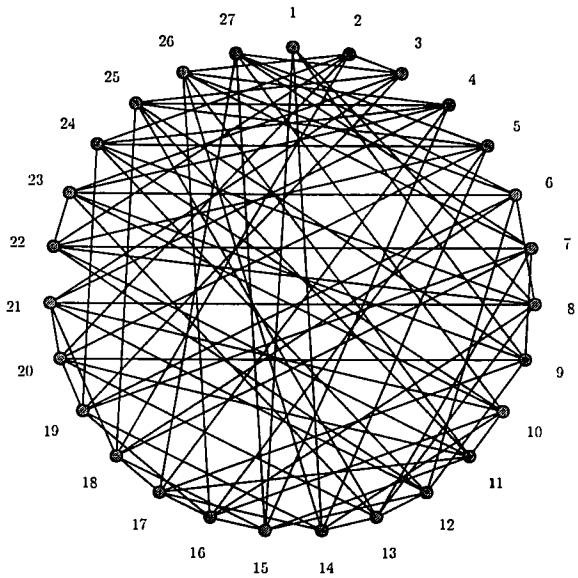


[H@@DFD@@@`TAQdBqB`bQBLUF@iLBbQTDqTETl`AZX@kJ`PNJa@SdT@FqE@@

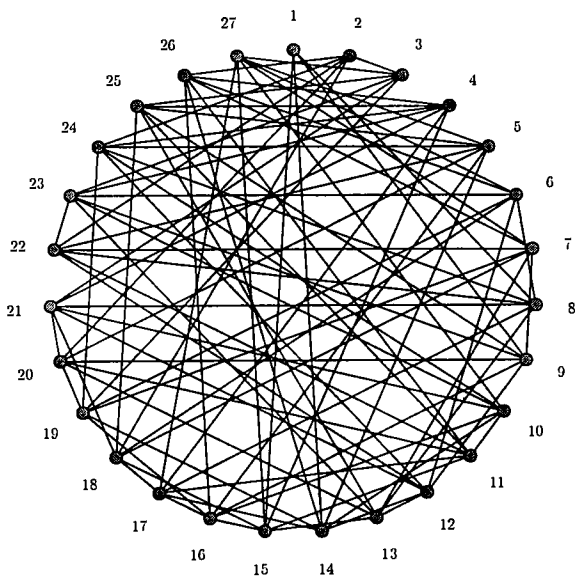


[H@@DFD@@@`TAQdCQBPaQBTUF@iLBbQTDqTETl`AZX@kJ`PNJa@SdT@FqE@@

(3,8,27,92) – Ramsey Graphs, 9 and 10 of 18

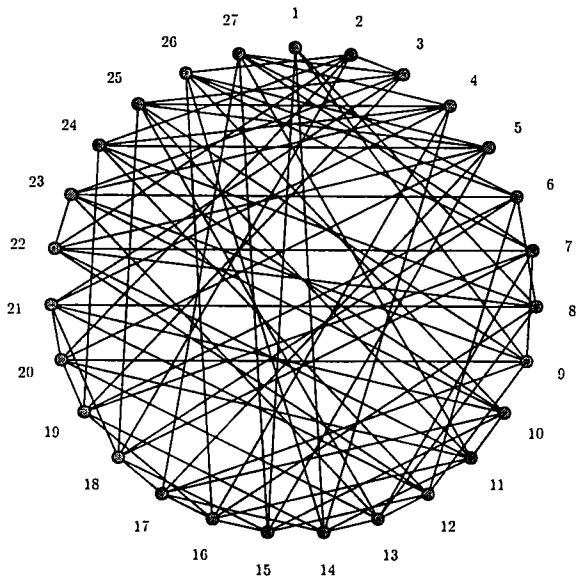


[H@@DJB@@@`d@qbDrA`RaDJUF@iLBbQTDqTERs@Aid@kJ`PNJa@UbT@ErE@@

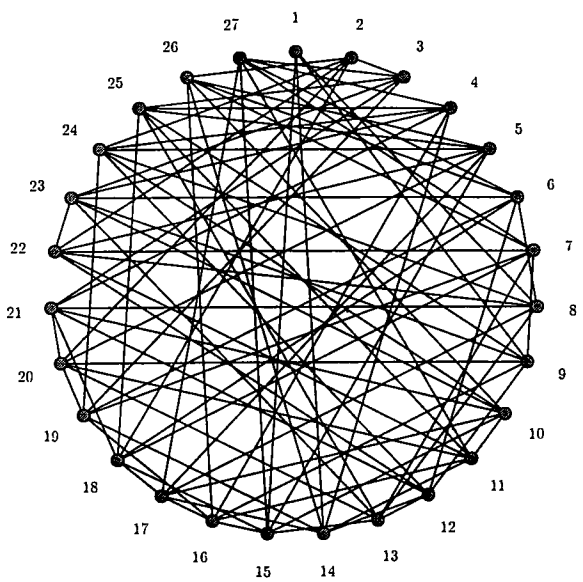


[H@@DJB@@@`d@qbERAPQaDRUF@iLBbQTDqTERs@Aid@kJ`PNJa@UbT@ErE@@

(3,8,27,92) – Ramsey Graphs, 11 and 12 of 18

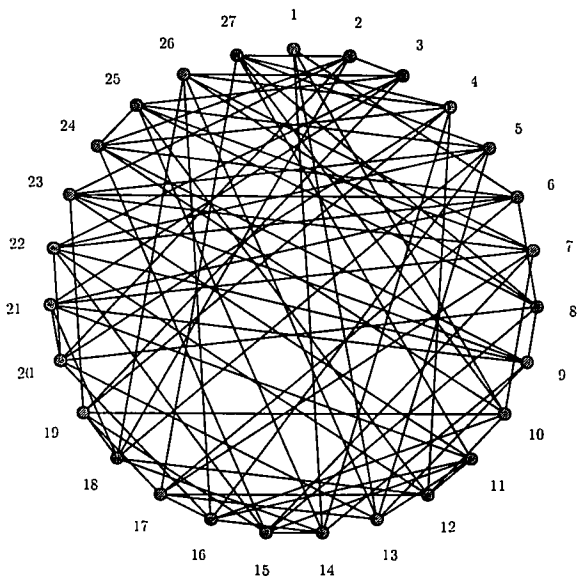


{H@@DJB@@@`d@qdBqB`bQBLUF@iLBbQTDqTERs@Aid@kJ`PNJa@SdT@FqE@@

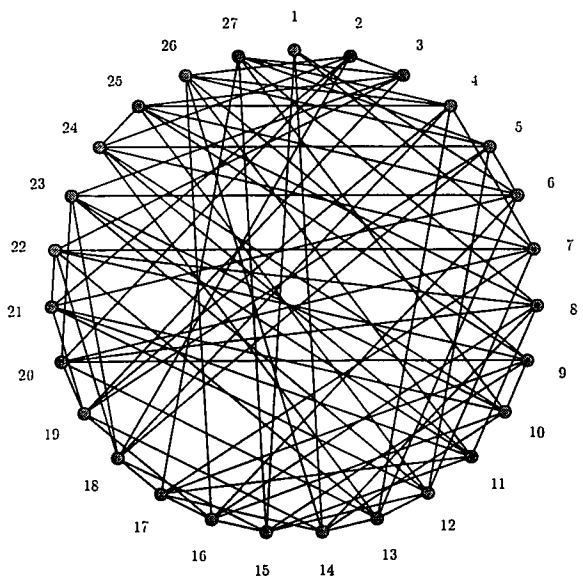


{H@@DJB@@@`d@qdCQBPaQBTUF@iLBbQTDqTERs@Aid@kJ`PNJa@SdT@FqE@@

(3,8,27,92) – Ramsey Graphs, 13 and 14 of 18

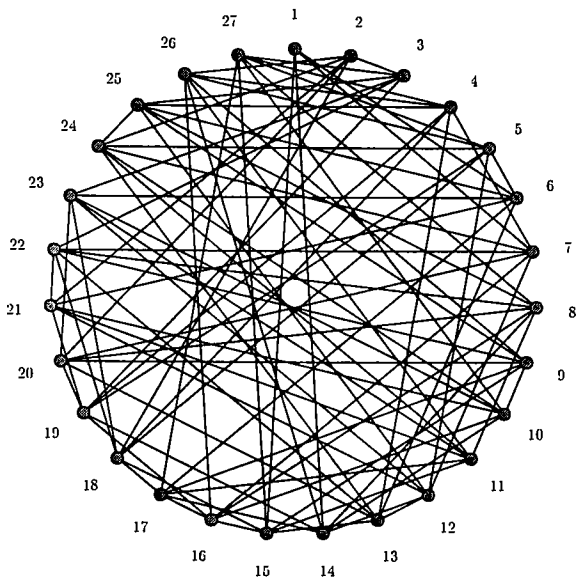


[HB@@HA``@BCbJdH@YHPrAaeHEA`aZbD`fhEKIHR]PDJfD@JqPALpe@KKA@@

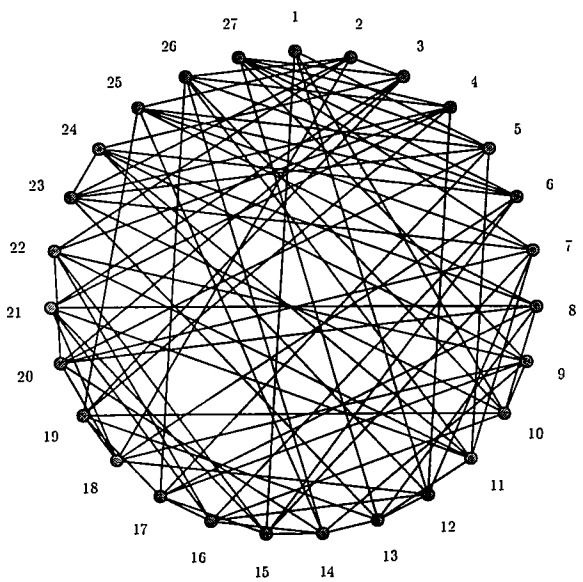


[HBL`@@@HBAPEA`TpE`Ja@jUX@ipBbGDED\BHxaIHxJISP@MY@AVJT@GJE@@

(3,8,27,92) – Ramsey Graphs, 15 and 16 of 18

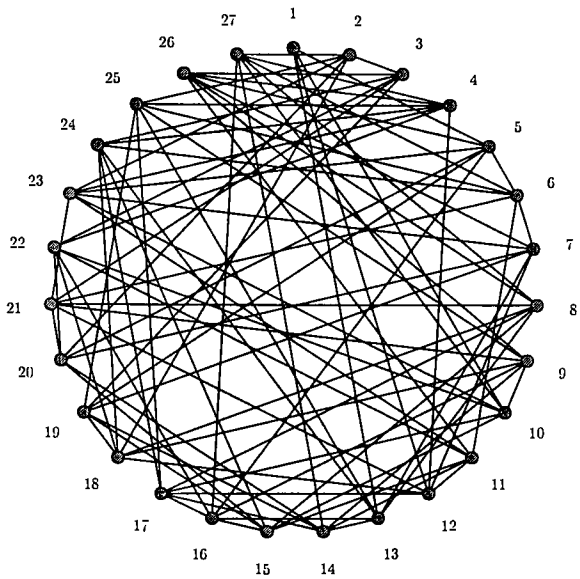


[HBL`@@@HBAPEA`UPEPFaAJUX@ipBbGDED\BHxaIHxJISP@MY@AVJT@GJE@@

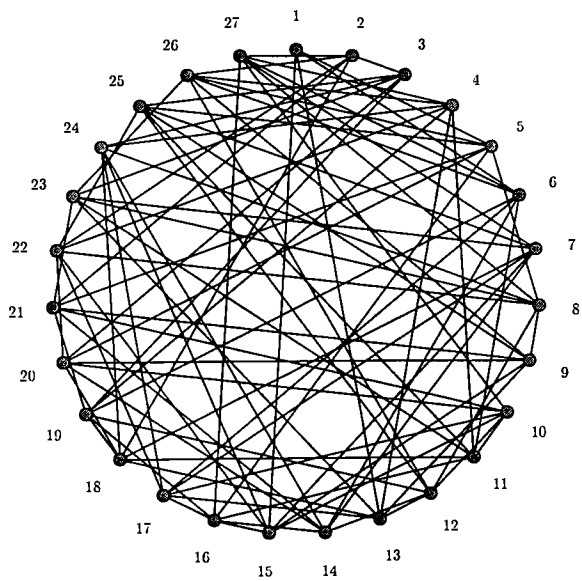


[H`@@@@`XTRP@S@HppHHjBXMDdA`dYVBHqHtTJLRtd@Dm`ASpP`UFR@GtP@@

(3,8,27,92) – Ramsey Graphs, 17 and 18 of 18



[H`HH@@`HHRQ`p@J`K@QE@UUAdATCJTC@edALFERtpAF`U`UHb`Nz@@JjJ@@



[Hb@@PA@@h`JdIAaPV@PSAdeBJAhHiQbIHqBTPJQFTAFHM`Jk@DWI@BJzB@@

6.4. (3,9) Ramsey Graphs

6.4.1. (3,9) Ramsey Graph Summary

This section contains a summary of the non−isomorphic $(3,9)$ Ramsey graphs enumerated by the author.

All of these graphs were enumerated using the EXPAND algorithm as originally specified in [GRA68a] and further detailed in [GRI82a], [RAD88a], and [RAD88b]. The theoretical framework for this enumeration algorithm is specified in Section §3.1. Pseudo code documenting this algorithm is provided in Section §4.3.4. and its implementation is described in Section §4.4.5.

The catalogue of $(3,8)$ graphs, which is documented in the previous section, forms the pool of $H_1(v)$ input graphs utilized for the EXPAND algorithm.

All $(3,9,n)$−graphs enumerated are believed to be newly enumerated graphs (i.e. not enumerated during any other investigations). The lower bounds for $e(3,9,n)$ with $n = 26,27,28,29,30$, and 31 were established in [RAD88b] as having values of $52,59,67,75,84$, and 93. At the current time the calculations performed by the author have resulted in a single $(3,9,26,52)$−graph therefore substantiating the lower bound for this parameter situation.

Table 6.4.1. provides a summary of all $(3,9)$−graphs enumerated. Note that the values for the graph order in the left x−axis are discontiguous. Graph counts enumerated by the author which were previously unknown are prefixed with a † character.

As previously mentioned the lower bound for various values of $e(3,9,n)$ has been investigated. [RAD88b] Since the exact value of $e(3,9,n)$ has not been established for various cases a dash character appears in the table at various points for which graphs may yet be calculated.

size	*order n*						*total*
e	26	27	28	29	30	31	
52	1						1
53	1						1
54	443						443
55	58103						58103
59							
60							
61		696					696
62		77129					77129
67							
68			94				94
69			8798				8798
70			227859				227859
75							
76							
77				559			559
78				25129			25129
84							
85							
86					591		591
87					3023		3023
93							
94							
95						160	160
96						4416	4416
total	58548	77825	236751	25688	3614	4576	407002

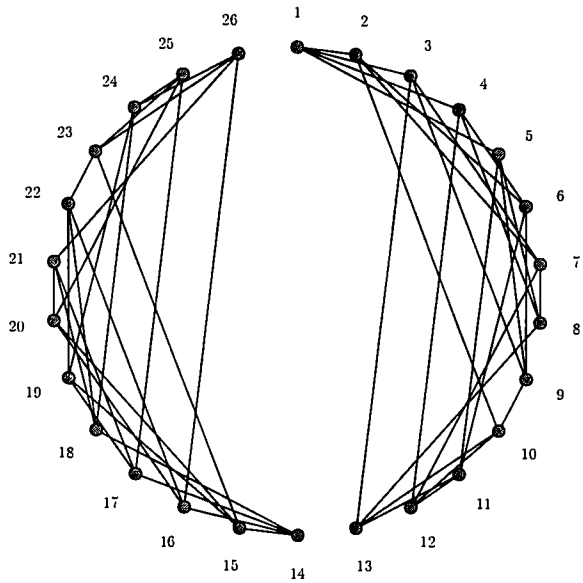
Table 6.4.1.1. – Number of $(3, 9, n, e)$–graphs, g_{39}.

6.4.2. $(3, 9, 26, 52)$ Ramsey Graph

The lower bound of the value $e(3, 9, 26)$ was established by Radziszowski and Kreher in [RAD88b]. This lower bound was established via use of the theorems provided in Chapter 2 as well as some simple Ramsey graph degree sequence evaluations. The calculation of this single graph serves to provide evidence for this lower bound and establishes the exact value for $e(3, 9, 26)$.

This section contains pictorial representations of the single $(3, 9, 26, 52)$–graph which was enumerated.

(3,9,26,52) – Ramsey Graph, 1 of 1



ZtaS@tlPHXBSHV@@@@H@B@@P@A@@AP@A`@@M@@E'@A@`@@p@@BS@@DK@

Chapter 7. Appendix A - grama Test Suite Example

This appendix contains the source code listing for the first grama demonstration program described in §4.4.6. The program listing provided is for the C programming language version of the program. The same exact program is also implemented in the Perl scripting language. The Perl version utilizes the SWIG package as a means for *hooking* into the grama Package C libraries. The Perl version of this program, although not provided here, is available.

ydemo0

```
/*
 * author:    Robert A. Getschmann
 * module:    ydemo0.c
 * program:   ydemo0
 *
 * usage:     ydemo0 [ -o <file> ]
 *
 * synopsis:  ydemo0 generates a graph with 6 nodes.  The adjacency matrix for
 *            the graph as well as the incidence matrix are then output.  The
 *            distance matrix, column characteristic matrix, row characteristic
 *            matrix, characteristic matrix, class vector and the class count
 *            vector are then generated and output.
 *
 *            If the series variable is set to 0 the first test suite is run.
 *            If it is set to 1 the second test suite is run.  The first test
 *            suite uses the graph methods to dump the various matrices while
 *            the second suite uses the native object methods (those associated
 *            with the matrices, and vectors etc) to dump the various structures.
 *
 * version:   $Id: ydemo0.c,v 1.9 2000/03/17 20:14:46 robg Exp $
 */


#include <assert.h>
#include <grama.h>
#include <stdio.h>
#include <string.h>
#include <types.h>

#define NP(x) ((x*)0)

extern char* optarg;
static char rcsId[] = "$Id: ydemo0.c,v 1.9 2000/03/17 20:14:46 robg Exp $";
static char* program;

int main(int argc, char** argv)
{
        FILE* fp;
        Graph *a, *b;
        Sequence* c;
        Vertex v1;
        char* file = NP(char);
        int o;

        /* Parse the program name as well as any command line options */
```

```
if (program = strrchr(*argv, '/')) ++program; else program = *argv;
while ((o = getopt(argc, argv, "o:s:")) != EOF) {
        switch(o) {
                case 'o': if (!(file = strdup(optarg))) exit(1); break;
                default: exit(1); break;
        }
}

/* Determine the destination of the output of the program */
if (fp = stdout, file && !(fp = fopen(file, "w"))) exit(1);
setvbuf(fp, 0, _IONBF, 0);

/* Graph 1 */
a = graphNew(6); assert(a != NP(Graph));
graphConnect(a, 1,2);
graphConnect(a, 1,4);
graphConnect(a, 1,5);
graphConnect(a, 1,6);
graphConnect(a, 2,3);
graphConnect(a, 2,4);
graphConnect(a, 2,5);
graphConnect(a, 3,4);
graphConnect(a, 3,6);
graphConnect(a, 4,5);
graphConnect(a, 5,6);

/* Graph 2 */
b = graphNew(10); assert(b != NP(Graph));
graphConnect(b, 1,5);
graphConnect(b, 1,9);
graphConnect(b, 2,3);
graphConnect(b, 2,5);
graphConnect(b, 2,6);
graphConnect(b, 2,8);
graphConnect(b, 2,9);
graphConnect(b, 4,10);
graphConnect(b, 8,10);

/* Adjacency Matrix Test   Graph 1 */
fprintf(fp, "Graph 1 Adjacency Matrix - Order: %d\n", graphOrder(a));
graphDisplayAdjMatrix(a, fp, NP(DisplayOption));
fputc('\n', fp);

/* Adjacency List Test - Graph 1 */

{
        int i, j;
```

```
        fprintf(fp, "Graph 1 Adjacency List - Order: %d\n",
                graphOrder(a));
        for (i = 1; i <= graphOrder(a); ++i) {
                fprintf(fp, "%d:", i);
                for (j = i; j <= graphOrder(a); ++j) {
                        if (graphAdjacent(a, i,j) == SUCCESS)
                                fprintf(fp, " %d", j);
                }
                fputc('\n', fp);
        }
        fputc('\n', fp);
}

/* Incidence Matrix Test - Graph 1 */
fprintf(fp, "Graph 1 Incidence Matrix - Order: %d, Size: %d\n",
        graphOrder(a), graphSize(a));
graphDisplayIncMatrix(a, fp, NP(DisplayOption));
fputc('\n', fp);

/* Incidence List Test - Graph 1 */

{
        int i, j;

        fprintf(fp, "Graph 1 Incidence List - Order: %d, Size: %d\n",
                graphOrder(a), graphSize(a));
        for (i = 1; i <= graphOrder(a); ++i) {
                fprintf(fp, "%d:", i);
                for (j = i; j <= graphOrder(a); ++j) {
                        if (incMatrixAdjacent(a->im, i,j) == SUCCESS)
                                fprintf(fp, " %d", j);
                }
                fputc('\n', fp);
        }
        fputc('\n', fp);
}

/* Distance Matrix Test  Graph 1 */
fprintf(fp, "Graph 1 Distance Matrix - Order: %d\n", graphOrder(a));
graphDisplayDistMatrix(a, fp, NP(DisplayOption));
fputc('\n', fp);

/* Column Characteristic Matrix Test  Graph 1 */
fprintf(fp, "Graph 1 Column Characteristic Matrix - Order: %d\n", graphOrder(a));
graphDisplayColCharMatrix(a, fp, NP(DisplayOption));
fputc('\n', fp);

/* Row Characteristic Matrix Test  Graph 1 */
```

```
        fprintf(fp, "Graph 1 Row Characteristic Matrix - Order: %d\n", graphOrder(a));
        graphDisplayRowCharMatrix(a, fp, NP(DisplayOption));
        fputc('\n', fp);

        /* Characteristic Matrix Test   Graph 1 */
        fprintf(fp, "Graph 1 Characteristic Matrix - Order: %d\n", graphOrder(a));
        graphDisplayCharMatrix(a, fp, NP(DisplayOption));
        fputc('\n', fp);

        /* Class Vector Test   Graph 1 */
        fprintf(fp, "Graph 1 Class Vector - Size: 6\n");
        graphDisplayClassVector(a, fp, NP(DisplayOption));
        fputc('\n', fp);

        /* Class Count Vector Test   Graph 1 */
        fprintf(fp, "Graph 1 Class Count Vector - Size: 2\n");
        graphDisplayClassCountVector(a, fp, NP(DisplayOption));
        fputc('\n', fp);

        /* Adjacency Matrix Test   Graph 2 */
        fprintf(fp, "Graph 2 Adjacency Matrix - Order: %d\n", graphOrder(b));
        graphDisplayAdjMatrix(b, fp, NP(DisplayOption));
        fputc('\n', fp);

        /* Breadth First Search Test   Graph 2 */
        fprintf(fp, "Graph 2 Breadth First Search from Vertex %d\n", v1=1);
        c = graphBFS(b, v1, NP(Sequence));
        sequenceDisplay(c, fp, NP(DisplayOption));
        fputc('\n', fp);
        fprintf(fp, "Graph 2 Breadth First Search Tree from Vertex %d\n", v1);
        graphBFST(b, v1, fp);
        fputc('\n', fp);

        /* Connectivity Test - Graph 2 */
        fprintf(fp, "Graph 2 Connectivity Test\n");
        if (graphConnected(b)) fprintf(fp, "Connectivity Test Passed\n");
        else fprintf(fp, "Connectivity Test Failed\n");

        fclose(fp);
        exit(0); return (0);
}
```

Chapter 8. Appendix B - Glossary

The following terms are pertinent to Combinatorics, Graph Theory, and Ramsey Theory and are used throughout this document.

adjacent

Two vertices in any given graph G are considered *adjacent* if they are joined by an edge from the set $E(G)$.

adjacency matrix

The *adjacency matrix* for any given graph G, is a $p \times p$ matrix $A = [a_{ij}]$ where each row and column of A corresponds to a distinct vertex from from the set $V(G)$, and $a_{ij} = 1$ if vertex v_i is adjacent to vertex v_j in G and $a_{ij} = 0$ otherwise.

arc

An *arc* is a directed edge.

automorphism

An *automorphism* for any given graph G is an isomorphism from the graph G onto itself.

bicritical graph

A *bicritical graph*, G, is one which the addition of an edge increase $C(G)$, the clique number, and the deletion of any edge increase $I(G)$, the independence number, and neither G nor $\overline{G}$ is a complete graph.

bipartite graph

A *bipartite graph* is a given graph $G = (V, E)$ for which the set of vertices V can be partitioned into two sets X and Y such that every edge is between a vertex in X and a vertex in Y.

bridge (cut-edge)

A *bridge (cut-edge)* is a single edge from the set E for a given graph $G = (V, E)$ which comprises the disconnecting set F for the graph G.

characteristic matrix

The *characteristic matrix*, X, of a graph G is defined as the term-wise composition of the row characteristic matrix X_r and the column characteristic matrix X_c. This matrix can be used to partition vertices of a graph into different classes. Rows of the *characteristic matrix* which are identical are considered to be in the same *class.*

circuit

A *circuit* is a closed trail.

circumference

The *circumference*, $c(G)$, for a connected graph G is then length of the longest cycle.

combination

A *combination* of n things taken k at a time is a subset of k elements selected from a set U of n elements. The number of different combinations of n things taken k at a time is the number of different subset of k element contained in the set U. If $1 \leq k \leq n-1$, then the number of combinations of n things taken k at a time is given by the formula: $C(n,k) = P(n,k)/k! = n!/k!(n-k)!$

combinatorics

Combinatorics is the field of study concerned with the arrangement of 3 elements into sets.

clique number

The *clique number*, $C(G)$, for a graphs G is the size of the largest clique in G.

column characteristic matrix

The *column characteristic matrix*, $X_c = [c_{ij}]$, of a graph G is defined to be the $p \times (p-1)$ matrix where c_{ij} is the number of vertices at distance j from vertex v_i.

column class vector

The *class vector*, $C = [k_i]$, of a graph G is defined to be the vector for which the element k_i = the class of the vertex v_i.

complement

A *complement* graph, H, of a graph $G = (V, E)$, is a graph which has the same vertex set, $V(H) = V(G)$ as G, with the property that any two vertices which are adjacent in G are non-adjacent in H, and any two vertices which are non-adjacent in H are adjacent in G.

complete graph

A *complete graph* of order n, denoted K_n, is a graph in which an edge exists between each pair of vertices in the set $V(K_n)$.

critical graph

A *critical graph* for a Ramsey number $R(l,k)$ is a graph on $R(l,k)-1$ vertices which does not contain sub–graphs K_l or K_k.

cut vertex

A *cut vertex* for a graph connected $G = (V, E)$ is a vertex from the set V such that the graph $G(V - v, E)$ is disconnected.

cycle

A *cycle* is a closed path.

degree

The degree of a vertex v is the number of edges incident to v.

degree sequence

The *degree sequence* for any given graph G is a sequence containing in decreasing order, the degrees of all of the vertices in the set $V(G)$.

diameter

The *diameter*, $d(G)$ for a connected graph G is the length of any longest geodesic.

directed graph (digraph)

A *directed graph (digraph)* G, is a graph in which each edge has a head and a tail, and is directed from one vertex in the set $V(G)$ to another vertex in the set $V(G)$.

Dirichlet's Box Principle

Dirichlet's Box Principle states that if a sufficiently large set of 3 elements is partitioned into a small number of blocks then at least one block contains a rather large number of elements. Also known as the *Pigeon Hole Principle.*

distance

The *distance* $d(u, v)$ for any two points u and v of a given graph G is the length of the shortest path between the two points u and v if such a path exists, otherwise $d(u, v) = \infty$.

distance matrix

The *distance matrix*, D, for a given graph G of order n is an $n \times n$ matrix for which each element (i, j) contains the distance of the shortest path between the two vertices i and j.

dominating edge set

A *dominating edge set denoted by* F, is a subset of the vertices of a graph, such that every edge not in F has a vertex in common with an edge in F.

dominating vertex set

A *dominating vertex set denoted by* D, and also known as an external dominating set *is a subset of the vertices of a graph, such that every vertex not in* D *is adjacent to at least one vertex in* D.

edge

An *edge* is a basic component of a graph. The collection of all edges for a given graph G forms the set E. Each member of the set E is a pair (u, v) where u and v are members of the set V.

Eulerian circuit

A *Eulerian circuit* for any given graph G, is a circuit which contains all of the edges of the set $E(G)$. Such a graph is referred to as a *Eulerian graph.*

full vertex

A vertex v of degree d in a (k, l, n) graph is defined to be *full* if $||H_2(v)|| = e(k, l-1, n-d-1)$, where $H_2(v)$ is the sub−graph induced by the vertices non-adjacent to v.

girth

The *girth*, $g(G)$, for a connected graph G is the length of the shortest cycle.

geodesic

The *geodesic* for a connected graph G is the shortest $u-v$ path.

graph

A *graph* $G = (V, E)$ is a finite non-empty set $V(G)$ of elements called vertices, together with a set $E(G)$ of two element subsets of the set $V(G)$ called edges.

graphical vector

A *graphical vector* is a vector v such that v is a valid degree sequence for some graph G.

Hamiltonian cycle (path)

A *Hamiltonian cycle (path)* for a graph G is a cycle (path) that contains every vertex of the set $V(G)$.

hyper-graph

A *hyper-graph* $H = (V, E)$ is a graph for which there exists a family of subsets of the set V. Each member of this family of subsets of V is called a *face.*

in-degree

The *in-degree* for a given vertex v of the set V from the digraph G is number of arcs which are directed towards the vertex v.

incidence matrix

An *incidence matrix* for any given graph $G(V, E)$ with $V = (1, 2, \ldots, m)$ and $E = (e_{1,e\_2,\ldots e\_n})$ is a $m \times n$ matrix $B = [b_{ij}]$ such that row i of B corresponds to vertex i for each i. Column k corresponds to edge e_k for each k. If a given edge e_k joins vertex i and vertex j entries b_{ik} and b_{jk} are 1 and 0 otherwise.

incident

Two vertices u and v are *incident* in a given graph G if there exists an edge in E which joins the two vertices.

independence number

The *independence number*, $I(G)$, for a graph G is the size of the largest independent set in G.

independent (non-adjacent)

Two vertices are considered *independent (non-adjacent)* if they are not joined by an edge from the set $E(G)$.

independent set

An *independent set* of order n for any given graph G is a group of vertices from the set $V(G)$ which form a complete sub−graph $\bar{K}_n$.

induced sub−graph

An *induced sub−graph* is a graph $H = (W, F)$ induced from any given graph $G = (V, E)$ such that an edge exists in F between two vertices in W if and only if and edge exists in E between those two vertices.

isolated edge

An *isolated edge* denoted *E(v1,v2)*, where *v1* and *v2* are the edge endpoints, is an edge for which the degree of *v1* and *v2* is 1.

isolated vertex

An *isolated vertex* is a vertex which has a degree of 0.

isomorphism

Two graphs G and H are considered *isomorphic* if and only if there exists a one to one and onto mapping, $f: V(G) \rightarrow V(H)$, such that for any edge xy in the set $E(G)$, the edge $f(x)f(y)$ is a member of the set $E(H)$, thereby preserving adjacency and non-adjacency. The function f is called an *isomorphism.*

loop

A *loop* is an edge represented by an unordered pair of endpoints in which the two elements are not distinct.

mixed graph

A *mixed graph* is a given graph G which contains at least on arc and at least on edge.

matching

A *matching* also known as an independent edge set is of edges M of a graph for which no two edges in M have a vertex in common.

multigraph

A *multigraph* G is a graph in which multiple edges between any two given vertices from the set $V(G)$ are allowed.

n-partite graph

An *n-partite graph* is a graph G in which the set of vertices V of G can be partitioned into n sets such that any edge of G joins two vertices in different *partite* sets.

order

The *order* for a graph G is the cardinality of the set $V(G)$.

orientation

An *orientation* of a simple graph G is obtained if every edge of G is replaced by an arc.

out-degree

The *out-degree* for a given vertex v of the set V from the digraph G is number of arcs which are directed out of the vertex v.

partition

A *partition* for a set S is any collection of disjoint subsets of S whose union is S.

path

A *path* for any given graph G is an $x-y$ walk in which no vertex from the set $V(G)$ is repeated.

permutation

A *permutation* is an ordered $k-$tuple $P = (a_1, a_2, \cdots, a_k)$ in which no two of the a_i's in P are the same. P is a permutation of n things taken k at at time if the a_i's are from U, a set of n elements. The following formula gives the number of n things taken k at a time. $P(n,k) = n!/(n-k)!$

Pigeon Hole Principle

The *Pigeon Hole Principle* states that if a sufficiently large set of elements is partitioned into a small number of blocks then at least one block contains a rather large number of elements. Also known as the *Dirichlet Drawer* or the *Dirichlet Bow Principle.*

preferred vertex

For a (k,l) graph G, if the vertices are partitioned into three sub$-$graphs, the vertex v, the sub$-$graph $H_1(v)$ generated by the vertices adjacent to v, and the sub$-$graph $H_2(G)$ generated by the vertices non-adjacent to v, the vertex v is *preferred*

pseudo$-$graph

A pseudo$-$graph G is a graph in which edges from a vertex v of the set $V(G)$ to the same vertex v are allowed.

r-regular graph

A *r-regular* graph G is a graph in which every vertex in the set $V(G)$ has degree r.

Ramsey graph

A *Ramsey graph* (k, l, n, e) is a $k-$clique free, $l-$independent set free graph on n vertices with e edges.

Ramsey number

A *Ramsey number* is the smallest number n such that any graph of order n contains a clique of size k or an independent set of size l.

row characteristic matrix

The *row characteristic matrix* $X_r = [r_{ij}]$, of a graph G is defined to be the $p \times (p-1)$ matrix where r_{ij} is the number of vertices at distance j from vertex v_i.

rule of product

The *rule of product* asserts that if T_1 is an n_1-set and if $M_2 = M(T_1, T_2, n_2)$, $M_3 = M(M_2, T_3, n_3)$, and finally if $M_r = M(M_{r-1}, T_r, n_r)$, then M_r ia an $(n_1 n_2 \cdots n_r)-$set.

rule of sum

The *rule of sum* asserts that if T_i is an n_i-set $(i = 1, 2, \ldots, r)$ and if $M = T_1 \bigcup T_2 \bigcup \cdots \bigcup T_r$ is a partition of M, then M is an $(n_1 + n_2 + \cdots + n_r)-$set.

sample

A *sample* is an $r-$tuple of not necessarily distinct elements of a give set S. The number of $r-$samples of an $n-$set is n^r.

set

A *set* is a collection of objects which has the property that given any object, either inclusion or exclusion of the object in the set can be determined. Let S be an arbitrary set of elements $a, b, c, \cdots$ The fact that an element s is an element of the set S is indicated by writing $s \mathit{member} S$. If each element of a set A is an element of the set S, then A is a subset of S and is denoted by $A \subseteq S$. If $A \subseteq S$ and $S \subseteq A$ the two sets are identical and this is denoted by $A = S$.

simplex

An $n-$dimensional *simplex* in a Euclidean space consists of $n+1$ linearly independent points $p_0, p_1, \cdots p_n$ together with all line segments $a_0 p_0 + a_1 p_1 + \cdots + a_n p_n$ where the $a_i \geq 0$ and $a_0 + a_1 + \cdots + a_n = 1$. A triangle with its interior and a tetrahedron with its interior are examples.

size

The *size* of a graph G is the cardinality of the set $E(G)$.

spanning sub−graph

A *spanning sub−graph* H for any given graph G is a sub−graph in which $V(H) = V(G)$.

strongly connected

A digraph G is *strongly connected* if there is a directed path from each vertex to every other vertex.

sub−graph

A *sub−graph* H for any given graph G is a graph for which $V(H) \epsilon V(G)$ and $E(H) \epsilon E(G)$.

subset

A set A is considered a subset of a given set B if for each $x \in A$, $x \in B$ also holds. The notation $A \subseteq B$ indicates that A is a subset of B.

tree

A *tree* T is a connected acyclic graph.

trail

A *trail* on any given graph G is a walk in which no edge from the set $E(G)$ is repeated.

underlying graph

An *underlying graph* of a digraph G is obtained if every arc of the digraph G is changed to an edge.

vertex

A *vertex* is a component of a graph. The collection of all vertices for a given graph G form the set V.

vertex classification

A *vertex classification* is a technique to partitions the vertices of a given graph according to some graph property that is invariant under isomorphism.

walk

A $x-y$ walk in any given graph G is s finite alternating sequence of vertices from the set $V(G)$ that begins with the vertex u and ends with the vertex v and in which each edge from the set $E(G)$ in the sequence joins a vertex that precedes it in the sequence to the vertex that follows it in the sequence.

weakly connected

A digraph G is *weakly connected* if its underlying graph is connected.

Z-sum

The $Z - sum$ for a vertex v in a (k, l) graph G is the sum of the degrees of the neighbors of v.

Chapter 9. References

Alo94a.
ALON, N., "Explicit Ramsey Graphs and Orthonormal Labelings," *The Electronic Journal of Combinatorics (http://www.combinatorics.org/)*, R12, 1, p. 8 pages (1994).

Arb81a.
ARBIB, M. A., A. J. KFOURY, AND R. N. MOLL, *A Basis for Theoretical Computer Science,* The AKM Series in Theoretical Computer Science, Springer-Verlag, New York, NY (1981).

Bal97a.
BALAKRISHNAN, V. K., *Graph Theory,* The McGraw-Hill Companies, INC., New York, NY (1997).

Bal96a.
BALAKRISHNAN, V. K., *Shaum's Outline of Graph Theory: Including Hundreds of Solved Problems,* The McGraw-Hill Companies, INC., New York, NY (1996).

Bal94a.
BALAKRISHNAN, V. K., *Shaum's Outline of Combinatorics,* The McGraw-Hill Companies, INC., New York, NY (1994).

Bea97a.
BEAZLEY, D. M., *SWIG Reference Manual,* Department of Computer Science, University of Utah (1997).

Cha92a.
CHAR, B. W., K. O. GEDDES, G. H. GONNET, B. L. LEONG, M. MONAGAN, AND S. WATT, *First Leaves: A Tutorial Introduction to Maple V,* Springer-Verlag, Toronto, Ontario (1992).

Cha77a.
CHARTRAND, G., *Introductory Graph Theory,* Dover Publications, INC., New York, NY (1977).

Chu83a.
CHUNG, F. R. K. AND C. M. GRINSTEAD, "A Survey of Bounds for Classical Ramsey Numbers," *Journal of Graph Theory,* 7, pp. 25-37 (1983).

Chu93a.
CHUNG, F.R.K, R. CLEVE, AND P DAGUM, "A Note on Constructive Lower Bounds for the Ramsey Numbers $R(3,t)$," *Journal of Combinatorial Theory.* Series B, 57, pp. 150-155 (1993).

Coc95a.
COCKCROFT, A., *Sun Performance and Tuning,* Prentice-Hall, INC., Mountain View, CA (1995).

Erd35a.
ERDOS, P. AND G. SZEKERES, "On a Combinatorial Problem in Geometry," *Compositio Math,* 2, pp. 463-470 (1935).

Erd47a.
ERDOS, P., "Some Remarks on the Theory of Graphs," *American Mathematical Society,* 53, pp. 292-294 (1947).

Gar78a.
GARDNER, M., "Mathematical Games," *Scientific American,* 238, 3, p. 29 (1978).

Gib88a.
GIBALDI, J. AND W. S. ACHERT, *MLA Handbook for Writers of Research Papers,* The Modern Language Association of America, New York, NY (1988).

Goo75a.
GOODMAN, A. W. AND J. S. RATTI, *Finite Mathematics with Applications,* Macmillan Publishing Co., Inc., New York, NY (1975).

Gou88a.
GOULD, R., *Graph Theory,* The Benjamin/Cummings Publishing Company, INC., Menlo Park, CA (1988).

Gra83a.
GRAHAM, R. L., "Rudiments of Ramsey Theory," *Conference Board of the Mathematical Sciences Regional Conference,* Series in Mathematics, 45, pp. 1-65 (1983).

Gra68a.
GRAVER, J. E. AND J. YACKEL, "Some Graph Theoretic Results Associated with Ramsey's Theorem," *Journal of Combinatorial Theory,* 4, pp. 125-175 (1968).

Gre55a.
GREENWOOD, R. E. AND A. M. GLEASON, "Combinatorial Relations and Chromatic Graphs," *Canadian Journal of Mathematics,* 7, pp. 1-7 (1955).

Gri82a.
GRINSTEAD, C. M. AND S. ROBERTS, "On the Ramsey Numbers R(3,8) and R(3,9)," *Journal of Combinatorial Theory.* Series B. *Graph Theory and Matroid Theory,* 33, pp. 27-51 (1982).

Har94a.
HARAY, F., *Graph Theory,* Addison-Wesley Publishing Company, Reading, MA (1994).

Hea98a.
HEAL, K. M., M. L. HANSEN, AND K. M. RICKARD, *Maple V Learning Guide,* Springer-Verlag, New York, NY (1998).

Kal67a.
KALBFLEISH, J. G., "Upper Bounds for Some Ramsey Numbers," *Journal of Combinatorial Theory,* 2, pp. 35-42 (1967).

Kal66a.
KALBFLEISH, J. G., "Chromatic Graphs and Ramsey's Theorem," *Ph.D. Thesis,* University of Waterloo (January 1966).

Kam98a.
KAMERICH, E., *A Guide to Maple,* Springer-Verlag, New York, NY (1998).

Ker78a.
KERNIGHAN, B. W. AND D. M. RITCHIE, *The C Programming Language 1st Edition,* Prentice-Hall, INC., Englewood Cliffs, NJ (1978).

Ker91a.
KERNIGHAN, B. W., "PIC A Graphics Language for Typesetting (Revised User Manual)," Bell Laboratories Computing Science Technical Report No. 116, Bell Laboratories (1991).

Ker88a.
KERNIGHAN, B. W. AND D. M. RITCHIE, *The C Programming Language 2nd Edition,* Prentice-Hall, INC., Englewood Cliffs, NJ (1988).

Ker82a.
KERNIGHAN, B. W., "A Typesetter Independent Troff," Bell Laboratories Computing Science Technical Report No. 97, Bell Laboratories (1982).

Ker77a.
KERNIGHAN, B. W. AND L. L. CHERRY, "A System for Typesetting Mathematics," *Comm. Assoc. Comp. Mach.,* 18, pp. 151-157, Bell Laboratories, Murray Hill, NJ (1977).

Ker64a.
KÉRY, G., "On a Theorem of Ramsey (in Hungarian)," *Matematikai Lapok,* 15, pp. 204-224 (1964).

Kim95a.
KIM, J. H., "The Ramsey Number $R(3,t)$ has Order of Magnitude $t^2/\log t$," *Random Structures and Algorithms,* 7, pp. 173-207 (1995).

Les76a.
LESK, M. E., "TBL A Program to Format Tables," Bell Laboratories Computing Science Technical Report No. 49, Bell Laboratories (1976).

Les78a.
LESK, M. E., "Typing Documents on the UNIX System Using the -ms Macros with Troff and Nroff," *UNIX Programmer's Manual,* 2, Bell Laboratories (1978).

Les77a.
LESK, M. E., *Some Applications of Inverted Indexes on the UNIX System,* Bell Laboratories (1977).

Lin94a.
LINT, J. H. VAN AND R. M. WILSON, *A Course in Combinatorics,* Cambridge University Press, New York, NY (1994).

Mac96a.
MACKENZIE, D., *autoconf: Creating Automatic Configuration Scripts,* Free Software Foundation (1996).

McK96a.
MCKAY, B. D., "autoson, a Distributed Batch System for UNIX Workstation Networks (Version 1.3)," Technical Report TR-CS-96-03, Computer Science Department, Australian National University (1996).

McK90a.
MCKAY, B. D., "nauty User's Guide (Version 1.5)," Technical Report TR-CS-90-02, Computer Science Department, Australian National University (1990).

McK95a.
MCKAY, B. D. AND S. P. RADZISZOWSKI, "$R(4,5) = 25$," *Journal of Graph Theory,* 19, pp. 309-322 (1995).

Oss92a.
OSSANNA, J. F. AND B. W. KERNIGHAN, "Troff User's Manual," Bell Laboratories Computing Science Technical Report No. 54, Bell Laboratories (1992).

Par94a.
PARKER, S. P., (EDITOR), *McGraw-Hill Dictionary of Mathematics,* McGraw-Hill, New York, NY (1994).

Pet78a.
PETTOFREZZO, A. J., *Matrices and Transformations,* Dover Publications, INC., Toronto, Ontario (1978).

Piw93a.
PIWAKOWSKI, K., *Some Remarks on Classical Ramsey Numbers,* System Modeling Control 7 (Zakopane, Poland), 2, pp. 106-110, Technical University of Gdańsk (1993).

Piw94a.
PIWAKOWSKI, K., "Applying Tabu Search to Determine New Ramsey Graphs," *The Electronic Journal of Combinatorics (http://www.combinatorics.org/),* R6, 3 (1994).

Rad01a.
RADZISZOWSKI, S. P., "Small Ramsey Numbers," *The Electronic Journal of Combinatorics (http://www.combinatorics.org/),* R8 (2001).

Rad91a.
RADZISZOWSKI, S. P. AND D. L. KREHER, "Minimum Triangle-Free Graphs," *Ars Combinatoria,* 31, pp. 65-92 (1991).

Rad88a.
RADZISZOWSKI, S. P. AND D. L. KREHER, "On (3,k) Ramsey Graphs: Theoretical and Computational Results," *Journal of Combinatorial Mathematics and Combinatorial Computing,* pp. 37-52 (1988).

Rad88b.
RADZISZOWSKI, S. P. AND D. L. KREHER, "Search Algorithm for Ramsey Graphs by Union of Group Orbits," *Journal of Graph Theory,* 12, 1, pp. 59-72 (1988).

Rad88c.
RADZISZOWSKI, S. P. AND D. L. KREHER, "Upper Bounds for Some Ramsey Numbers R(3,k)," *Journal of Combinatorial Mathematics and Combinatorial Computing,* 4, pp. 207-212 (1988).

Ram30a.
RAMSEY, F. P., "On a Problem of Formal Logic," *Proceedings of the London Mathematical Society,* 2, 30, pp. 264-286 (1930).

Ray94a.
RAYMOND, E. S., *Making Pictures with GNU PIC,* Bell Laboratories (1994).

Rős88a.
ROSEN, K. H., *Discrete Mathematics and its Applications,* The Random House/Birkhauser Mathematics Series, Random House, Inc., New York, NY (1988).

Rys63a.
RYSER, H. J., *Combinatorial Mathematics,* The Carus Mathematical

Monographs, The Mathematical Association of America, Rahway, NJ (1963).

Ser91a.

SEROUL, R. AND S. LEVY, *A Beginner's Guide to TeX,* Springer-Verlag, New York, NY (1991).

Sta79a.

STATON, W., "Some Ramsey-Type Numbers and the Independence Ratio," *Transactions of the American Mathematical Society,* 256, pp. 353-370 (1979).

Ste92a.

STEVENS, W. R., *Advanced Programming in the UNIX Environment,* Addison-Wesley Publishing Company, New York, NY (1992).

Sto93a.

STOUSRUP, B., *The C++ Programming Language 2nd Edition,* Addison-Wesley Publishing Company, New York, NY (1993).

Tut84a.

TUTTE, W. T., *Graph Theory,* Addison-Wesley Publishing Company, INC., Menlo Park, CA (1984).

Wal68a.

WALKER, K., "Dichromatic Graphs and Ramsey Numbers," *Journal of Combinatorial Theory,* 5, pp. 238-243 (1968).

Win88a.

WINN, J. A., JR., *Asymptotic Bounds for Classical Ramsey Numbers,* Polygonal Publishing House, Washington, NJ (1988).