1997

Applying a genetic algorithm to improve the lower bounds of multi-color ramsey numbers

Shardul Rao

Applying a Genetic Algorithm to Improve the Lower Bounds of Multi-Color Ramsey Numbers

Shardul Rao
Computer Science Department
Rochester Institute of Technology

August 6, 1997

Rochester Institute of Technology
Department of Computer Science

Applying a Genetic Algorithm to Improve the Lower Bounds of Multi-Color Ramsey Numbers

Shardul Rao

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

Approved by:

Professor Stanislaw P. Radziszowski

Professor Peter G. Anderson

Professor Walter A. Wolf

August 6, 1997

Contents

Abstract

Ramsey Theory studies conditions when a combinatorial object contains necessarily some smaller given objects. The role of Ramsey Numbers is to quantify some of the general existential theorems in Ramsey Theory.

The objective of this thesis is to try to improve the lower bounds of Ramsey Numbers, in particular the bounds for multi-color graph Ramsey Numbers. Let $G_1, G_2, \ldots, G_m$ be graphs on some n. $R(G_1, G_2, \ldots, G_m)$ denotes the m-color *Ramsey number* for graphs, avoiding G_i in color i for $1 \leq i \leq m$. Thus, to show $R(G_1, G_2, \ldots, G_m) \geq N$, we need to find an edge coloring for a K_N graph using m colors (for N as large as possible) avoiding G_i in color i.

An order-based Genetic Algorithm (GA) combined with a greedy coloring heuristic is used as a search heuristic in finding the coloring. Each chromosome is a permutation representing the order in which the edges of graph are colored. (This approach was far more successful than a string representing the coloring.)

The algorithm was successful. The known lower bounds for $R(C_4, C_4, C_4)$, $R(C_4, C_4, K_3)$, $R(C_4, K_3, K_3)$, $R(C_5, C_5, C_5)$ were matched, and two new lower bounds for Ramsey numbers, $R(C_4, C_4, K_3, K_3) \geq 25$ and $R(C_5, C_4, K_3) \geq 13$, were found.

Acknowledgments

First of all, I would like to thank Dr. Stanislaw Radziszowski for introducing me to the wonderful theory of Ramsey numbers. His extensive research in this area was very useful for me in finding different sources of information.

I would also like to thank Dr. Peter Anderson, for encouraging me to solve this problem using Genetic Algorithms. His ideas of implementation were very helpful in writing the code. I also thank him for providing support when I had to take a break in my studies for a family emergency.

I also thank Dr. Walter Wolf, for accepting my request to be on my advisors committee and evaluating my thesis.

Last but not least, I would like to thank all my friends. Without their best wishes, it would have been difficult to complete this thesis.

1 Introduction

In a collection of six people either three of them mutually know each other or three of them mutually do not know each other.

This may be considered as a non-trivial example of Ramsey theory. This section introduces Ramsey theory and its relevance to graph theory.

1.1 Notations

Let $G = (V, E)$ be a graph with vertex set V and edge set E. The number n of vertices of G is called the *order* of G, and the number q of edges in G is called the *size* of G.

For $S \subseteq V$ we denote by $G[S]$ the graph with vertex set S and edge set of all $(u, v) \in E$ such that v and u belong to S. $G[S]$ is called *induced subgraph of* G. The *complement* of a graph G, denoted by G^c is the graph having same vertex set as G, but two vertices are adjacent in G^c if and only if they are not adjacent in G. A *complete* graph is a graph in which every two vertices are joined by an edge. K_N denotes the complete graph of order N.

A largest complete subgraph of a graph G is called a *clique* in G, and is denoted by $\beta(G)$. A subset of the vertex set of a graph G is called an independent set of vertices if the subgraph induced by it has no edges. The *independence number* of G is denoted by $\alpha(g)$ and is defined to be the cardinality of a largest independent set in G. Note that $\alpha(G) = \beta(G^c)$. The cardinality of a set S is denoted by $|S|$.

Following the notation in [GRS], we denote the set $\{1, 2, \ldots, n\}$ by $[n]$. We sometimes use $[n]$ to refer to an arbitrary set of cardinality n. We denote the set $\{Y : Y \subset S, |Y| = k\}$ by $[S]^k$, and when $S = [n]$ we write $[n]^k$. An r-coloring of a set S is a map

$$X : S \to [r].$$

For $s \in S$, $X(s)$ is called the color of s. A set $T \subset S$ is called a monochromatic (under X) if X is constant on T.

We write $n \to (s_1, \ldots, s_r)^k$ if, for every r-coloring of $[n]^k$, there exists i, $1 \leq i \leq r$, and a subset T of $[n]$, $|T| = s_i$, so that $[T]^k$ is colored i.

In the case $s_1 = s_2 = \ldots = s_r = s$ we use the shortened $n \to (s)^k_r$. If r is not indicated it is assumed to be 2. The Ramsey function $R_k(s_1, \ldots s_r)$ denotes the minimum n such that $n \to (s_1, \ldots, s_r)^k$.

1.2 Ramsey Theory

Ramsey theory, a deep generalization of pigeonhole principle, is due to Ramsey [1930]. To introduce the theory, assume that among 6 persons, each pair of persons are either friends or enemies. Then either there are 3 persons who are mutual friends or 3 persons who are mutual enemies.

Stated in other way, suppose that S is any set of 6 elements. If we divide the 2-element subsets of S into two classes, X and Y, then either

1. there is a 3-element subset of S all of whose 2-element subsets are in X, or

2. there is 3-element subset of S all of whose 2-element subsets are in Y.

Generalizing these conclusions, suppose that p and q are integers with $p, q \geq 2$, then a positive integer N has the (p, q) *Ramsey property* if the following holds: Given any set S of N elements, if we divide the 2-element subsets of S into two classes X and Y, then either

1. there is a p-element subset of S all of whose 2-element subsets are in X, or

2. there is q-element subset of S all of whose 2-element subsets are in Y.

Thus, the number 6 has the $(3, 3)$ Ramsey property. However, the number 5 does not have the $(3, 3)$ Ramsey property (as there is no p or q element subset satisfying above condition). Note that if the number N has the (p, q) Ramsey property and $M \geq N$, then the number M has the (p, q) Ramsey property.

Ramsey's Theorem states that, if p and q are integers with $p, q \geq 2$, then there is a positive integer N has the property (p, q) *Ramsey property.* The smallest integer N which has the (p, q) Ramsey property is called *Ramsey Number* and is denoted by $R(p, q)$.

The computation of the Ramsey numbers is in general a difficult problem. *Graph theory* makes it convenient to discuss Ramsey numbers. In studying Ramsey numbers $R(p, q)$, think of S as the vertex set of a graph, and of a set of 2-element subsets of S as the edge set of this graph. Then, to say that a number N has the (p, q) Ramsey property means that whenever S is a set

of N elements and we have a graph G with vertex set S, then if we divide the 2-element subsets of S into edges of G and edges of G^c, either there are p vertices all of which are joined by edges in G or there are q vertices all of which are joined by edges in G^c.

To say in other words, a number N has the (p, q) Ramsey property if and only if whenever we color the edges of K_N, the complete graph of N vertices, with each edge being colored either blue or red, then either K_N has a red K_p or a blue K_q.

Thus, to find $R(G_1, G_2, \ldots, G_t)$ we need to find the largest number N such that an i-colored complete graph K_N does not contain a subgraph G_i in color i, for $1 \leq i \leq t$. For example, $R(C_4, C_4, C_4) = 11$ means that there is no graph K_N, where $N \geq 11$, which can be 3-colored in such a way that there is no monochromatic quadrilateral, and such coloring exists on 10 vertices.

1.3 Background

Graph Ramsey theory has grown from nonexistence in early 60s to become one of the presently most active areas in Ramsey theory. To state in a simple way, let $G_1, G_2, \ldots, G_t$ be graphs. An integer N is said to have Ramsey property $(G_1, G_2, \ldots, G_t)$, if every coloring of edges of complete graph K_N in t colors $1, 2, \ldots, t$ give rise, for some i, to a subgraph that is (isomorphic to) G_i and is colored all in color i. The graph Ramsey number $R(G_1, G_2, \ldots, G_t)$ is smallest integer N with graph Ramsey property $(G_1, G_2, \ldots, G_t)$ [Rob].

One of the simplest and most general results in Graph Ramsey theory, due to

Chvátal and Harary [1972], is the following: For a graph G (without isolated vertices), let $\chi(G)$ denote the chromatic number of G and let $c(G)$ denote the cardinality of the largest connected component of G.

$$R(G,H) \geq (\chi(G)-1)(c(H)-1)+1$$

Using this results, Chvátal [1977] proved one of the most elegant results: For any tree T_m with m vertices

$$R(T_m,K_n) = (m-1)(n-1)+1$$

Several other interesting results can be found in [GRS]. There are well known results for computing the upper bounds of classical Ramsey numbers.

Much less research has been done in the area of multi-color graph Ramsey numbers as compared to classical Ramsey numbers. As described in next section, very few exact values are known. In all the problems considered in this thesis, the subgraph G_i is either a complete graph or a cycle of order 4 or 5. The number of colors t, was limited to 3, 4, or 5.

The objective of this thesis was to learn to apply concepts of genetic algorithms to Ramsey theory and try to find a new value of some Ramsey number or to improve the lower bounds. Another objective was to perform the search using the under utilized computing power of RIT computer science department.

The next section presents some of the known and unknown bounds of multi-color graph Ramsey numbers, on which the experiments were carried out.

A comprehensive list of bounds can be found in Dr. Radziszowski's paper [Ra1].

1.3.1 Known and unknown bounds

The only known value of a multi-color classical Ramsey number is

- $R(3,3,3) = R(3,3,3;2) = 17$ [GG]

Greenwood and Gleason define a 3-coloring of K_{16} labeling the vertices by $GF(16)$ and coloring $\{\alpha, \beta\}$ by the cubic character of $\alpha - \beta$. They prove that there are no monochromatic triangles and hence $R(3,3,3) = 17$.

These are some of the bounds of multi-color classical numbers:

$$55 \leq R(3,4,4) \leq 79 \qquad \text{[KLR]}$$

The result below is proved in [PR] by improving its upper bound with help of a computer algorithm.

$$30 \leq R(3,3,4) \leq 31 \qquad \text{[Ka2][PR]}$$

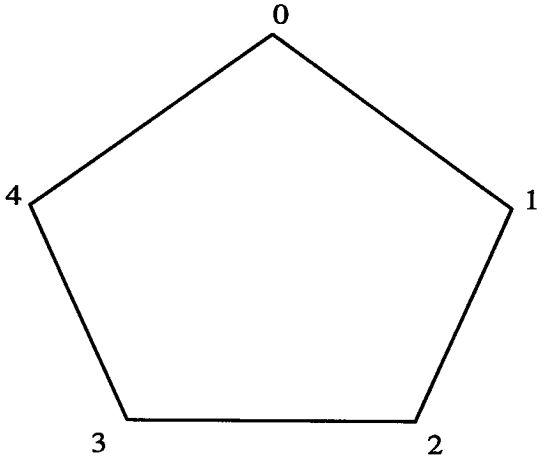
Multi-color general graphs:

$$R(C_4, C_4, C_4) = 11 \qquad \text{[BS]}$$
$$R(C_4, C_4, K_3) = 12 \qquad \text{[Sch]}$$
$$R(C_4, K_3, K_3) = 17 \qquad \text{[ER]}$$
$$R(C_5, C_5, C_5) = 17 \qquad \text{[Yr1]}$$
$$R(C_4, C_4, C_4, C_4) \geq 18 \qquad \text{[Ex1]}$$
$$R(C_4, C_4, C_4, C_4, C_4) \geq 25 \qquad \text{[Ex1]}$$

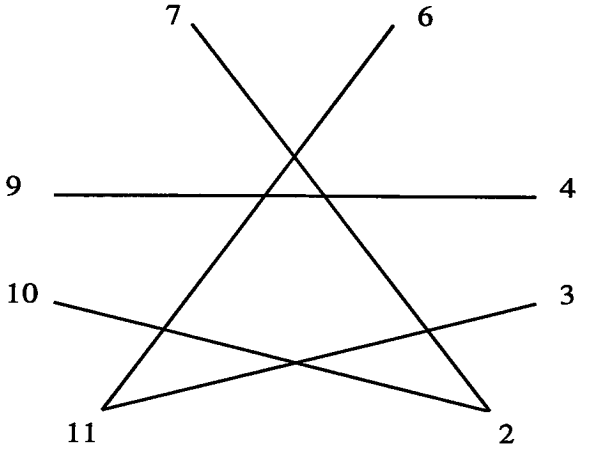
This thesis provides explicit lower bounds for previously uninvestigated Ramsey numbers as follows:

$$R(K_3, K_3, C_4, C_4) \geq 25$$
$$R(C_5, C_4, K_3) \geq 13$$

The figure on the next page shows some of the construction of graphs of n vertices containing neither k-clique nor l-independent set, which gives lower bound $R(k, l) > n$.



R(3,3) > 5



R(4,3) > 8

Figure 1: Small Ramsey Graphs

2 Genetic Algorithms

2.1 Introduction

Genetic algorithms are search algorithms based on the mechanics of natural selection and natural genetics. They combine survival of the fittest among string structures with a structured yet randomized information exchange to form a search algorithm with some innovative flair of human search. In every generation, a new set of artificial creatures (strings) is created using bits and pieces of the fittest of the old. While randomized, genetic algorithms are no simple random walk. They efficiently exploit historical information to speculate on new search points with expected improved performance [Gld].

In order for genetic algorithms to surpass their more traditional cousins in the quest for robustness, GAs must differ in some very fundamental ways. Genetic algorithms are different from other optimization and search procedures in four ways:

1. GAs work with a coding of the parameter set, not the parameters themselves.
2. GA search uses a population of points, not a single point.
3. GAs use payoff (objective function) information, not derivatives or other auxiliary knowledge.
4. GAs use probabilistic transition rules, not deterministic rules.

Genetic algorithms require the natural parameter set of the optimization problem to be coded as a finite-length string over some finite alphabet (but not for the order based ones).

A simple GA that yields good results in many practical problems is composed of three operators:

1. Selection
2. Crossover
3. Mutation

Selection is the process in which individual strings are chosen to breed (i.e., participate in crossover) according to their objective function values, f (also called fitness function). Selecting strings according to their fitness values means that strings with higher values have a higher probability of producing offspring.

Simple crossover follows selection. Each pair of selected strings undergoes crossing over as follows: an integer position k along the string is selected uniformly at random between 1 and the string length less one $[1, l-1]$. Two new strings are created by swapping all characters between positions, $k+1$ and l inclusively.

Mutation is the occasional (with small probability) random alteration of the value of a string position. The mutation operator, usually plays a secondary role in simple GA.

2.2 Representation and Approach

Using the problem definition from [Rob], we need to find an edge coloring for a K_N graph (for N as large as possible) avoiding G_i in color i.

The most common representation for storing colors one can think of, is an adjacency matrix, where each entry represents color of edge (i, j). The colors can be stored as numbers $0, 1, 2, \ldots$ in the array. Using this representation the entire population can be randomly initialized as strings of $0, 1, \ldots, t$ colors. As the graph is complete and undirected, we need to store only the lower triangle of matrix. This lower triangle can be mapped into a single dimensional array, the chromosome with the small alphabet $[t]$.

In a different approach, instead of using this string as a chromosome we use a permutation. A permutation represents the order in which the edges are colored. This representation can be combined with a greedy coloring approach, where each edge is colored, in order and using the lowest possible color number to avoid monochromatic G_i, penalizing use of higher number of colors.

In the computation of fitness, the non-optimal solution can be penalized by lowering its fitness value by one, every time it fails to avoid G_i in color i. The objective is to get a permutation such that number of colors required is not greater than m.

For the above representation, the *order based* Genetic Algorithm is used. We need an operator analogous to crossover, which permits exchange of impor-

tant ordering similarities between pairs of parents to form offspring. The following cross-overs were used for making children:

1. **Partially matched cross-over (PMX):** Under PMX, two strings (permutations) are aligned, and two crossing sites are picked uniformly at random along the strings. These two points define a *matching section* that is used to effect a cross through position by position exchange operators. That is, first perform a two-point crossover, then repair the children so they are valid permutations. (PMX arose in considering ways to tackle a blind traveling salesman problem).

2. **Order cross-over (OX):** The order crossover operator starts off in a manner similar to PMX, but instead of using point by point exchanges to effect the mapping, it uses a sliding motion to fill the holes left by transferring the mapped positions.

   Although PMX and OX are similar, they process different kind of similarities. PMX tends to respect absolute value positions, whereas OX tends to respect relative value position.

3. **Cycle cross-over (CX):** CX operator is a cross of different color. It performs recombination under the constraints that each element comes from one parent or the other.

The implementation of these cross-overs is attached in Appendix.

2.3 Algorithm and Implementation

The structure of the genetic algorithm used for Ramsey numbers in this thesis is shown below.

1. Randomly initialize the population.
2. Select two winners (chromosome) using a pair of $T-$element tournaments. Call them parents.
3. Perform cross-over, then mutate the children. The children replace the tournament losers of step 2.
4. Evaluate the fitness of children.
5. Repeat steps 2 through 5 until a specified optimum value is reached or for a fixed number of times.

The following parameters affect the behavior of this algorithm:

- population size
- mutation rate
- tournament size
- number of loops
- seed

2.3.1 Supporting functions

Given a permutation, of the $\binom{N}{2}$ edges of K_N, we need to color those edges in that order. As the edges are numbered (to map to single dimensional array), we need two end nodes of an edge to check if a graph G_i is being formed in color i.

A `look_up` table is used to get value of i for a given edge e. The table is initialized as:

```
k = 0
for i → 0 to N-1 do
    for j → 0 to i-1 do
        look_up[k] = i
        k = k+1
    end
end
```

and the value of j can be computed as `j = look_up[e] - i × (i-1)/2`. The color to be assigned to edge (i, j) needs to be composed with all other edges adjacent to nodes i and j to avoid G_i in color i. For example, to check if a triangle is being formed, the following function can be used.

```
for k → 0 to N-1 do
    if i, j, k are distinct then
        if color (i,j) = color (i,k) = color (j,k) then
            return false
        endif
    endif
end
return true
```

As the colors are stored in an array, to get color of edge (i,j) the index of array is computed as:

```
i × (i-1)/2 + j     if i > j
j × (j-1)/2 + i     if j > i
```

Using the function pointer of C language, the function to check of G_i is being formed can be changed at run-time depending on the problem and the number of colors.

The fitness function `fv(who)` uses the above functions to compute the `the_fitness` of individual string. The variable `who` denotes the individual `p[who]`, a row in the population i.e. the `p[POP_SIZE][individual_length]`. The value is computed as:

```
for i → 0 to individual_length do
    I = look_up[p[who][i]]
    J = p[who][i] - I × (I-1)/2
    this_color = 0
    done = 0
    while not done
        done = Check[this_color](I,J,this_color)
        if not done then
            this_color = this_color + 1
            if this_color ≥ no_of_colors then
                break
            endif
        endif
    end
    if this_color ≥ no_of_colors then
        the_fitness = the_fitness - 1
    endif
    color[p[who][i]] = this_color
end
```

where `color[]` is used to store the coloring, `Check[]` is the array of pointer functions and `no_of_colors` is a parameter indicating maximum number of colors for the current problem.

2.3.2 The main program

The main algorithm is given below:

```
init()
for who → 0 to POP_SIZE-1 do
    fitness[who] = fv(who)
    if hero ≥ MAX_HERO then break endif
end
trial = 0
while true
    trial = trial + 1
    if trial ≥ LOOPS then break endif
    tournament(tournament_size, p1, c1)
    tournament(tournament_size, p2, c2)
    make_children(p1, p2, c1, c2)
    if MUT_RATE ≥ 0.0 then
        mutate(c1), mutate(c2)
    endif
    fitness[c1] = fv(c1), fitness[c2] = fv(c2)
    if hero ≥ MAX_HERO then break endif
end
```

The main program starts by randomly initializing population p, value of MAX_HERO (the goal), individual_length, look_up table, etc. The fitness

value of each chromosome in the initial population is computed. There is a slight chance that the `MAX_HERO` will be found even before evolution.

The main loop of the program (**while** `true`) performs the actual work of genetic algorithm. A tournament is held twice to get two parents and two children. Every time the `tournament()` function is called it returns a winner and a loser to serve as parent and child respectively. The algorithm for `tournament()` is given below. The function which gets random `j` is written in such a way that the value of `j` is not repeated for that iteration.

After the tournament, a crossover is performed between selected parents to get two new children, who replace the the tournament losers. This loop continues till the desired maximum hero is reached or number of trials exceeds `LOOPS`.

```
winfit = -9999
losefit = 9999
for i → 0 to tournament_size-1 do
    get a random j
    if fitness[j] > winfit then
        winfit = fitness[j]
        winner = j
    endif
    if fitness[j] < losefit then
        losefit = fitness[j]
        loser = j
    endif
end
```

This approach is different from the one used in usual GAs, in a way the tournament is held [Gld]. The easiest way is to create a biased roulette wheel where each current string in the population has a roulette wheel slot sized in proportion to its fitness.

To select, we simply spin a weighted roulette wheel for POP_SIZE times. If a string's fitness value represents f percent of the total fitness, each spin turns up that string with a probability of $f/100$. In this way, more highly fit strings have a higher number of offsprings in succeeding generation.

Whenever a reproduction is necessary, the complete old population is replaced by new one by successively selecting parents and performing crossover on them to get new members of the population.

Where as, in the genetic algorithm used, *the present investigating* children are thrown back immediately into the population replacing the old ones.

3 Experiments

3.1 Experimental Setup

The search behavior in a GA depends on a number of parameters, such as: population size, mutation rate and tournament size. The initial population depends on value of 'seed' used to randomly initialize population.

Even if we take five values of each of the parameters the number of times the program is required to run is $5^4 \times 3$ (for each type of crossover). In order to reduce the computations, given three sets of five values each, the following arrangement requires 25 experimental runs.

	p	q	r	s	t
1	a	b	c	d	e
2	e	a	b	c	d
3	d	e	a	b	c
4	c	d	e	a	b
5	b	c	d	e	a

The massive computation involved in running the program 25 times for each type of cross-over and for each seed value, was divided by using multiple computers.

A UNIX shell script used for running the program with different combinations (25 in all) of three parameters, was run on different computers for different

seed values (different initial population) using the 'rsh' utility of UNIX. The results were stored in a file one for each problem and computer.

The use of multiple computers for experiments helped to speed up the search. The following values of population size, mutation rate and tournament size and their combinations were used for these experiments.

After doing some initial experiments the following values of parameters were fixed for all problems.

population size	mutation rate	tournament size
1000	0	2
2000	0.001	7
5000	0.005	12
15000	0.01	19
40000	0.05	26

Various different values of seed were used, specially for those Ramsey numbers whose exact values are now known.

3.2 Effect of various parameters

Even after performing several experiments, it was difficult to identify the effect of the three parameters, tournament size, population size, and mutation rate, on the search.

The main reason for this that the majority of solutions were found in the initial population, i.e., even before evolution. Thus these parameters were never actually used. The actual values (or range of values) of parameters for these problems along with the number of colorings found, is given below. The effect of parameters on more difficult problems is discussed below.

problem	seed	POP_SIZE	ts	MUT_RATE	colorings	N
$R(C_4, C_4, C_4)$	0-6000	1000-15000	2-17	0-0.05	30	9
$R(C_4, C_4, K_3)$	0-6000	1000-15000	2-17	0-0.05	25	12
$R(C_5, C_5, C_5)$	0-6000	1000-15000	2-17	0-0.05	43	16

The other problems, viz., $R(C_4, K_3, K_3)$, $R(C_4, C_4, K_3, K_3)$, and $R(C_5, C_4, K_3)$ required a large member of evaluations to achieve successful colorings.

- $R(C_4, K_3, K_3)$: The solution for this problem was found only when we used the smallest mutation rate of 0.001. The other parameters had no visible effect on the search. As high as 140,000 fitness value evaluations were required and 16 colorings were found for a complete graph of size 16.

- $R(C_4, C_4, K_3, K_3)$: Only 6 colorings were found for a complete graph of size 24. The successful parameters were, mutation rate was again 0.001, population size was 1000 and the tournament size was 7.

- $R(C_5, C_4, K_3)$: 83 colorings were found for a graph of size 12, but very few of them in the initial population. There was no specific effect of any of the parameters on the search.

4 Results & Conclusions

This section presents some of the results achieved using the genetic algorithm for search. The results were stored in the 'mc' (multicolored) format developed by Dr. Radziszowski for storing graphs.

The graphs are stored as follows: If c is the number of colors then $ceiling(\log(c))$ bits are used to code one entry of the matrix (actually half of it). Color 0 is used to handle partial colorings (not used in this thesis). 6 bits are sufficient to make one printable character. The first char of mc-format gives the number of vertices and the remaining bytes (6 bits of info each) as needed. The last one is possibly padded with 0-bits.

The results shown below are after decoding the corresponding 'mc' format. These are adjacency matrices showing the coloring of complete graphs K_N. The lines following the matrix show color number i, degrees of nodes 1 to N with adjacent edges in color i, number of edges in color i and number of triangles & quadrilaterals respectively. The first four values are previously known and were matched using this GA.

4.1 $R(C_4, C_4, C_4) = 10$

```
  2 2 1 3 3 3 1 1 2
2   1 1 1 2 3 3 2 3
2 1   2 1 1 2 3 3 2
1 1 2   3 2 2 1 3 3
3 1 1 3   2 1 3 1 2
3 2 1 2 2   3 1 2 1
3 3 2 2 1 3   2 1 1
1 3 3 1 3 1 2   2 1
1 2 3 3 1 2 1 2   3
2 3 2 3 2 1 1 1 3
```

```
0 deg: 0000000000,   0 edge,   0 tr
1 deg: 3333433433,  16 edge,   4 tr,  0 quad
2 deg: 3343243233,  15 edge,   3 tr,  0 quad
3 deg: 3323323333,  14 edge,   2 tr,  0 quad
```

4.2 $R(K_3, C_4, C_4) = 12$

```
  3 3 1 1 1 1 2 1 2 2
3   1 3 2 1 1 1 2 2 3
3 1   1 1 2 3 3 1 3 2
1 3 1   3 2 3 1 3 1 1
1 2 1 3   3 3 1 2 1 1
1 1 2 2 3   2 3 2 1 1
1 1 3 3 3 2   2 2 1 1
2 1 3 1 1 3 2   1 3 3
1 2 1 3 2 2 2 1   1 1
2 2 3 1 1 1 1 3 1   2
2 3 2 1 1 1 1 3 1 2
```

```
0 deg: 00000000000,   0 edge,   0 tr
1 deg: 54455444555,  25 edge,   0 tr
2 deg: 33212432433,  15 edge,   3 tr,  0 quad
3 deg: 23443234122,  15 edge,   2 tr,  0 quad
```

4.3 $R(K_3, K_3, C_4) = 17$

```
  3 2 3 2 1 2 1 1 3 2 2 1 3 2 1
3   2 1 2 1 3 1 1 1 2 3 1 3 2 2
2 2   2 1 2 1 1 2 1 1 3 1 2 3 3
3 1 2   3 3 1 2 3 2 1 2 2 1 2 1
2 2 1 3   1 3 2 1 2 3 1 2 2 3 1
1 1 2 3 1   1 2 3 2 1 2 2 1 2 3
2 3 1 1 3 1   2 1 2 3 1 3 2 1 2
1 1 1 2 2 2 2   2 3 2 1 3 1 1 3
1 1 2 3 1 3 1 2   2 1 3 3 1 2 2
3 1 1 2 2 2 2 3 2   2 1 3 1 3 1
2 2 1 1 3 1 3 2 1 2   1 2 2 1 3
2 3 3 2 1 2 1 1 3 1 1   1 2 3 2
1 1 1 2 2 2 3 3 3 3 2 1   1 1 2
3 3 2 1 2 1 2 1 1 1 2 2 1   2 3
2 2 3 2 3 2 1 1 2 3 1 3 1 2   1
1 2 3 1 1 3 2 3 2 1 3 2 2 3 1
```

```
0 deg: 0000000000000000,   0 edge,   0 tr
1 deg: 5665566665666655,  45 edge,   0 tr
2 deg: 6566665656655665,  45 edge,   0 tr
3 deg: 4434434344344345,  30 edge,   5 tr,  0 quad
```

4.4 $R(C_5, C_5, C_5) = 17$

```
  1 1 3 2 2 2 1 1 1 1 3 1 2 3 1
1   3 1 1 1 1 2 2 3 3 1 3 1 1 2
1 3   1 1 1 1 3 3 2 2 1 2 1 1 3
3 1 1   2 2 2 1 1 1 1 3 1 2 3 1
2 1 1 2   3 3 1 1 1 1 2 1 3 2 1
2 1 1 2 3   3 1 1 1 1 2 1 3 2 1
2 1 1 2 3 3   1 1 1 1 2 1 3 2 1
1 2 3 1 1 1 1   2 3 3 1 3 1 1 2
1 2 3 1 1 1 1 2   3 3 1 2 1 1 2
1 3 2 1 1 1 1 3 3   2 1 2 1 1 3
1 3 2 1 1 1 1 3 3 2   1 2 1 1 3
3 1 1 3 2 2 2 1 1 1 1   1 2 3 1
1 3 2 1 1 1 1 3 2 2 2 1   1 1 3
2 1 1 2 3 3 3 1 1 1 1 2 1   2 1
3 1 1 3 2 2 2 1 1 1 1 3 1 2   1
1 2 3 1 1 1 1 2 2 3 3 1 3 1 1
```

```
0 deg: 0000000000000000,   0 edge,   0 tr
1 deg: 8888888888888888,  64 edge,   0 tr
2 deg: 4334444343344443,  29 edge,   8 tr,  2 quad
3 deg: 3443333434433334,  27 edge,   8 tr,  2 quad
```

4.5 $R(C_5, C_4, K_3) \geq 13$

This new bound was found for previously uninvestigated Ramsey number. Even after trying a lot (performing lot of experiments with different parameters), this lower bound could not be improved using this GA.

```
  1 1 2 1 1 3 1 2 3 1 2
1   2 1 2 3 1 3 1 1 3 1
1 2   1 2 3 1 2 1 1 3 1
2 1 1   1 1 2 1 3 2 1 3
1 2 2 1   3 1 3 1 1 3 1
1 3 3 1 3   1 2 1 1 2 1
3 1 1 2 1 1   1 3 2 1 3
1 3 2 1 3 2 1   1 1 2 1
2 1 1 3 1 1 3 1   3 1 2
3 1 1 2 1 1 2 1 3   1 3
1 3 3 1 3 2 1 2 1 1   1
2 1 1 3 1 1 3 1 2 3 1
```

```
0 deg: 000000000000,   0 edge,   0 tr
1 deg: 666666666666,  36 edge,   0 tr
2 deg: 323322232222,  14 edge,   4 tr,  0 quad
3 deg: 232233323333,  16 edge,   0 tr
```

4.6 $R(K_3, K_3, C_4, C_4) \geq 25$

This is another new bound, found for previously uninvestigated Ramsey number. This bound could not be improved further using this GA.

```
  1 1 2 3 4 2 1 2 2 2 1 2 4 1 4 1 1 1 2 3 4 1 1
1   3 3 1 2 1 2 1 4 1 4 1 1 2 3 4 2 2 1 2 1 3 2
1 3   1 4 1 2 2 2 1 2 4 2 4 2 1 3 2 2 2 1 3 2 3
2 3 1   1 2 1 2 1 3 1 3 1 1 2 4 4 2 2 1 2 1 3 2
3 1 4 1   1 2 2 2 1 2 3 2 4 2 1 4 2 2 2 1 3 2 1
4 2 1 2 1   2 1 2 2 2 1 2 1 1 3 1 1 1 2 3 3 1 4
2 1 2 1 2 2   4 3 1 3 2 3 3 1 2 2 1 1 4 2 2 1 1
1 2 2 2 2 1 4   1 1 3 2 1 2 4 1 2 4 3 1 1 1 4 3
2 1 2 1 2 2 3 1   3 4 1 4 3 1 2 1 1 1 4 2 2 1 1
2 4 1 3 1 2 1 1 3   1 1 4 1 2 4 1 2 2 3 2 2 4 2
2 1 2 1 2 2 3 3 4 1   2 3 2 1 2 2 1 1 4 2 4 1 1
1 4 4 3 3 1 2 2 1 1 2   1 3 2 1 3 2 2 1 1 1 2 4
2 1 2 1 2 2 3 1 4 4 3 1   4 1 2 1 1 1 3 2 2 1 1
4 1 4 1 4 1 3 2 3 1 2 3 4   2 1 3 2 2 2 1 2 2 1
1 2 2 2 2 1 1 4 1 2 1 2 1 2   1 2 3 4 1 1 1 4 3
4 3 1 4 1 3 2 1 2 4 2 1 2 1 1   1 1 1 2 3 4 1 2
1 4 3 4 4 1 2 2 1 1 2 3 1 3 2 1   2 2 1 1 1 2 3
1 2 2 2 2 1 1 4 1 2 1 2 1 2 3 1 2   3 1 1 1 3 4
1 2 2 2 2 1 1 3 1 2 1 2 1 2 4 1 2 3   1 1 1 3 4
2 1 2 1 2 2 4 1 4 3 4 1 3 2 1 2 1 1 1   2 3 1 1
3 2 1 2 1 3 2 1 2 2 2 1 2 1 1 3 1 1 1 2   4 1 3
```

```
4 1 3 1 3 3 2 1 2 2 4 1 2 2 1 4 1 1 1 3 4   1 1
1 3 2 3 2 1 1 4 1 4 1 2 1 2 4 1 2 3 3 1 1 1   2
1 2 3 2 1 4 1 3 1 2 1 4 1 1 3 2 3 4 4 1 3 1 2

0 deg: 000000000000000000000000,   0 edge,   0 tr
1 deg: a9697a89a889a7aa9aaaaaa9, 109 edge,   0 tr
2 deg: 77a8a8977897788678878565,  90 edge,   0 tr
3 deg: 244433433334342343334445,  41 edge,   7 tr,  0 quad
4 deg: 433232243433343432231434,  36 edge,   4 tr,  0 quad
```

A Genetic Algorithm was applied successfully to one of the most interesting and difficult problems in combinatorics, the Ramsey numbers. As shown above, the bounds of existing $R(C_4, C_4, C_4)$, $R(C_4, C_4, K_3)$, $R(C_4, K_3, K_3)$, $R(C_5, C_5, C_5)$ were achieved and two new Ramsey numbers $R(C_4, C_4, K_3, K_3) \geq 25$ and $R(C_5, C_4, K_3) \geq 13$ were found.

The order based genetic algorithm combined with greedy approach was faster and efficient as compared to the usual string based approach.

References

[Ban] Bannani F., "Bounds of Classical Ramsey Numbers", Ph.D. Thesis, Dept. of Mathematics and Statistics, Carelton University, 1988

[BS] Bialostocki A. and Schönheim J., "On Some Turán and Ramsey Numbers", *Graph Theory and Combinatorics*, Academic Press, London, 1984, 29-33

[ER] Exoo G. and Reynolds D.F., "Ramsey Numbers Based on C_5 Decompositions", *Discrete Mathematics* 71, 1988, 119-127

[Ex1] Exoo G., "Ramsey Graphs with a Computer", *Congressus Numerantium* 59, 1987, 31-36

[Gld] Goldberg D.E., *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison Wesley, 1989.

[GRS] Graham R.L., Rothschild B.L., Spencer J.H., *Ramsey Theory*, John Wiley & Sons, 1980

[GG] Greenwood R.E. and Gleason A.M., "Combinatorial Relations and Chromatic Graphs", *Canadian Journal of Mathematics* 7, 1955, 1-7.

[Ka2] Kalbfleisch J.G., "Chromatic Graphs and Ramsey's Theorem", Ph.D. Thesis, University of Waterloo, January 1966.

[KLR] Kreher D.L., Wei Li and Radziszowski S.P., "Lower Bounds of Multi-Colored Ramsey Numbers From Group Orbits", *Journal of Combinatorial Mathematics and Combinatorial Computing* 4, 1988, 87-95.

[McN] McNamara J.N., "Two New Ramsey Numbers", MS Thesis, Rochester Institute of Technology, 1992.

[PR] Piwakowski K. and Radziszowski S., "New Upper Bound for the Ramsey Number $R(3,3,4)$ to appear.

[Ra1] Radziszowski S., "Small Ramsey Numbers", *Electronic Journal of Combinatorics*, 1996.

[Rob] Roberts F., *Applied Combinatorics*, Prentice Hall Inc., 1984, 325-348

[Sch] Schutle C.U., "Ramsey-Zahlen fűr Bäume and Kreise", Ph.D. Thesis, Heinrich-Heine-Universität Dűsseldorf, 1992.

[Wei] Wei Li, "An Algorithmic Approach for Multi-Color Ramsey Numbers", MS Thesis, Rochester Institute of Technology, 1987.

[Xin] Xin Jin, "Ramsey Numbers Involving Triangle: Theory and Algorithm", MS Thesis, Rochester Institute of Technology, 1993.

[Yr1] Yuabsheng Y. and Rowlinson P., "On the Third Ramsey Numbers of Graphs with Five Edges", *Journal of Combinatorial Mathematics and Combinatorial Computing* 11, 1992, 213-222.

APPENDIX

```
void pmx(A, B)
int *A, *B;
{
        int *As, *Bs;
        int     i, start  , end ;
        int LB = 0, UB = individual_length-1;
        /* subscript vector indexed by contents */
        As = ivector(LB, UB);
        Bs = ivector(LB, UB);
        for(i = LB; i <= UB; i++) {
                As[A[i]] = i;
                Bs[B[i]] = i;
        }
        end = random_int(individual_length-1);
        start = random_int(end-1);
        for(i = start; i <= end; i++) {
                int tmp = A[i];
                int tmp1 = B[Bs[tmp]];
                /* swap the contents of A[i] and the with the contents
                   of A at positon pointed by the contents of
                   As(subscript of A) at position B[i]
                */
                A[i] = A[As[B[i]]];
```

```
                A[As[B[i]]] = tmp;
                /* swap the contents of B[i] and the with the contents
                   of B at positon pointed by the contents of
                   Bs(subscript of B) at position A[i]
                */
                B[Bs[tmp]] = B[i];
                B[i] = tmp1;
        }
        free_ivector(As, LB, UB);
        free_ivector(Bs, LB, UB);
}

void cx(A, B)
int *A, *B;
{
        int *As, *swp;
        int i, stpt, current;
        int LB = 0, UB = individual_length-1;
        /* the subscript vector index by the contents of A */
        As = ivector(LB, UB);
        /*The vector indicating whether to swap or not to */
        /*  0 - not to swap */
        /*  1 - to swap */
        swp = ivector(LB, UB);
        for(i = LB; i <= UB; i++) {
```

```
                As[A[i]] = i;
                swp[i] = 1;
        }
        stpt = random_int(individual_length);
        swp[stpt] = 0;
        current = As[B[stpt]];
        /* i.e not reached the starting pt of the cycle*/
        while(current != stpt) {
                swp[current] = 0;
                current = As[B[current]];
        }
        for(i = LB; i <= UB; i++) {
                if (swp[i]) {
                        stpt = A[i];
                        A[i] = B[i];
                        B[i] = stpt;
                }
        }
        free_ivector(As, LB, UB);
        free_ivector(swp, LB, UB);
}

void ox( A, B )
int *A, *B;
{
```

```
int *As, *Bs;
int *Ac, *Bc;
int *At, *Bt;
int *tempA, *tempB;
int i, j, jA, jB, done, start = 0, end = 1;
int LB = 0, UB = individual_length-1;
As = ivector(LB, UB);
Bs = ivector(LB, UB);
Ac = ivector(LB, UB);
Bc = ivector(LB, UB);
At = ivector(LB, UB);
Bt = ivector(LB, UB);
tempA = ivector(LB, UB);
tempB = ivector(LB, UB);
for(i = LB; i <= UB; i++) {
        As[A[i]] = i; /* subscript A vector */
        At[i] = A[i]; /* copy of A */
        Bs[B[i]] = i; /* subscript B vector */
        Bt[i] = B[i]; /* copy of B */
        Ac[i] = 0;    /* the affected ones */
        Bc[i] = 0;
        tempA[i] = -1;
        tempB[i] = -1;
}
for(i = start ; i <= end; i++) {
```

```
        int tmp;


        Bc[Bs[At[i]]] = 1;
        Ac[As[Bt[i]]] = 1;
        tmp = A[i];
        A[i] = B[i];
        B[i] = tmp;
}
if (start > LB) {
        jA = start - 1;
        jB = start - 1;
}
else {
        jA = UB;
        jB = UB;
}
/* sort of moving the block right */
for(i = end; i >= start; i--) {
        if (!Ac[i]) {
                A[jA] = At[i];
                jA--;
                if (jA < LB) {
                        jA = UB;
                }
        }
```

```
        if (!Bc[i]) {
                B[jB] = Bt[i];
                jB--;
                if (jB < LB) {
                        jB = UB;
                }
        }
}
/* get the numbers to put in the remaining spots */
if (end == UB) {
        i = LB;
}
else {
        i = end + 1;
}
jA = LB;
jB = LB;
done = 0;
while(!done) {
        if (!Ac[i]) {
                tempA[jA] = At[i];
                jA++;
        }
        if (!Bc[i]) {
                tempB[jB] = Bt[i];
```

```
                jB++;
        }
        i++;
        if (i == start) {
                done = 1;
        }
        else {
                if (i > UB) {
                        i = LB;
                        if (i == start) done = 1;
                }
        }
}
if (end < UB) {
        jA = end + 1;
        jB = end + 1;
}
else {
        jA = LB;
        jB = LB;
}
i = 0;
while ((tempA[i] != -1) || (tempB[i] != -1)) {
        if (tempA[i] != -1) {
                A[jA] = tempA[i];
```

```
                        jA++;
                        if (jA > UB) {
                                jA = LB;
                        }
                }
                if (tempB[i] != -1) {
                        B[jB] = tempB[i];
                        jB++;
                        if (jB > UB) {
                                jB = LB;
                        }
                }
                i++;
        }
        free_ivector(As, LB, UB);
        free_ivector(Bs, LB, UB);
        free_ivector(Ac, LB, UB);
        free_ivector(Bc, LB, UB);
        free_ivector(At, LB, UB);
        free_ivector(Bt, LB, UB);
}
```