Effectiveness of Genetic Algorithms in Finding Steinhaus Graphs with Maximal Clique Size

Anurag M. Chandra

Honors Thesis
Department of Computer Science
Washington and Lee University

***Thesis Advisor*: Dr. Thomas P. Whaley**

May 1998

Acknowledgements

I would like to take this opportunity to thank several people. Firstly, I want to thank Dr. Whaley and Dr. Dymacek for being outstanding mentors to me during the past four years and for giving me the opportunity to conduct research with them on Steinhaus graphs and Parallel Computing during the summers of 1996 and 1997. I would like to thank Dr. Whaley for encouraging me to do this thesis, for providing guidance at every step throughout this process, and for being patient with me. I would like to thank Dr. Dymacek for also encouraging me in this process, for helping me find various documents on Steinhaus graphs, and for explaining graph theory. I would also like to thank Dr. Pamela Vermeer and Dr. Kenneth Lambert for all that they have taught me and for being supportive during my undergraduate career.

I would also like to thank the Massachusetts Institute of Technology for the use of GAlib. This GA package, available at http://lancet.mit.edu/ga, allowed me write the programs. I would like to thank Jeff Knudson of University Computing for helping me install GAlib.

The Computer Science seniors - Noah Egorin, John Thrall, John Hills, and Geoffrey Bourne – deserve many thanks for their support, competitive spirit, and friendship. I am thankful to Ryan Beaman, Rob Sein, and Ton Chartisathian for their friendship and for the experience of living at 8 North Main. And I would like to thank all the families that made me feel at home here in the US.

Lastly, I would like to express my deepest gratitude to my parents for everything they have done for me. Without them, this effort would not have been possible.

Anurag Chandra
Lexington, VA
May 1998

Table of Contents

Introduction

The goal of this research was to study the effectiveness of genetic algorithms (GA) in finding Steinhaus graphs of a given size whose maximal cliques were as large as possible. This investigation was based on the results obtained by Dr. Wayne Dymacek (Professor of Mathematics, Washington and Lee University) and Dr. Thomas Whaley (Professor of Computer Science, Washington and Lee University) during their research at Washington and Lee University in 1992. In addition, a report by mathematicians Daekeun Lim and Jin Hwan Kim on cliques in Steinhaus graphs also provided results related to this research.

In the first chapter of this thesis, the concept of genetic algorithms will be discussed. The various characteristics of GAs, their applications, and certain theoretical aspects will be described.

The second chapter will focus on Steinhaus graphs and the occurrence of maximal clique size in Steinhaus graphs. These graphs are a special type of undirected graphs that have been studied by Dr. Dymacek and Dr. Whaley for numerous years.

Chapter 3 will discuss the reasons for studying GAs using this particular problem. It will also explain how GAs were applied to the problem and will include a description of the algorithm.

Finally, in Chapter 4, the experiments performed during this research and their results will be explained. Chapter 5 will conclude this paper by looking at some future research possibilities for GAs applied to this problem.

Chapter 1: An Introduction to Genetic Algorithms

Evolutionary computation simulates biological evolution where computer systems evolve a population of solutions to a given problem (Mitchell 2). Genetic algorithms are the chief example of evolutionary computation. GAs are adaptive algorithms which are used to solve optimization and search problems. Evolution strategies, evolutionary programming, and genetic programming are other areas closely related to GAs.

Individuals in a population in nature compete for available resources and to attract mates. The individuals that are most successful in attracting mates and surviving will have a greater probability of creating offspring while other individuals will be unable to reproduce because they cannot survive or attract mates. Each successive generation produces a fitter population, and the characteristics of highly fit individuals will eventually be spread to more members of the population. Thus, the species will evolve as generations pass.

John Holland developed the fundamental rudiments of GAs during the 1960s and 1970s at the University of Michigan. Holland's GAs mimic the process of evolution of natural populations according to Charles Darwin's model of the "survival of the fittest" (Beasley, Bull, and Martin, "Part 1" 1). In genetic algorithms, each individual member of the population represents a possible solution to the given problem. The individuals are assigned fitness scores using the fitness function that measures how good that solution is for the problem. Individuals with higher fitness are given greater opportunity to mate with other members of the population to produce the offspring for the next generation. The least fit individuals are less likely to reproduce and will be eliminated. Thus, the new generation will contain more characteristics of the fitter individuals of the previous generation. The evolution of the population will converge toward a near-optimal solution to the problem. For any given problem, not all candidate solutions are evaluated; instead, the GA finds the good solutions by searching only a small portion of the possible candidates.

In this chapter, we discuss the basic features of genetic algorithms. Section 1.1 will outline the characteristics of GAs. In Sections 1.2, 1.3, and 1.4, we describe solution coding, fitness functions, and the reproduction operators. In sections 1.5 and 1.6, theoretical aspects of GAs and applications of GAs are addressed.

1.1 Basic Characteristics of GAs

The following steps are common to all GAs: populations of candidate solutions, selection according to fitness of an individual, offspring generated by crossover, and random mutation of the offspring (Mitchell 8). However, before executing a GA, the solutions must have a representation or coding, a fitness function must be created, and crossover techniques and the mutation probability must be established. Figure 1.1 illustrates a standard GA (Beasley, Bull, and Martin, "Part 1" 3).

```
BEGIN /* genetic algorithm */
    Generate initial population
    Compute fitness of each individual

    WHILE NOT finished DO
    BEGIN /* produce new generation */
        FOR population_size/2 DO
        BEGIN /* reproductive cycle */
            Select two individuals from old generation for mating
                /* biassed in favour of the fitter ones */
            Recombine the two individuals to give two offspring
            Compute fitness of the two offspring
            Insert new offspring in new generation
        END

        IF population has converged THEN
            finished := TRUE
    END
END
```

Figure 1.1: A Traditional Genetic Algorithm

1.2 Solution Coding

A potential solution to an optimization problem is a representation of the set of parameters. Chromosomes in a GA population represent possible solutions for the problem, and they are individuals that make up the population. A chromosome may be designed by putting together in a string a set of parameters for the problem. Usually, these chromosomes are in the form of bit strings. All members of the population have the same length.

For instance, one might want to maximize the following function: $f(y) = y + |\sin(32*y)|$, $0 \leq y < \Pi$. In this case, the parameters for this function are the real values for *y*. This variable may be represented using a binary string. During the fitness calculation, the binary string will be converted to a real number, and *f(y)* will be evaluated with this value (Mitchell 9).

1.3 Fitness Function

The fitness of each selected chromosome is assessed by the fitness function. Fitness is a numerical score for the particular chromosome according to how well the chromosome solves the given problem. This score is supposed to be "proportional to the 'utility' or 'ability' of the individual which the chromosome represents" (Beasley, Bull, and Martin, "Part 1" 3). For each problem, a fitness function must be designed. Chromosomes with higher fitness scores will be given a greater likelihood of contributing to the next generation. Therefore, the design of a good fitness function is key to the success of the GA.

Usually in function optimization problems, the fitness function is simply the function being optimized. In the example of maximizing $f(x,y) = x^2 - y^2$, the fitness function will find the value of $f(x,y)$ depending on the values of x and y which are represented by the chromosome. So if x and y were 17 and 5 respectively, $f(x,y) = 17^2 - 5^2 = 289 - 25 = 264$ would be the fitness value. The higher the value of $f(x,y)$, the better the fitness of the chromosome will be. Thus, in future generations, the probability of reproducing with this chromosome as a parent is high. The GA will get a greater chance of obtaining offspring which maximize $f(x,y)$.

1.4 GA Reproduction Operators

In the reproduction stage of the GA, chromosomes are selected from the total population and are recombined to produce offspring. The selection process is based on the fitness of the chromosomes and gives more opportunities to fitter chromosomes for being involved in reproduction. After being added to the mating pool, the chromosomes undergo *crossover* and *mutation* in order to create the offspring.

Crossover This operator takes two chromosomes and cuts them at a randomly chosen point and produces two "head" segments and two "tail" segments (Beasley, Bull, and Martin, "Part 1" 3). By swapping the "tail" segments, two new chromosomes are produced which become the offspring. This is called single point crossover and is illustrated in Figure 1.2. Thus, there are two new chromosomes that inherit some characteristics of the parent chromosomes.

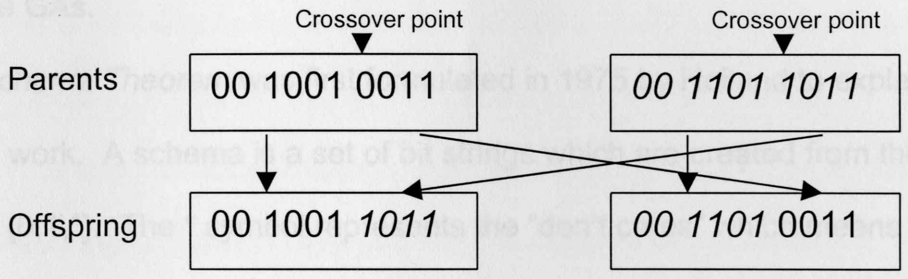


Figure 1.2: Single Point Crossover

Mutation This operator is used to alter one or more genes (positions) in a chromosome. Some of the bits of a chromosome are randomly chosen and flipped with a small probability. Figure 1.3 shows the implementation of the mutation operator. The mutation rate is the probability of modifying each gene in the chromosome.

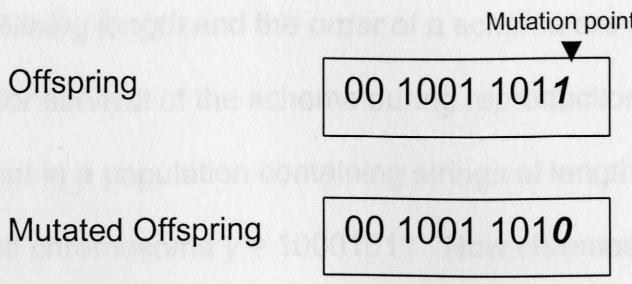


Figure 1.3: Single mutation

1.5 Theoretical Aspects of GAs

GAs are simple to describe and implement. However, the reason why they work is more difficult to explain. John Holland's *Schema Theorem* and other hypotheses have been put forth and help us comprehend better the theoretical aspects of the GAs.

The *Schema Theorem* was first formulated in 1975 by Holland to explain how the GAs work. A schema is a set of bit strings which are created from the alphabet A = {0,1,*}. The * symbol represents the "don't cares" which means that it can match anything. For instance, the schema H = *1*11 matches four strings: {01011, 01111, 11011, 11111}. These four strings are *instances* of H (Mitchell 27). Similarly, some of the schema contained in the chromosome 00011 are 0***1, 000**, ***11, and *0*1*. The number of non-* symbols in a schema is called the *order* of the schema. In other words, the order is the number of fixed positions in a schema. Schema H is of *order* 3. The *defining length* of the schema is the distance between the outermost fixed positions. For schema H, the *defining length* is 3.

The *defining length* and the *order* of a schema are important in calculating probabilities for survival of the schema during reproduction (Michalewicz 44). Let us assume that in a population containing strings of length 8, chromosome *x* = 01100110 and chromosome *y* = 10001011. Now chromosome *x* matches, among others, the schema S_1 = 0******0 and S_2 = **10****. Assume further that both *x* and *y* have been selected for crossover and the crossover point is at position 6. Thus the offspring produced from *x* = 011001|10 and *y* = 100010|11 will be x_1 = 01**10**0111 and y_1 = 10001010. S_2 is kept intact in offspring x_1. S_1 is destroyed because the fixed positions were at the front and rear of the schemata, and no offspring contained it after crossover. A shorter *defining length*, as in the case of S_2, will give a greater chance to the schemata to survive crossover. With mutation, the order of the schema is important to its survival. Let us assume that x_1 = 01**10**0111 undergoes mutation and the mutation point is at position 2. Then x_{11} = 00**10**0111 and the mutated offspring still preserves schema S_2. If S_2 had a bigger *order*, then the probability of destroying the schema would be higher.

Thus, it can be seen that short, low order schema have a better probability of surviving reproduction. According to Holland, a chromosome's high fitness results from the good schemata it has, and passing this good schemata to the next generation will enhance the likelihood of finding better solutions. Holland's *Schema Theorem* concludes that short, low-order schemas with above-average fitness will receive an exponentially increasing number of reproductive opportunities in successive generations (Michalewicz 51).

David E. Goldberg's Building Block Hypothesis asserts that the GA is powerful because it is able to find good building blocks that are schemata of short *defining length* and low *order* (41-42). The idea behind this hypothesis is that the GA tries to construct better strings with each generation based on the good schemata from the previous generation. The GA does not try outright to achieve the optimal solution by experimenting with all combinations of solutions. This leads to improved performance of the population as they evolve for many generations.

1.6 Applications of GAs

GAs have been used for solving a wide variety of problems. Some of the applications are listed below and illustrate the flexibility of genetic algorithms.

Optimization GAs have been used extensively for research purposes in this area. For numerical optimization, GAs have proven to be better than conventional optimization techniques on difficult, discontinuous, multimodal functions. GAs have been applied to combinatorial optimization problems such as the traveling salesperson problem, bin packing, job-shop scheduling, and circuit layout (Beasley, Bull, and Martin, "Part 1" 13).

Machine learning GAs have been used to evolve rules for machine learning systems such as neural network weights, maze solving, sensors for robots, and classifier systems.

Economics Models of the development of bidding strategies and economic market emergence have been implemented using GAs.

Image processing With x-rays and satellite pictures, two images of the same area, which are taken at different times, are aligned by the use of GAs. The GA finds a set of equations that can alter one image and make it fit onto the other.

Evolution The study of individual learning, the evolution of species, and their effect on each other is conducted with the use of GAs too (Mitchell 15-16).

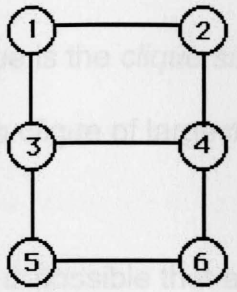
Chapter 2: Steinhaus Graphs

At Washington and Lee University, Dr. Wayne Dymacek, Dr. Thomas Whaley, and several undergraduate students participating in the R.E. Lee Summer Research Program have conducted research on various properties of *Steinhaus graphs* and their complements for many years. Among the graph properties investigated are the following: bipartite graphs, planarity, degree sequences, edge counts, chromatic number, and maximal clique size.

In this chapter, we introduce the notion of a Steinhaus graph and address some related ideas. In particular, we discuss the occurrence of maximal clique size in Steinhaus graphs.

2.1 The Definition and Organization of Steinhaus Graphs

Steinhaus graphs are a special class of undirected graphs. Let $T = a_{1,1}a_{1,2}\ldots a_{1,n}$ be a string of 0's and 1's of length n. T can be used to generate the *Steinhaus matrix*, A = $[a_{i,j}]$, of a Steinhaus graph as follows. All entries along the main diagonal contain a zero. So $a_{1,1}, a_{2,2}, \ldots, a_{n,n}$ are zeroes. A *Steinhaus triangle* is the upper-triangular part of the Steinhaus matrix and excludes the main diagonal. Since the first row of the Steinhaus matrix is given by T, the remaining cells of the Steinhaus triangle can be computed using the Exclusive-Or operation. This operation is applied to the cell directly above and the cell that is northwest of the entry being calculated. If both these cells contain the same value, either 0 or 1, then the entry being computed receives a 0. Otherwise, the entry receives a 1. Since Steinhaus graphs are undirected, they are symmetric along the main diagonal. As a result of this property, the lower-triangular part of the Steinhaus matrix is filled in by symmetry. Figure 2.1 is an example of a Steinhaus graph and its adjacency matrix.



	1	**2**	**3**	**4**	**5**	**6**
1	0	1	1	0	0	0
2	1	0	0	1	0	0
3	1	0	0	1	1	0
4	0	1	1	0	0	1
5	0	0	1	0	0	1
6	0	0	0	1	1	0

Figure 2.1: A Steinhaus graph (011000) and its matrix

Since the adjacency matrix of a Steinhaus graph is determined by the first row $[a_{1,j}]$ from j=2 to n, this is called the *generating string* of the graph. Thus, a generating string of length n-1 identifies a Steinhaus graph with n vertices. Counting all possible generating strings for n vertices yields a total of 2^{n-1} Steinhaus graphs. For example, the total number of Steinhaus graphs with 15 vertices is 2^{14}. The *size* of a Steinhaus graph is the number of vertices in the graph. The row number in the matrix corresponds to the vertex in the graph.

2.2 Cliques in Steinhaus Graphs

A clique in an undirected graph G = (V, E) is a subset V' of V, each pair of which is connected by an edge in E (Cormen, Lieserson, and Rivest 947). It is, thus, a complete sub-graph contained within the graph G. The number of vertices contained in a clique is the *clique size*. An optimization problem involving cliques is finding a clique of largest possible size in graphs of a certain size.

In general graphs, it is possible that a graph of size *n* will have a maximal clique of size *n*; i.e., a complete graph of *n* vertices. However, in the case of Steinhaus graphs, all vertices are not adjacent to each other. Hence, the maximal clique size of Steinhaus graphs with *n* vertices will be less than *n*. The size of the largest clique in Steinhaus graphs with *n* vertices, for $n \geq 2$, is bound by $\lceil (n + 3)/3 \rceil$ (Lim and Kim 2). For example, for Steinhaus graphs of size 22, the maximal clique size is 9.

Chapter 3: Applying GAs to Steinhaus Graphs

Let MCS(G) be the size of the largest clique is graph G. SMCS(n) is defined to be the maximum of MCS(G) where G is a Steinhaus graph with n vertices. Then, a Steinhaus graph of size n has the Maximal Clique Property (MCP) if MCS(G) = SMCS(n). In other words, G has the MCP if G is a Steinhaus graph whose maximal clique is as large as possible among Steinhaus graphs of its size. The objective of this research is to investigate the effectiveness of GAs at finding Steinhaus graphs with the MCP.

Section 3.1 examines the reasons for choosing the problem of Steinhaus graphs with MCP to study GAs. The design of the program and implementation of the GAs is explained in Section 3.2.

3.1 Why apply GAs to Steinhaus graphs with MCP?

During their research in 1992, Dr. Dymacek and Dr. Whaley had gathered data on Steinhaus graphs with MCP. The data from this research is available in Appendix A. It shows, through size 26, the number of Steinhaus graphs with n vertices that have a maximal clique size of k. For $n = 3m + 1$, $m \geq 4$, the data suggests that there are four Steinhaus graphs with n vertices having MCP, and for these graphs, MCS(G) = $m + 2$. Thus, we know from Appendix A that there are only four Steinhaus graphs of sizes 13, 16, 19, 22, and 25 that have the MCP. This empirical information provided the basis for testing the GAs developed in this research.

In addition, during the period of this research, it was discovered that Daekeun Lim and Jin Hwan Kim have recently classified cliques in Steinhaus graphs. Their research determined the generating strings of the graphs with MCP. They proved that for $n \geq 25$ vertices, out of 2^{n-1} graphs, the number of graphs with the MCP is four if $n = 3m + 1$. Although they did not assert this fact for the smaller sizes, the identification of these Steinhaus graphs confirmed the empirical data gathered at Washington and Lee University.

The generating string of the Steinhaus graphs provides a straightforward encoding for the chromosomes in the population of the GA. Additionally, the fitness function would calculate the MCS(G) where G was a Steinhaus graph of the given size. These are discussed further in Section 3.2. The SMCS(n) was available from the previous research. It was also known that the targeted graphs were rare among the huge search space for larger sizes of Steinhaus graphs.

Thus, this problem contained all the components for studying the application of GAs – good encoding, obvious fitness function, large search space, few individuals with optimal values and empirical data for confirming the results.

3.2 Design and Implementation of the GAs

Different tests were conducted to find the graphs with the MCP. These noted the effects of the different elements of the GAs, such as mutation and crossover, on the ability to find these graphs. The pseudo-code for the genetic algorithm used in this research is shown in Figure 3.1.

```
BEGIN /* Clique GA */
    User input:  graph size, maximal clique size, population size,
        number of generations, percentages for crossover and
        mutation, number of runs

    WHILE NOT completed_runs DO
    BEGIN /* Run GA */
        Generate random initial population
        Compute clique size (fitness) of each individual

        WHILE NOT finished_GA DO
        BEGIN /* produce new generation */
            Reproduce through crossover
            Compute clique size (fitness) of offspring
            Mutate some offspring
            Insert new offspring in new generation

            IF best fitness is maximal clique size THEN
            finished_GA := TRUE
        END

        Print results from this run
        IF number of runs completed THEN
            completed_runs := TRUE
    END
END
```

Figure 3.1: Algorithm using GA to find graphs with maximal clique size

As indicated in this pseudo-code, the user has the responsibility of setting the population size and the number of generations to evolve this population. The user also controls the likelihood of applying crossover and mutation to the parents and offspring respectively. The user also inputs the size of the graphs to consider and the known optimal value.

The binary generating strings serve as the encoding (chromosomes) for the graphs. In the fitness function, the chromosome is expanded to form the adjacency matrix for that particular graph, G. The fitness function then calculates MCS(G). The GA is seeking to find the graphs that have the MCP. The clique size as the fitness function is correctly representative of the utility of the chromosome.

The research involved the investigation of different ways of forming populations in each generation (overlapping versus non-overlapping), different methods of crossover (single point versus two-point), and the effect of the rate of mutation. The results will be discussed in the next chapter.

Chapter 4: Experiments and Results

The experiments were designed to explore the effectiveness of genetic algorithms in finding the Steinhaus graphs with maximal clique size. In this chapter, five experiments will be described and their results will be discussed. In these experiments, results for all the sizes were very similar, and so, only results of Steinhaus graphs of size 22 are shown.

For all the experiments, the new generations were created by (1) selecting individuals for reproduction according to fitness, (2) applying crossover to these selected individuals according to the crossover rate, and (3) applying mutation to the offspring according to the mutation rate.

Any individual in the population has a probability *p* of being selected for reproduction where *p* is the fitness of the individual divided by the sum of the fitness of all individuals in the population (Michalewicz 32). Hence, individuals with higher fitness will have a higher probability of being selected for reproduction.

The crossover rate, which is the probability of crossover, is usually set between 0.6 and 1.0, and in these experiments, it was fixed at 0.9. The crossover rate gives the expected number of chromosomes that will undergo crossover after being selected for mating. This number is equal to the Population Size * Crossover Rate. A random real number between 0 and 1 is chosen for each chromosome in the mating pool, and if this number is less than the crossover rate, the chromosome is selected for crossover. Crossover is then applied randomly to these newly selected individuals (Michalewicz 33). The chromosomes in the mating pool that do not participate in crossover are replicated to produce the offspring. In Section 4.3, single point crossover is compared to two-point crossover.

The mutation rate, the probability of mutation, gives the expected number of bits to be mutated in each generation. This number is equal to the Mutation rate * Chromosome Length * Population size. For instance, if the population size is 128, and the length of the chromosome is 21, the total number of bits in the population is 2688. For each bit in the population, a random real number between 0 and 1 is chosen. If this number is less than the mutation rate, the bit is flipped (Michalewicz 33). If the mutation rate is set to 0.1, then the expected number of mutated bits in each generation would be 268.8. Different rates of mutation were used, and their effect will be described in Section 4.2.

Population sizes varied by powers of two from 32 to 128. Similarly, the number of generations was set in powers of two from 128 to 1024. This method was chosen to enable easier calculation of the total number of graphs computed by the GA and will be described in further detail in Section 4.4.

In the first four experiments, the gathered data is the probability of finding one Steinhaus graph with the MCP; i.e., how often the GA will find one of these graphs, if it is run 100 times. For example, with the single point crossover rate fixed at 0.9, the mutation rate set to 0.1, and with a population size of 64, which is evolved for 512 generations, the GA found one Steinhaus graph with the MCP about 91% of the times it was run. In all these experiments, the different success rates possible under the different circumstances are observed and discussed.

The last experiment, explained in Section 4.5, focuses on which Steinhaus graphs, of the above mentioned sizes, with MCP are found by the GA.

4.1 Overlapping and Non-overlapping Populations

The genetic algorithm evolves a population of individuals over a given number of generations. There are two methods to form the new population for each generation. The first experiment was to compare the GAs usage of these two methods and their effect on finding graphs with the MCP successfully.

Firstly, non-overlapping populations were used in the GA. David E. Goldberg describes a GA as a simple GA when it uses non-overlapping populations (41). The simple GA creates an initial population, which is based on the specifications of the population size and the chromosome size entered by the user. The use of non-overlapping populations by the GA creates an entirely new population at each generation. Chromosomes from the previous population are selected for reproduction, and they undergo crossover to create offspring for this new population. The new population still has the same number of individuals as the previous population. The GA is stopped when the termination criteria, such as convergence of population or number of generations, is met.

Overlapping populations was the second form of populations used. This type of GA is also known as a steady state GA. Similar to the simple GA, the steady state GA creates an initial population dependent on the input of the user and concludes when the termination criteria are achieved. The difference in this GA lies in the reproductive stage when the new population is formed for the next generation. In each generation, the algorithm selects chromosomes for reproduction and creates a temporary population containing the offspring produced from the crossover. This population of offspring is evaluated according to the fitness function and is added to the main population. Then the population is truncated to the original size by discarding the chromosomes with the lowest fitness. It is possible that the newly generated offspring are discarded. Either a specified number of individuals can be removed each generation or removal may be based on a percentage of the population. All experiments conducted in this research used the replacement percentage. The replacement percentage is the percentage of the population that will be replaced each generation and was kept constant at 0.6.

4.1.1 Results

All the variables, such as crossover and mutation rates, were kept constant, and then the simple GA and the steady state GA were run on the same population sizes for the given number of generations. Table 4.1 displays the data comparing the simple GA and the steady state GA.

Table 4.1 clearly indicates the higher success rate of the GA with overlapping populations. Using a steady state GA, one graph with the MCP was found a very high percentage of the time while the simple GA found these graphs a much smaller percentage of the time. For all sizes that were tested and at different mutation levels for each size, the result was the same: the use of overlapping populations in the GA outperformed the use of non-overlapping populations. This is possibly because in the steady state GA, the worst individuals in the populations are discarded in each generation and replaced by better ones. So the overall fitness of the population is higher, and the population is able to converge faster. Thus, the steady state GA finds the graphs with maximal cliques faster than the simple GA. Since the steady state GA was more successful, it was used in the rest of the experiments.

Population Size	Number of Generations	Overlapping Populations	Non-overlapping Populations
32	*1024*	0.85	0.4
64	*256*	0.91	0.1
64	*512*	0.91	0.2
128	*256*	1.0	0.13
128	*512*	1.0	0.33

Table 4.1: Probability of finding one graph with MCP, using overlapping and non-overlapping populations (Size 22, Mutation = 0.1)

4.2 Effects of Mutation

For the conducted experiments, the optimal level of mutation was not known. Hence, for all experiments, different levels of mutations were tested. Trial and error finally produced the best mutation rates for the GA.

4.2.1 Results

The results of the effects of mutation are shown below in Table 4.2. The population sizes and number of generations were kept constant, and mutation rates were changed.

Mutation Level	Success Rate
0.01	0.4
0.05	0.96
0.1	1.0
0.15	0.7
0.2	0.45

Table 4.2: Probability of finding one Steinhaus graph with MCP, using different mutation levels (Size 22, Population = 128, Generations = 256)

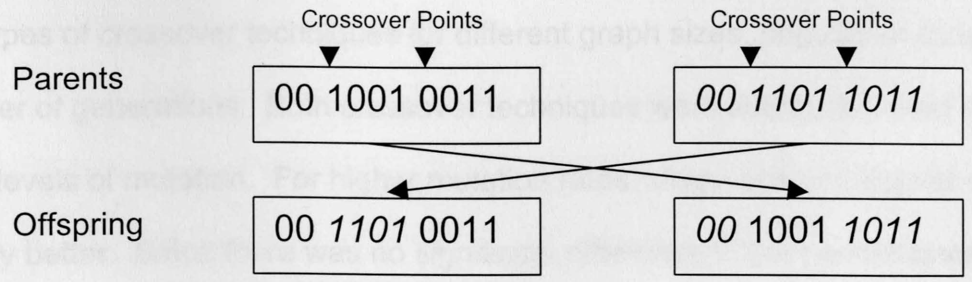
The mutation rate is just a small factor in the reproductive phase of the GA and is typically between 0.001 to 0.01 (Beasley, Bull, and Martin, "Part 1" 3). However, using such low levels of mutation is not helpful to our problem, and greater than normal levels of mutation have to be used to produce the best results. When very little or no mutation is used, the probability of finding graphs with the MCP is very low. As seen in Table 4.2, for the mutation rate of 0.01, the success of the GA is only 40%. With less mutation, the success rate is even lower. Similarly, for very high levels of mutation, the GA did not perform well. As the mutation level increased beyond 0.15, the performance of the GA fell rapidly.

The experiments showed that the optimal levels of mutation for this problem were 0.05 and 0.1. Table 4.2 reiterates this fact. One would assume that for different population sizes as well as for different sizes of the Steinhaus graphs, the optimal mutation level would vary. However, this was not the case. For all population sizes, the best results were consistently found when the mutation levels were set at 0.05 and 0.1. This observation also holds for all sizes of Steinhaus graphs that were considered in this research.

4.3 Crossover: Single Point and Two-Point

Single point crossover and two-point crossover are the most commonly used techniques of crossover. Other forms of crossover include multi-point crossover and uniform crossover. This experiment examines the use of single point crossover and two-point crossover in the GA and their effectiveness in finding Steinhaus graphs with maximal clique size.

Single point crossover was explained in Chapter 1. Two-point crossover follows the same principle except for cutting the chromosomes in two points rather than one point as in single point crossover. Cutting the chromosome in two points divides it into three pieces. Offspring A will receive the first and the third pieces of the chromosome from the first parent while Offspring B will get the first and third pieces from the second parent. The middle pieces of the parent chromosomes are then swapped. Thus, two new offspring are created. This is illustrated in Figure 4.1.



<u>Figure 4.1</u>: Two-Point Crossover

With the addition of one more crossover point, more building blocks and schema are likely to be disturbed. However, the search space may be searched more thoroughly with this procedure.

4.3.1 Results

Table 4.3 compares the performance of the single point crossover and the two-point crossover. Again, all variables were kept constant, and only the crossover technique was changed in the GA.

Mutation Level	Single Point Crossover	Two-Point Crossover
0.05	0.88	0.8
0.1	0.91	0.95

Table 4.3: Probability of finding one graph with the MCP using different crossovers (Size 22, Population = 64, Generations = 512)

The figures in Table 4.3 are representative of similar success rates of the two types of crossover techniques for different graph sizes, population sizes, and number of generations. Both crossover techniques were evenly matched for most levels of mutation. For higher mutation rates, single point crossover was slightly better. Since there was no significant difference in the performance between the two crossover techniques, the use of either technique by the GA would have produced good results. All of our other experiments only used single point.

4.4 Total Number of Graphs Computed

An exhaustive search to find all Steinhaus graphs with the MCP would inspect all the graphs of the given size. Since the sizes of Steinhaus graphs that were considered in this research had only four graphs with the MCP, in order to find one of these graphs, one fourth of the number of graphs would have to be searched on average. For example, there are 2^{21} graphs of size 22, out which there are four graphs that have the MCP. On the average, we would expect to inspect 2^{19} graphs chosen at random before finding one with the MCP.

In each generation, every member of the population of the GA has its fitness measured. As graphs are the members of the population, every graph's clique size is computed by the fitness function. Since a genetic algorithm evolves its population for a given number of generations, the total number of graphs evaluated in a genetic algorithm is as follows:

*Number of graphs evaluated = P * G*

Here, P represents the size of the population used in the GA, and G denotes the number of generations.

Table 4.4 displays results gathered from tests where the population size is 128 and the number of generations is 512. So the total number of graphs evaluated in this experiment is at most $2^7 * 2^9$, which equals 2^{16}.

4.4.1 Results

This experiment searched for the optimal combination of population size and number of generations that would yield a high probability of finding a graph with the MCP.

The best performance of the GA for this problem is shown in Table 4.4. For the mutation rates of 0.05 and 0.1, the GA was able to find one graph with the MCP every time. So the combination where the population size is set to 2^7 and the number of generations is 2^9 is very effective in finding these graphs. With this combination, the GA finds one graph with the MCP by computing at most 2^{16} graphs. The combinations of population size and number of generations shown in Table 4.2 and Table 4.3 are not as effective but still have a good success rate. In Table 4.2, a population of 2^7 graphs is evolved for 2^8 generations, while in Table 4.3, a population of 2^6 is evolved for 2^9 generations. Thus, at most, the total number of graphs the GA computes in these instances is 2^{15}. The use of the GAs gives a high probability of finding these graphs unlike the other method in which searching 2^{19} random graphs might not yield the graph with the MCP.

Mutation Level	**Success Rate**
0.05	1.0
0.1	1.0

Table 4.4: Probability of finding one graph with the MCP

(Size 22, Population = 128, Generations = 512)

We observed above that the total number of graphs evaluated in the GA's best performance is 2^{16}. This figure, however, is an upper bound for the number of graphs evaluated. So, when the GA is executed with a population size of 128 and the user inputs the number of generations as 512, this population does not necessarily evolve for all those generations. The GA stops when the graph is found, and this event usually occurs before 200 generations. Table 4.5 displays the average number of generations taken by the GA to find one graph with the MCP. It also indicates the average number of graphs that were actually evaluated by the GA.

Population Size	**Number of Generations**	**Number of Graphs Evaluated**
64	86	5504
128	64	8192

Table 4.5: Average number of generations to find a graph with the MCP and actual number of graphs evaluated (Size 22, Mutation = 0.05)

When a population size is set to 64, the GA finds the graph with the MCP in 86 generations on average. The actual number of graphs computed is 64 * 86, which is equal to 5504. This is a remarkable improvement on the exhaustive search, which would on average compute 524,288 graphs to find a graph with the MCP. For the larger population size of 128, the GA solves the problem in only 64 generations. In this instance, however, the actual number of graphs computed is 8192. This figure, although slightly higher than when the population size is set to 64, is still significantly better than computing 524,288 graphs with the exhaustive search.

In Table 4.5, it is noticed that the GA computes a relatively small number of graphs when the population size is 64. Table 4.6 shows the success upon using different population sizes for the mutation rate of 0.05. With the lower population size, the GA solves the problem 88% of the time, while evaluating only 5504 graphs on average. The population size of 128 is clearly better in finding a graph with the MCP as it has a 100% success rate. Although, fewer graphs are computed when the smaller population is used, the GA solves the problem more consistently using the bigger population size.

Population Size	**Success Rate**
64	0.88
128	1.0

Table 4.6: Probability of finding one graph with MCP (Size 22, Mutation = 0.05)

All the results shown are for size 22. Similar results were obtained for all the sizes that were tested in this experiment. Also, as the size of the graph increases, the performance of the GA is more effective because it is more successful in reducing the number of graphs computed.

4.5 Maximal Cliques Found

The *partner* of a Steinhaus graph is the graph with the generating string $[a_{n-i,\, n}]$ from i=1 to n-1 (Lim and Kim 1). In classifying the cliques in Steinhaus graphs, Daekeun Lim and Jin Hwan Kim identified the four graphs that have maximal clique size with $n \geq 25$ where $n = 3k + 1$. These graphs are represented by the following generating strings: (*a*) $0(101)^k$, (*b*) $0(011)^k$, (*c*) $0(100)^{k-1}(101)$, and (*d*) $0(010)^{k-1}(011)$. The partner of the graph with generating string (*a*) is the graph with generating string (*b*). Similarly, the graph with generating string (*c*) and the graph with the generating string (*d*) are partners (Lim and Kim 12). These four generating strings also hold for the smaller sizes of Steinhaus graphs that were tested in this research. This experiment attempted to find out which of these graphs were found by the GA and how often.

4.5.1 Results

The data in Table 4.7 shows how often the graphs with each generating string were found. These results were based on the first graph found with the MCP.

Generating Strings	Times Found (%)
$0(011)^k$	39
$0(101)^k$	31
$0(100)^{k-1}(101)$	18
$0(010)^{k-1}(011)$	12

Table 4.7: Finding graphs with the MCP (Size 22)

The partners, (*a*) and (*b*), are found first more often than the other pair of partners. One of these two graphs is usually found first between 70% and 75% of all runs of the GA. The reason for this occurrence is unknown.

These observations hold for all sizes considered in this investigation. In fact, for the other sizes, the percentage of runs these graphs were found differs from the figures in Table 4.7 by a very small amount.

Chapter 5: Conclusions

This research suggests that GAs have to be greatly customized in order to apply it to the search for Steinhaus graphs with the MCP. The results described in Chapter 4 provide the parameters to be used in order to optimize GAs for this purpose. Overlapping populations and certain rates of mutation were found to improve the success of GA in solving the problem. Crossover technique did not affect the performance of the GA.

The information from this study lays the groundwork for a variety of areas for further investigation. The possibility of computing fewer graphs can be explored. During each generation, the fitness function calculates the fitness of each chromosome. Since many of these chromosomes are retained for the next generation in the steady state GA, computing their fitness again is unnecessary. If possible, chromosomes could be labeled when their fitness is computed for the first time. Thus, in future generations, the GA would not need to compute the fitness of that particular chromosome again. This would considerably reduce the number of graphs being computed by the GA and would make it more effective. It is also not known why certain graphs with the MCP are found more often. More investigation is required on this subject.

Appendix A

n	1	2	3	4	5	6	7	8	9	10
1	1									
2	1	1								
3	1	3								
4	1	5	2							
5	1	8	7							
6	1	9	22							
7	1	12	47	4						
8	1	14	82	31						
9	1	18	131	106						
10	1	18	200	287	6					
11	1	20	277	693	33					
12	1	22	327	1558	140					
13	1	26	398	3132	535	4				
14	1	27	467	5830	1848	19				
15	1	30	552	9908	5804	89				
16	1	33	642	15304	16407	377	4			
17	1	38	771	22421	40874	1412	19			
18	1	37	899	30739	94223	5103	70			
19	1	38	1060	39831	203417	17558	235	4		
20	1	39	1155	48980	417028	56318	748	19		
21	1	42	1296	58250	818163	168444	2311	69		
22	1	43	1357	67234	1543595	477166	7554	198	4	
23	1	46	1492	74932	2811962	1281583	23741	528	19	
24	1	49	1614	80966	4940779	3288919	74852	1368	60	
25	1	54	1797	87772	8349480	8097792	236456	3699	161	4
26	1	54	1907	95681	13538121	19185907	722476	9861	405	19

k

The entry in the (*n*,*k*) position is the number of Steinhaus graphs with *n* vertices that have clique number *k*.

References

Beasley, David, David Bull, and Ralph Martin. *An Overview Genetic Algorithms: Part 1, Fundamentals*. University Computing. 15(2) 58-69. 1993.

Beasley, David, David Bull, and Ralph Martin. *An Overview Genetic Algorithms: Part 2, Research Topics*. University Computing. 15(4) 170-181. 1993.

Cormen, Thomas, Charles Leiserson, and Ronald Rivest. *Introduction to Algorithms*. New York: McGraw-Hill, 1991.

Dymacek, Wayne M., Matthew Koerlin, and Tom Whaley. *A Survey of Steinhaus Graphs*. To appear in the Proceedings of the Eighth International Conference on Graph Theory, Combinatorics, Algorithms, and Applications (Kalamazoo, Michigan, 1996).

Goldberg, David E. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Reading, Mass.: Addison-Wesley, 1989.

Lim, Daekeun, and Jin Hwan Kim. *Cliques in Steinhaus Graphs*. Pre-print.

Mitchell, Melanie. *An Introduction to Genetic Algorithms.* Cambridge, Mass.: MIT Press, 1996.

Michalewicz, Zbigniew. *Genetic Algorithms + Data Structures = Evolution Programs*. New York : Springer-Verlag, 1994.