A VISUAL GENETIC ALGORITHM TOOL

CHIN WEN PING

DISSERTATION SUBMITTED IN FULFILMENT
OF THE REQUIREMENTS
FOR THE DEGREE OF MASTER OF SOFTWARE ENGINEERING

FACULTY OF COMPUTER SCIENCE
AND
INFORMATION TECHNOLOGY
UNIVERSITY OF MALAYA

MAY 2006
Abstract

This dissertation addresses on developing a Visual Genetic Algorithm Tool for an optimization problem. Genetic algorithms are a part of evolutionary algorithms and are based on nature’s evolutionary process. Solutions are able to reproduce among themselves to create fitter individuals contributing towards the optimal solution. The attractiveness of Genetic Algorithm lies in the fact that it is capable of searching good solutions from a large search space.

The optimization problem that this dissertation will visually address is the Traveling Salesman Problem, defined as a NP-Complete class of problems. Many algorithms have been designed to solve the problem and now this dissertation considers genetic algorithm as a method to solve the Traveling Salesman Problem visually. The class of algorithms used traditionally to solve the Traveling Salesman Problem is defined as approximation algorithms.

The visual tool is able to clearly show the evolution of the chromosomes and the effects of genetic algorithm towards obtaining the optimal solution. In addition, the visual tool is able to graphically visualize the improvements made to the tour route optimization and a graph that documents the tour distance throughout the optimization process. The tool is then used solve the Traveling Salesman Problem and the results documented ensuring that the tool was proper designed and implemented.
Acknowledgements

I would like to take an opportunity to thank those who have been actively involved in making this dissertation a success. They are Associate Professor Raja Ainon, the staffs of Faculty of Computer Science and Information Technology of University of Malaya, my family and friends. I appreciate the comments and guidance throughout the dissertation period, helping me improve.
Table of Contents

Abstract .............................................................................................................. ii
Acknowledgements .......................................................................................... iii
Table of Contents ............................................................................................ iv
List of Figures ................................................................................................... vi
List of Tables ...................................................................................................... vi
List of Tables ...................................................................................................... viii

Chapter 1. Introduction .................................................................................. 1
  1.1 Research Objectives ................................................................................. 2
  1.2 Research Scope ........................................................................................ 2
  1.3 Research Methodology ............................................................................ 3
  1.4 Results Summary ..................................................................................... 4
  1.5 Outline of Dissertation ........................................................................... 4

Chapter 2. Genetic Algorithms ..................................................................... 6
  2.1 How do Genetic Algorithms Work? ......................................................... 7
  2.2 The Genetic Algorithm Operators ............................................................ 10
    2.2.1 Population Size .................................................................................. 10
    2.2.2 Crossover .......................................................................................... 11
      2.2.2.1 Single Point Crossover ................................................................. 12
      2.2.2.2 Multipoint Crossover ................................................................. 12
    2.2.3 Mutation ............................................................................................ 13
    2.2.4 Selection Mechanism ........................................................................ 14
      2.3.4.1 Roulette Wheel Selection ......................................................... 14
      2.3.4.2 Uniform Stochastic Selection ................................................. 15
      2.3.4.3 Tournament Selection ............................................................. 16
    2.3.5 Elitism ............................................................................................... 17
  2.4 Data Encoding .......................................................................................... 17
  2.5 Applications of Genetic Algorithm ........................................................... 18
  2.6 Experiments with Genetic Algorithm – Rastrigin’s function ................. 19
    2.6.1 Results and Observations ................................................................ 21
      2.6.1.1 Population Size ......................................................................... 21
      2.6.1.2 Mutation Rate .......................................................................... 24
  2.7 Experiments with Genetic Algorithm – Rosenbrock’s function .......... 26
    2.7.1 Results and Observations ................................................................ 28
  2.8 Conclusion of the Experiments with GA ................................................. 31

Chapter 3. The Traveling Salesman Problem .............................................. 33
  3.1 Approximation Algorithms ..................................................................... 34
  3.2 Types of TSP ........................................................................................... 34
  3.3 Hamilton Cycle ........................................................................................ 36
  3.4 The Euclidean Distance .......................................................................... 37
  3.5 Solving the TSP ....................................................................................... 38
    3.5.1 The Nearest-Neighbor Heuristic .................................................. 40
    3.5.2 Insertion Heuristics ......................................................................... 43
List of Figures

Figure 2.1: Single Point Crossover .......................................................... 12
Figure 2.2: Multipoint Crossover (5 points) ............................................. 12
Figure 2.3: Binary Mutation ................................................................. 13
Figure 2.4: The Roulette Wheel Selection .............................................. 15
Figure 2.5: An Example of Stochastic Universal Selection .................... 16
Figure 2.6: Rastrigin's Function .............................................................. 20
Figure 2.7: The Local Minima Points of Rastrigin's Function ................... 20
Figure 2.8: The Output Based on a Population Size of 20 ....................... 22
Figure 2.9: The Output Based on a Population Size of 30 ....................... 23
Figure 2.10: The Output Based on a Population Size of 50 ...................... 24
Figure 2.11: The Output Based on a Mutation Rate of 0.5 ....................... 25
Figure 2.12: The Output Based on a Mutation Rate of 0.7 ......... 26
Figure 2.13: Rosenbrock's Function ...................................................... 27
Figure 2.14: The Local Minima Point in Rosenbrock's Function .......... 27
Figure 2.15: The Output Based on Roulette Wheel Selection ............... 29
Figure 2.16: The Output Based on Random Stochastic Selection ......... 29
Figure 2.17: The Output Based on Tournament Selection .................... 30

Figure 3.1: The Traveling Salesman Problem ........................................... 33
Figure 3.2: The Hamilton Cycle ............................................................ 36
Figure 3.3: The Tour For a 5 City TSP .................................................... 37
Figure 3.4: The Euclidean Distance in a 2 Dimensional TSP ................. 38
Figure 3.5: The Nearest Neighbor Heuristic to Solve a TSP .................. 42
Figure 3.6: Cheapest Insertion .............................................................. 44
Figure 3.7: Farthest Insertion ................................................................. 45

Figure 4.1: The Limitation of Binary Encoding Scheme ......................... 48
Figure 4.2: The Limitation of the Traditional Crossover Operator ......... 50
Figure 4.3: The Random Keys Representation ....................................... 51
Figure 4.4: The 2-opt Method ............................................................... 54
Figure 4.5: The Partially Mapped Crossover Example .......................... 56
Figure 4.6: Ordered Crossover for a 9 City TSP ..................................... 57
Figure 4.7: An Example of Cycle Crossover ........................................... 58
Figure 4.8: The Greedy Crossover in Application .................................. 60
Figure 4.9: The Inversion Mutation ....................................................... 61
Figure 4.10: The Insertion Mutation ...................................................... 62
Figure 4.11: The Displacement Mutation ............................................. 62
Figure 4.12 Screen Capture of Matlab’s Genetic Algorithm Tool .......... 63

Figure 5.1: The Fitness Evaluation Based on Pixels ............................. 67
Figure 5.2: The Greedy Crossover for a Sample TSP ............................. 69
Figure 5.3: The Inversion Mutation ....................................................... 69
Figure 5.4: The User Interface of The Visual Genetic Algorithm Tool for TSP .............................. 71
Figure 5.5: The Randomly Generated Cities in The Map ...................... 73
Figure 5.6: Pipes and Filters Architecture ............................................ 75
Figure 5.7: The Pipes and Filters Architecture for the Visual Genetic Algorithm Tool for TSP

Figure 5.8: The Collaboration Diagram for the Visual Genetic Algorithm Tool

Figure 5.9: The Class Diagram for the Visual Genetic Algorithm Tool

Figure 6.1: The 20 City TSP

Figure 6.2: The Tour of the 20 City TSP at the 14th Generation

Figure 6.3: Shows the Tour Distance Reducing per Generation

Figure 6.4: The Tour of the 20 City TSP at Generation Number 107

Figure 6.5: The Tour of the 20 City TSP at Generation Number 185

Figure 6.6: The Optimal Tour of the 20 City TSP with Tour Distance of 1254 Pixels

Figure 6.7: Tour Route With Nearest Neighbor Heuristics

Figure 6.8: The 50 City TSP

Figure 6.9: The Tour of the 50 City TSP at Generation Number 54

Figure 6.10: The Optimal Tour of the 50 City TSP with a Tour Distance of 2751 Pixels

Figure 6.11: Graph of 20 City TSP

Figure 6.12: Graph of 50 City TSP
List of Tables

Table 2.1: The Results of Using GA for Rosenbrock's Function with Varying Selection Mechanisms. ................................................................. 30

Table 6.1: The Results for 20 City TSP With Varying Population Size. ......................... 97
Table 6.2: The Results for 50 City TSP With Varying Population Size. ......................... 99
Chapter 1. Introduction

A genetic algorithm (GA) is a heuristic used to find approximate solutions to difficult-to-solve problems through application of the principles of evolutionary biology to computer science. Genetic algorithms use biologically derived techniques such as inheritance, mutation, natural selection, and reproduction. Genetic algorithms are a particular class of evolutionary algorithms akin to nature’s evolution.

Problems, which appear to be particularly appropriate for solution by genetic algorithms, include timetabling and scheduling problems, and many scheduling software packages are based on genetic algorithms. Genetic algorithms are often applied as an approach to solve global optimization problems. As a general rule of thumb genetic algorithms might be useful in problem domains that have a complex fitness landscape as recombination is designed to move the population away from local minima that a traditional hill-climbing algorithm might get stuck in. This property of evolutionary algorithms makes it ideal to optimize complex neural networks that might have many local minima.

In addition, custom computer applications began to emerge in a wide variety of fields, and these algorithms are now used by all companies big or small to solve difficult scheduling, data fitting, trend spotting, budgeting and virtually any other type of combinatorial optimization. For example, the genetic algorithm might be used for scheduling problems in the airlines industry. The appeal of using Genetic Algorithms for complex problem lies in the fact that, the algorithm is able to adapt to any problem providing a platform where the system itself is learning and evolving to produce an optimized solution to a problem.
1.1 Research Objectives

The objective of the dissertation is to develop a Visual Genetic Algorithm Tool for a Traveling Salesman Problem. The tool will be able to visually show an improved solution to the problem in each generation. In a Traveling Salesman Problem, the objective is to find the shortest tour that the salesman must make, making it an optimization problem. Genetic algorithm is used to optimize the tour route for the salesman to make, ensuring that the shortest route possible is taken. The Visual Genetic Algorithm Tool must depict the optimization of the Traveling Salesman Problem graphically via a map of the tour. The routes will reduce in distance in each generation and is directly translated to reduction in the length of the line denoting the path traveled by the salesman in the map. This represents the visual effect of genetic algorithm in optimizing the tour route.

1.2 Research Scope

The focus of this research is to develop a Visual Genetic Algorithm Tool for a Traveling Salesman Problem. A study is conducted on genetic algorithm in general covering techniques and methodologies to solve complex optimization problems. A study on the Traveling Salesman Problem (TSP) and the methods used to solve the problem are conducted. Research is conducted on Visual Genetic Algorithm Tools and the features of the tools. Then the focus of the research is on using genetic algorithm to solve TSP depicting the effects of genetic algorithm visually as it solves the TSP.
1.3 Research Methodology

A quantitative research approach is generally used in this dissertation, as the dissertation’s objective is to develop a Visual Genetic Algorithm Tool to determine the relationship between two independent variables. A quantitative research is research, in which the data is usually gathered using a more structured research tool such as a program and which involves experimentation. The two independent variables in discussion for this dissertation is the distance traveled by the salesman and time. Time in this instance is the generation of genetic algorithm; the generation is incremental in nature much similar to time. The results of the dissertation are also measurable based on recorded results.

To start the research, a through literature review on genetic algorithms and the Traveling Salesman Problem are conducted. Library research is first conducted to collect and study the related materials pertaining to the main topics of interest. To further cement the concepts of genetic algorithm reviewed, experiments are conducted to tests the properties of genetic algorithm on optimization problems. The focus on the experiments is to have a firmer grip on the varying effects and effectiveness of genetic algorithm has on solving problems related to optimization problems with large search space.

Visual programming techniques and methodologies are reviewed to determine which programming language is best suited to implement the visual tool. The Visual Genetic Algorithm Tool for Traveling Salesman Problem is then designed based on reviewed techniques and implemented taking into consideration the best-known methods.
Experiments are then conducted with the completed tool and the results along with the observations made of the visual tool are recorded for interpretation and analysis. The result recorded is the distance traveled by the salesman covering all cities in the problem against the generation number.

Finally, the consistency of the tool is also measured to ensure that there is no error in the tool. Since this dissertation involves optimizing a NP-complete problem, the consistency of the tool can be checked during execution with the distance traveled reducing per generation as genetic algorithms optimize the tour distance. Errors can be checked during program execution in avoiding illegal tours, since the optimal solution cannot be ascertained in approximation algorithms in NP-complete problems.

1.4 Results Summary

Genetic Algorithm used to solve the Traveling Salesman Problem must show improvement on the tour distance over each generation. The improvement is defined as the reduction in tour distance. The tour distance must reduce from generation to generation to reflect a success for optimization of the problem. Since this is a NP-complete problem, finding the genetic algorithm will be used to find a good route through each of the cities in the tour. The Visual Genetic Algorithm Tool will depict the reduction in tour routes via a map of the tour.

1.5 Outline of Dissertation
The remainder of this dissertation consists of an introduction to Genetic Algorithms in Chapter 2 and the focus of the optimization problem of Traveling Salesman Problem in Chapter 3. Chapter 2 is an overview of Genetic Algorithms and its applications. Chapter 3 gives a similar overview to the Traveling Salesman Problem providing insights on traditional methods used to solve the Traveling Salesman Problem. Chapter 4 addresses Genetic Algorithm and how it is typically used to solve the Traveling Salesman Problem.

Chapter 5 describes how the design and implementation of the Visual Genetic Algorithm Tool tailored specifically to solve the Traveling Salesman Problem. Chapter 6 provides the results and observations of the Visual Genetic Algorithm Tool used to solve the Traveling Salesman Problem. And finally in Chapter 7, the conclusion is made and gives some direction on future work.
Chapter 2. Genetic Algorithms

Genetic Algorithm (GA) belongs to elementary stochastic optimization algorithms inspired by evolution. GA was developed by John Holland in the 1970s (Holland, 1975) and the idea is based on Darwin’s theory of evolution.

The first person to write about the basis of natural selection was Charles Darwin in his book titled The Origin of Species by Means of Natural Selection. Darwin’s theory of evolution is based on principals of natural selection, involving recombination and mutation, which provided the survival of the best-adapted individuals in the population. Reproduction of the two very fit individuals produces offspring with high probability of successful adaptation and survival. Only reproduction operation is not sufficient to effectively create well adaptable individuals with new characteristics. It is necessary to include mutations to enrich the population with new genetic material.

Biological evolution is progressive change of genetic information (genotype) in the population after certain numbers of generations. Another important term is fitness, which in biology is defined as relative ability of survival and genotype reproduction in certain environment.

The term biological individual can be replaced by the term chromosome, which is represented by a string of tokens containing information about the individual (genotype). Next important term is population of chromosomes that enter the reproduction process with a probability proportional to its fitness. However, mutations play important role by
enabling new genetic information to be brought into the population, thus resulting in increase in fitness of chromosomes. In the next generation, chromosomes with lower fitness from current generation are being replaced with ones that have higher fitness. Repeating this reproductive cycle on the population of chromosomes will after certain time with high probability lead to emergence of chromosomes with new characteristics that will positively influence their fitness and replace chromosomes without new characteristics.

Nature has found a way to evolve effective and stable creatures. When new species arrive, they will automatically be compared with earlier species in sense of gathering food, defense and breed. Species with great adaptability to the surrounding environment will get high fitness and probability to survive while species with low suitability will on the other hand not have same probability of survival. Sudden change in the environment will likely cause a wipe out such low adapting species. Surviving species will continue breeding and produce more similar species. With some small probability, mutation will appear and change the species behavior. Most of the time these changes will be for the worse but sometimes it will also be for the better and create a species with greater fitness.

2.1 How do Genetic Algorithms Work?

The basic principles of GA was developed by John Holland (Holland, 1975). They have since been reviewed and applied the concepts have been applied on a wider range (Mitchell, 1996), (Koza, 1992) and (Whitley, 2001). The chapter is based on literature by the four authors mentioned beforehand.
The basic concept of GA is based on survival of the fittest and it implies that the ‘fitter’ individuals are more likely to survive and have a greater chance of passing their ‘good’ genetic features to the next generation.

In nature, the blueprint of individuals is contained in their DNA. The DNA can be then represented as a string of genes, with each gene or combination of genes representing a particular feature. Reproduction is the ‘crossover’ of two DNA strings to produce a new blueprint that has genes from both parents. Mutation can also occur when a particular gene is not an exact copy of either parent.

In GA terms, a candidate solution is often referred to as a chromosome or string, which is a sequence of encoded numbers. This is commonly referred to as a bit string if the numbers are binary encoded. The process involved in GA optimization problem is based on that of natural evolution and generally works as follows:

1. Randomly generate an initial population of potential solutions
2. Evaluate the suitability or ‘fitness’ of each solution
3. Select two solutions in favor of fitness
4. Crossover the solution at a random point on the string to produce two new solutions
5. Mutate the new solutions based on a mutation probability
6. Go to step 2 and repeat

There are many stopping criteria for the GA process. The most popularly used is when the GA process stops when the limit generation number has been reached or when a certain fitness level has been reached which means the problem has been optimized.
A simple pseudo-code of the algorithm is as follows:

Begin GA

\[ g := 0 \ {\text{ generation counter }} \]

Initialize population \( P(g) \)

Evaluate population \( P(g) \) \{ i.e., compute fitness values \}

while not optimized do

\[ g := g + 1 \]

Select \( P(g) \) from \( P(g-1) \)

Crossover \( P(g) \)

Mutate \( P(g) \)

Evaluate \( P(g) \)

End while

End GA

The appeal of GA comes from their simplicity and elegance as robust search algorithms as well as from their power to discover good solutions rapidly for difficult high-dimensional problems such as the Traveling Sales Person problem, which is in the NP-complete domain.

GA is useful and efficient when:

- The search space is large, complex or poorly understood
• Domain knowledge is scarce or expert knowledge is difficult to encode to narrow the search space
• No mathematical analysis is available
• Traditional search methods fail

The advantage of the GA approach is the ease with which it can handle arbitrary kinds of constraints and objectives; all such things can be handled as weighted components of the fitness function, making it easy to adapt the GA to the particular requirements of a very wide range of possible overall objectives.

2.2 The Genetic Algorithm Operators

Selection, crossover and mutation are the basic operators involved in GA. The operators are the basis of the GA. The section serves to show the effects of varying the GA operators in a search.

2.2.1 Population Size

The population size is the number of candidate solutions in any one generation. In natural evolution the total population size is governed by what is sustainable by the environment and similarly in GA the larger the population size, the more computationally intensive is the search.
In nature, the bigger the gene pool the more diverse is the genetic properties that make up the population with many individuals each with their own set of characteristics that enable them to survive. One simple advantage of the diversity is that there will be no dominant genes that, for instance, may be susceptible to a particular disease and results in the elimination of the whole species.

If the population size is small, the dominant or strong individual with a high fitness value quickly dominates the gene pool and diversity if reduced. The outcome is that individuals (local optima) are quickly created but the dominance of particular genes restricts the search space reducing the chances that global optimum is found. In (Patton, Dexter and Punch III, 1997), the effects of increasing the population size for reaching an effective solution is shown. The effect might be primarily due to effects of heavy crossover of the enlarged population size.

As new solutions are generated, it is common to keep the population size constant by eliminating individuals by the laws of nature through death, although this does not have to be the case. The advantage of GA is that good individuals do not have to die and can be retained for indefinite reproduction. This retention scheme of fit individuals is known as elitism.

2.2.2 Crossover

Along with mutation, crossover is an operator to create new solutions from the parents.
2.2.2.1 Single Point Crossover

One crossover point is randomly selected. The binary string from beginning of parent A to its crossover point is copied to the new offspring on the same positions. The rest (from the same crossover point of parent A to its tail) is copied to the new offspring on the same positions. Figure 3.1 shows the single point crossover.

![Figure 2.1: Single Point Crossover.](image)

2.2.2.2 Multipoint Crossover

Two crossover points are selected. The string from beginning of parent 1 to its first crossover point and the binary string from its second crossover point to its end are copied to the new offspring. The rest (the first crossover point of parent 2 to its second crossover point) is copied to the new offspring in the same fashion. Figure 3.2 shows the multipoint (5 points) crossover.

![Figure 2.2: Multipoint Crossover (5 points).](image)
2.2.3 Mutation

In natural evolution, mutation is a random process where one allele of a gene is replaced by another to produce a new genetic structure.

In GA, mutation is randomly applied with low probability, typically in the range 0.001 and 0.01, and modifies elements in the chromosomes. Usually considered as a background operator, the role of mutation is often seen as providing a guarantee that the probability of searching any given string will never be zero and acting as a safety net to recover good genetic material that may be lost through the action of selection and crossover (Goldberg, 1989). In addition, its purpose is to maintain diversity within the population and inhibit premature convergence.

![Binary Mutation](image)

Each bit of the string has the potential to mutate, based on a mutation probability ($P_m$). In binary encoding, mutation involved flipping of a bit from 0 to 1 or vice versa. Figure 3.3 shows the binary mutation in action where the 3rd bit is inverted from 0 to 1.
2.2.4 Selection Mechanism

The selection procedure is for choosing individuals (parents) on which to perform reproduction operation in order to create new solutions. The idea is that the individuals with high fitness values are more prominent in the process, with the hope that the child they create will have an even higher fitness value.

2.3.4.1 Roulette Wheel Selection

A popular selection mechanism is the roulette wheel selection, which descends from the gambling game with the same name. It places the chromosomes into a roulette wheel with the same portion as their fitness, like a section in a pie. The wheel starts rolling and when it stops the chromosome closest to a defined position will be selected. As chromosomes with high fitness will cover larger volume of the wheel than chromosomes with lower fitness, there are higher probabilities that high fitness chromosomes will be selected. This is depicted in Figure 3.4 where numbers 1 to 5 are labels for a different chromosome. The probability that a chromosome is selected is as follows:

\[ p_{\text{select}}(x) = \frac{f(x)}{\sum f} \]

Where \( x \) denotes a chromosome and \( f(x) \) is the fitness of chromosome \( x \).
Unfortunately, there are some drawbacks with this method, such as high probability that only elite genotypes will be chosen, if there is a large gap between the best and bad chromosomes.

2.3.4.2 Uniform Stochastic Selection

Stochastic universal selection provides zero bias and minimum spread. The individuals are mapped to contiguous segments of a line, such that each individual's segment is equal in size to its fitness exactly as in roulette-wheel selection. Here equally spaced pointers are placed over the line, as many as there are individuals to be selected.
Figure 2.5: An Example of Stochastic Universal Selection.

Figure 3.5 illustrates an example where for 6 individuals to be selected, the distance of the pointer of selection is 1/pointer or 1/6 (0.167). The individuals of 1, 2, 3, 4, 6, and 8 are selected.

The outcome of a certain run of the selection scheme is as close as possible to the expected behavior, i.e. the mean variation is minimal. Even though it is not clear whether there are any performance advantages in using uniform stochastic selection, it makes the run of a selection method more predictable (Bickle and Thiele, 1995).

2.3.4.3 Tournament Selection

Tournament selection involves picking a number of strings at random from the population to form a "tournament" pool. The tournament selection operator can be rationalized simply as a selection mechanism that uses roulette selection N times to produce a tournament subset of chromosomes. At each of the N rounds of selection in a tournament, the fitness of the competitors is compared where the fitter individual will progress to the next round. This will and eventually produces a single individual with the highest fitness and the individual will be selected for reproduction.
The tournament selection focuses on selection pressure towards individuals of higher fitness in the population (Legg, Hutter, Kumar, 2004). This might introduce the effect of elitism where the rationale lies that individuals with higher fitness are most likely to produce offspring.

### 2.3.5 Elitism

Elitism is a mechanism used in GA, which ensures that the chromosomes of the most highly fit member(s) of the population are passed on to the next generation without being altered by other genetic operators. Using elitism ensures that the minimum fitness of the population can never reduce from one generation to the next. Elitism usually brings about a more rapid convergence of the population. In some applications elitism improves the chances of locating a global minima or maxima, while in others it reduces it.

In (Chakraborty and Chaudhuri, 2003), the introduction of elitism or by keeping the best string in the population allows the convergence to the global optimal solution starting from any arbitrary initial population.

### 2.4 Data Encoding

In order to implement a GA, a set of parameters is sought that will give the best solution to a particular problem. These parameters must be encoded into a string so that the genetic operators can be applied. In early pioneering work (Holland, 1975), binary encoding is most commonly used and gives rise to today’s popularity of the binary encoding. The main
argument for this is that, binary representations decomposes the problems into the largest number of possible building blocks and that GA works by processing these building blocks (Whitley, 2001).

The binary encoding is the most common method for representation of phenotype in GA. The reason for its usefulness is due to computation factors like calculation speed and ease of implementation. The encoding scheme consists of strings of bits 1 or 0 and is a direct translation to the binary string representing an equal value. The 1s and 0s together represent a number.

In choosing an encoding scheme, the nature of the problem will play a major role. Techniques such as shifting, which is a mechanism, which dynamically switches between Gray code representation in order to escape a local optima (Barbulescu, Watson and Whitley, 2000).

2.5 Applications of Genetic Algorithm

The possible applications of genetic algorithms are immense. Any problem that has a large search domain could be suitable tackled by GA. The architecture of systems that implements GA is more able to adapt to a wide range of problems. A GA functions by generating a large set of possible solutions to a given problem. It then evaluates each of those solutions, and decides on the fitness of the individuals. These solutions then reproduce new solutions and similar concept applies for the next generation of solutions. This way you evolve your search space scope to a point where you can find the solution.
With the above explained intrinsic properties of GA, the applications of GA are found in the field of problem optimization that includes timetabling, scheduling, and adapting models of complex systems to fit new data. In addition, there is immense potential for GA to be applied in the field of Artificial Intelligence such as Evolutionary Algorithms and Evolutionary Programming.

2.6 Experiments with Genetic Algorithm – Rastrigin’s function

To objective of this section is to evaluate the effects and importance of the various parameters in GA. The GA is used to search for the global minimum of Rastrigin’s function. Rastrigin’s function is as follows:

\[ y = 20 + x_1^2 + x_2^2 - 10(\cos 2\pi x_1 + \cos 2\pi x_2) \]

The global minimum is when \( x_1 = 0 \) and \( x_2 = 0 \) and the output is 0 (\( y = 0 \)). Figure 3.6 shows the various local minima associated with the function.
The following contour plot of Rastrigin’s function in Figure 3.7 shows the alternating local maxima and local minima with the global minimum at coordinates (0,0).

Figure 2.6: Rastrigin's Function.

Figure 2.7: The Local Minima Points of Rastrigin's Function.
The experiment is conducted with Matlab’s Genetic Algorithm and Direct Search toolbox. The y-axis of all the graphs is the fitness values in a logarithmic scale and the x-axis is the generations of the GA.

2.6.1 Results and Observations

The results of varying the GA parameters for the Rastrigin’s function are shown in the figures below. All comments and discussion related to each figure are included in the figure below.

The points colored in black at the bottom of the graph denote the best fitness values, while the points above colored in blue them denote the averages of the fitness values in each generation. The graph displays the best and mean values in the current generation numerically at the top.

2.6.1.1 Population Size

Population size = 20, Stochastic Selection, Mutation Rate = 1.0

The mutation rate of 1.0 means that the chromosomes are always mutating in each generation after crossover occurs. The next sub-section will evaluate the effects of different mutation rates on the population fitness.
The best value obtained from the search space is 0.0075257 and the mean result is 0.080284.

![Graph showing fitness values over generations](image)

Figure 2.8: The Output Based on a Population Size of 20.

**Population size = 30, Stochastic Selection, Mutation Rate = 1.0**

The best value obtained from the search space is 0.00027784 and the mean result is 0.041216. The results seem to be better if compared to the results of population size 20. By increasing the numbers of individuals, the diversity in the gene pool increases and the solution is able to produce a fitter individual.

The fitness of individuals had improved and it is reflected in the improvement of the mean value of the solution.
Figure 2.9: The Output Based on a Population Size of 30.

**Population size = 50, Stochastic Selection, Mutation Rate = 1.0**

The best value obtained from the search space is 0.00071305 and the mean result is 0.01255. Although the best result is worse than the search space of the population size 30, the mean is much better approaching the value of 0. This is due to the probabilistic nature of stochastic selection.

The increase in population size brings better convergence of the solution and this is due to the fact that analysis shows that this is primarily due to heavy reliance which crossover places on the initial population (Patton, Dexter and Punch III, 1997).
2.6.1.2 Mutation Rate

Population size = 20, Stochastic Selection, Mutation Rate = 0.5

The best value obtained from the search space is 0.99534 and the mean result is 1.004.
Figure 2.11: The Output Based on a Mutation Rate of 0.5.

**Population size = 20, Stochastic Selection, Mutation Rate = 0.7**

The best value obtained from the search space is 0.00062419 and the mean result is 0.79905. Higher probability of mutation leads to quicker convergence towards target environment (Magnus, 2004).

An experiment with population size shows that the results showed that for simpler problem with small problem size uniform crossover performs better. While on more complex problems with larger population size the 2-point crossover gave better result. Small population converges fast so there is a need for more disruptive crossover to facilitate
mixing. Large population provides with accurate information through sufficient sampling and hence a less disruptive crossover performs better (De Jong and Spears, 1990).

![Figure 2.12: The Output Based on a Mutation Rate of 0.7.](image)

### 2.7 Experiments with Genetic Algorithm – Rosenbrock’s function

This section focuses on the effects of various selection mechanisms and their effectiveness in optimization problems. GA will be used to optimize Rosenbrock’s function. The objective is to find the minimum point. Rosenbrock’s function is also known as Dejong’s second function. The Rosenbrock function is defined as:

\[
f(x,y) = 100(x-y^2)^2 + (1-x)^2
\]

\[f(x) \text{ is minimum when } x \text{ and } y \text{ is } 1 \text{ and } f(x) = 0\]
The Rosenbrock function is a common test in optimization problems because it has high degree of non-linearity and converges slowly if you use the steepest descent type methods. Figure 2.13 shows the Rosenbrock function and Figure 2.14 shows the global minimum associated with the function against the many local minimums.

Figure 2.13: Rosenbrock's Function.

Figure 2.14: The Local Minima Point in Rosenbrock's Function.
The experiment will focus on using various selection mechanisms in the GA optimization process for comparison on the effectiveness in solving the Rosenbrock optimization problem. The 3 selection mechanisms considered are: - Roulette Wheel Selection, Random Stochastic Selection and Tournament Selection. All the other parameters in the GA remain constant throughout the 3 experiments. The population size is 20 and the GA is run for 100 generations with no other stopping criteria.

The experiment is conducted with Matlab’s Genetic Algorithm and Direct Search toolbox. The points colored in blue denote the averages of the fitness values in each generation while the points colored in black denotes the best fitness value in each generation. The y-axis of all the graphs is the fitness values in a logarithmic scale and the x-axis is the generations of the GA.

2.7.1 Results and Observations

Roulette Wheel Selection
Random Stochastic Selection

Figure 2.15: The Output Based on Roulette Wheel Selection.

Figure 2.16: The Output Based on Random Stochastic Selection.
**Tournament Selection**

![Figure 2.17: The Output Based on Tournament Selection.]

The best values and mean values for the 3 selection mechanisms are as in Table 2.1.

Table 2.1: The Results of Using GA for Rosenbrock's Function with Varying Selection Mechanisms.

<table>
<thead>
<tr>
<th>Selection Mechanism</th>
<th>Best Value</th>
<th>Mean Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Roulette Wheel Selection</td>
<td>0.00054666</td>
<td>0.039485</td>
</tr>
<tr>
<td>Random Stochastic Selection</td>
<td>0.0020681</td>
<td>0.039892</td>
</tr>
<tr>
<td>Tournament Selection</td>
<td>0.00092307</td>
<td>0.014105</td>
</tr>
</tbody>
</table>
It can be concluded in Table 2.1 that the Roulette Wheel Selection provides an individual with the best fitness. This might be associated with the elitism factor apparently existing in Roulette Wheel Selection.

The best fitness of the population is provided by using the Tournament Selection mechanism. The Tournament selection is actually a series of individuals selected for comparison with the rest and the highest fitness individuals will be selected based on the selection. This might lead to a population with an overall better fitness than the rest and the selection process is more thorough.

Random Stochastic Selection also provides a good approximated answer to the problem but lags behind Roulette Wheel and Tournament selection due to the random selection process of individuals for mating. Random Stochastic Selection ignores the fact that if the individual is fitter, the chances of selection for reproduction are higher (elitism) and the selection is truly random with the selection of the parents for reproduction process.

In short, all the three selection mechanisms prove to be an effective method to solve the Rosenbrock function. Certain selection mechanisms are more effective with certain problems. Elitism might cause premature convergence of the solution and thus certain measures must be taken to minimize the effects. A suggested method is through normalization of the fitness values after certain numbers of generation to avoid selection of only the fittest individual.

2.8 Conclusion of the Experiments with GA
GA serves to be a versatile tool in solving complex optimization problems such as Rastrigin’s function and Rosenbrock’s function. GA proves to be an effective tool for solving NP-Complete problems that it could be shown in 100 generations a very close approximation to the actual answer is obtained. In the Rosenbrock function, by using Random Stochastic Selection method, the output is 0.0020681 against 0. This gives an error of 0.2% where the search space for the solution is infinite. It can be concluded that GA can be effectively and efficiently used to solve optimization problems with certain level of accuracy and is appropriate for solving approximation problems.
Chapter 3. The Traveling Salesman Problem

The Traveling Salesman Problem (TSP) can be defined as a salesman who has to visit every city in his territory exactly once and return to the starting point. Given the time of travel between all cities, how should he plan his itinerary for minimum time taken for the entire tour?

The search space for the TSP is a set of permutations of N cities. Any single permutation of N cities yields a solution – a complete tour of the cities. The optimal solution is a permutation, which yields the minimum cost or time taken for the tour. The size of the search space is N factorial or N!. This is visually depicted in Figure 3.1.

### 3-City Traveling Salesman Problem

The search space is 3! or $3 \times 2 \times 1 = 6$ possible tours

The possible tours of the salesman:
1. A-B-C
2. B-C-A
3. C-A-B
4. A-C-B
5. C-B-A
6. B-A-C

Figure 3.1: The Traveling Salesman Problem.
The Traveling Salesman Problem is a NP-Complete problem; it arises from the large number of cities to travel and the number of possible solutions to the problem.

The fitness of the TSP tour is the sum of the Euclidean distance between every city in the tour.

3.1 Approximation Algorithms

Approximation algorithms are algorithms that do not compute optimal solutions; instead, they compute solutions that are "good enough". Often we use approximation algorithms to solve problems that are computationally expensive but are too significant to give up on altogether. The TSP is one example of a problem usually solved using an approximation algorithm due to the complex nature of the problem.

In (Ghana-Hercock, 2003) it is shown that TSP is a complex problem that requires an alternative solution from the traditional solution-searching algorithm.

Such problems are considered to have no solution in real time, although by approximating the solutions to these problems to within specified tolerances approximate solutions may be obtained.

3.2 Types of TSP
There exist many variations of the TSP. The variations exist to test the limits of approximation algorithms to solve NP-Complete problems and due to the complex nature of the problem at hand. The most common forms of TSP according to (Gross and Yellen, 2003a) are:

- **Symmetric TSP (STSP):**
  Given a complete (undirected) graph $K_n$ with weights on the edges, find a Hamiltonian cycle in $K_n$ of minimum (total) weight. This is where the distance between $A$ to $B$ is equal to the distance between $B$ to $A$. The distance is similar in both directions.

- **Asymmetric TSP (ATSP):**
  Given a complete directed graph $\bar{K}_n$ with weights on the arc, find a Hamiltonian cycle in $\bar{K}_n$ of minimum weight. This is the special instance where the distance from $A$ to $B$ is not equal to the distance from $B$ to $A$. An example of a practical application of an asymmetric TSP is route optimization using street-level routing (asymmetric due to one-way streets, slip-roads and motorways).

- **Euclidean TSP:**
  Special case of STSP in which the vertices are points in Euclidean plane and the weight on each edge is the Euclidean distance between its endpoints.

Each variation of TSP presents a new problem definition. In a TSP with 10 cities might have a different solution to a STSP than an ATSP since the distance is different in each
direction for an ATSP. The solution for ATSP might differ for each run due to the
difference in distance in each direction. An Euclidean TSP is also complex in nature with a
10 city TSP having possible solution search space of 10!.

3.3 Hamilton Cycle

The Hamilton Cycle is a famous graph theory used to solve the TSP. The Hamilton Cycle
can be defined as:

*Input: An unweighted graph G.*

*Output: Does there exist a simple tour that visits each vertex of G without repetition?*

In general, a cycle that passes through every vertex in a graph is called a Hamilton Cycle
and a graph with such a cycle is called a Hamiltonian. A Hamilton cycle thus is called a
tour. Figure 3.2 illustrates the concept of a Hamilton Cycle.

![Hamilton Cycle](image)

**Hamilton Cycle: a c e f d b a**

*Figure 3.2: The Hamilton Cycle.*
An example would be a 5 city TSP. Figure 3.3 depicts the possible routes and the Hamilton Cycle associated with the TSP. In this case, the shortest distance of the tour is the Hamilton Cycle.

Figure 3.3: The Tour For a 5 City TSP.

3.4 The Euclidean Distance

The Euclidean distance or Euclidean metric is the "ordinary" distance between the two points that one would measure with a ruler, which in turn can be proven by repeated application of the Pythagorean theorem. By using this formula as distance, Euclidean space becomes a metric space and is quantifiable.

The Euclidean distance for two points \( a = (a_1, \ldots, a_n) \) and \( b = (b_1, \ldots, b_n) \) in Euclidean \( n \)-space is defined as:

\[
d(a, b) := \sqrt{(a_1 - b_1)^2 + (a_2 - b_2)^2 + \cdots + (a_n - b_n)^2} = \sqrt{\sum_{i=1}^{n} (a_i - b_i)^2}
\]
The Euclidean distance for a 3 city TSP is as depicted in Figure 3.4. The similar concept applies for the Euclidean distance for City A to City B and City C to City A.

3-City Traveling Salesman Problem

![Diagram](image)

The Euclidean distance between city B and city C

Figure 3.4: The Euclidean Distance in a 2 Dimensional TSP.

The fitness of the tour is the total Euclidean distance from A to B, B to C and C to A. The shortest Euclidean distance is the Hamilton cycle of the TSP and is the solution to the TSP.

3.5 Solving the TSP

TSP can be solved by graph theory, in a graph, a tour in which we visit every other vertex exactly once before returning to the vertex at which we started is called a Hamilton cycle.

To solve the traveling salesman problem, we can use a graph $G = (V,E)$ as a model and look for the Hamilton cycle with the shortest length (Loudon, 1999). $G$ is a complete, undirected, weighted graph, wherein $V$ is a set of vertices representing the points we wish
to visit and $E$ is a set of edges representing connecting between the points. Each edge in $E$ is weighted by the distance between the vertices that defines it. Since $G$ is complete and undirected, $E$ contains a total of $V(V-1)/2$ edges.

Another way to solve the traveling salesman problem is by exploring all possible permutations of the vertices in $G$. Using this approach, since each permutation represents one possible tour, we simply determine which one results in the tour that is the shortest. Unfortunately, this approach is not all practical because it does not run in polynomial time. A polynomial-time algorithm is one whose complexity is less than or equal to $O(n^k)$, where $k$ is some constant. This approach does not run in polynomial time because for a set of $V$ vertices, there are $V!$ possible permutations; thus exploring them requires $O(V!)$ time, where $V!$ is the factorial of $V$, which is the product of all numbers from $V$ down to 1 (Loudon, 1999). (Cook, 2003) has shown the various methods used to reduce the time taken for finding the solution and the time associated with the search such as cutting plane method and hybrid genetic algorithms.

In general, non-polynomial-time algorithms are avoided because even for small inputs, problems quickly become intractable (Loudon, 1999). Actually, the traveling salesman problem is a special type of non-polynomial-time problem called NP-complete. NP-complete problems are those which no polynomial-time algorithms are known, but for which no proof refutes the possibility either; even so, the likelihood of finding such an algorithm is extremely slim. With this in mind, normally the traveling salesman problem is solved using an approximation algorithm.
3.5.1 The Nearest-Neighbor Heuristic

One way to compute an approximate traveling-salesman tour is to apply the nearest-neighbor heuristic. The algorithm works as follows. We begin with a tour consisting of only the vertex at the start of the tour. We color this vertex black. All other vertices are white until added to the tour, at which point we color them black as well. Next, for each vertex \( v \) not already in the tour, we compute a weight for the edge between the last vertex \( v \) no already in the tour; we compute a weight for the edge between the last vertex \( u \) added to the tour and \( v \). The weight of an edge from \( u \) to \( v \) is the Euclidean distance between \( u \) and \( v \) using the Euclidean distance as a parameter; we select the vertex closest to \( u \), color it black, and add it to the tour. We then repeat this process until all vertices have been colored black. At this point, we add the start vertex to the tour again to form a complete cycle.

The pseudo code of the Nearest Neighbor algorithm is as follows:

\[\text{Input: A weighted complete graph } G\]

\[\text{Output: A sequence of labeled vertices that forms a Hamiltonian cycle}\]

\[\text{Start at any vertex } v.\]

\[\text{Initialize } l(v) = 0.\]

\[\text{Initialize } i = 0.\]

\[\text{While there are unlabeled vertices}\]

\[i = i + 1.\]

\[Traverse the cheapest edge that joins } v \text{ to a unlabeled vertex, say } u.\]
Set \( l(w) = i \)

\( v = w \)

As is typical of greedy algorithms, the nearest-neighbor heuristic is very fast, and it is easy to implement (Gross and Yellen, 2003b). The algorithm sometimes performs quite well.

Figure 3.5 illustrate a solution to the traveling salesman problem using the nearest-neighbor heuristic. Normally when a graph is drawn for the traveling salesman problem, the edges connecting every vertex to each other are not explicitly shown since the edges are understood. In the figure, each vertex is displayed along with the coordinates of the point it represents. The dashed lines at each stage show the edges whose distances are being compared. The darkest line is the edge added to the tour.
The nearest-neighbor heuristic has some interesting properties. Like the other algorithms, it resembles breadth-first search because it explores all of the vertices adjacent to the last vertex in the tour before exploring deeper in the graph. The heuristic is also greedy because each time it adds a vertex to the tour, it does so based on which looks best at the moment. Unfortunately, the nearest neighbor added at one point may affect the tour in a negative way later. Nevertheless, the heuristic always return a tour whose length is within a factor of 2 of the optimal tour length, and in many cases it does better than this. Other techniques exist to improve a tour once we have computed it. One technique is to apply an exchange heuristic.
3.5.2 Insertion Heuristics

An intuitive approach to the TSP is to start with a sub-tour, i.e. a tour on small subsets of nodes, and then extend this tour by inserting the remaining nodes one after the other until all nodes have been inserted. This provides a stepped approach to solve a complex problem and provides a platform where it could be applied generally.

There are several possibilities for implementing such an insertion scheme. They can be classified according to these features:

- How to construct the initial tour.
- How to choose next node to be inserted.
- Where to insert chosen node

The starting tour is usually some tour on three nodes, e.g. those nodes that form the largest triangle. For Euclidean problems, a good starting tour is the tour that follows the convex hull of all nodes. This is a reasonable choice since the sequence of nodes from the convex hull tour is respected in any optimal tour (Loudon, 1999).

A new node is usually inserted into the tour at the point that causes the minimum increase in the length of the tour.

The major difference between insertion schemes is the order in which the nodes are inserted.
• Cheapest Insertion:

Among all nodes not inserted so far, choose a node whose insertion causes the lowest increase in the length of the tour. The idea behind this strategy is pure greediness, of course. Figure 3.6 illustrates the Cheapest Insertion to solve a TSP problem.

![Figure 3.6: Cheapest Insertion.](image)

In Figure 3.6, a tour is given as an example, the new city added provided it reduces the length of the total distance of the tour and the next city is considered until all cities are included in the tour.
• Farthest Insertion:

Insert the node whose minimal distance to a tour node is maximal. The idea behind this strategy is to fix the overall layout of the tour early in the insertion process.

![Farthest Insertion](image)

**Figure 3.7: Farthest Insertion.**
Chapter 4. Genetic Algorithm and TSP

Genetic Algorithm’s feature of solving problems with the concept of survival of the fittest could be applied to the TSP. Genetic Algorithm is a robust and effective search procedure for NP-Complete problems in the sense that, although they may not outperform highly tuned, problem-specific algorithms, genetic algorithm can be easily applied to a broad range of NP-complete problems with performance characteristics no worse than the theoretical lower bound of an $N^3$ setup (De Jong and Spears, 1989).

The solution search space of the TSP is $N!$, where $N$ is the number of cities can leads to an enormously large search space. Using Genetic Algorithm’s property of evolution, this method can evolve possible solutions from a pool of possible solutions, where each generation is better than the next.

There are 3 general areas of interests that are associated in developing a Genetic Algorithm tool to solve the TSP.

1. Proper representation to encode a tour

2. Devise applicable genetic operators to keep building blocks and avoid illegality

3. Prevent premature convergence
In order to develop a solution of the TSP, the 3 areas should be addressed properly. The importance of a proper representation to encode a tour can be highlighted with using a binary encoding for chromosome string, which can lead to an illegal tour upon completing a crossover. These characteristics of proper data encoding can lead to an effective method for solving TSP with Genetic Algorithms.

The TSP could be divided into clusters as in (Loius and Tang, 1999). A tour with N cities and P clusters; the average number of cities in each cluster is N/P and therefore the search space for each cluster is (N/P)!. The total search space is P(N/P)!. When N is large, this approach will save several orders of magnitude worth of time since P x (N/P)! << N! for a large N and P.

### 4.1 Population Size

In the Visual Genetic Algorithm Tool, the population size is a critical factor in producing offspring of high fitness from the varied gene pool. For the population size based on (Alander, 1996) the conclusion that may reasonably be drawn from this study is that the size of the population in a genetic algorithm for TSP is not critical. Further studies have practically concluded that, to solve a TSP, a population size between \(n\) and \(2n\) where \(n\) is the tour length is needed to effectively solve the TSP (Yang, 1997). This brings to the critical points that for the design of the Visual Genetic Algorithm Tool, the minimum population size can be determined to solve a TSP, where \(n\) represents the minimum for the TSP.
(Reeves, 1993) investigated the minimum practical population size in a GA, applying the requirement that every point in the search space be reachable by crossover alone; that is, that the initial population be likely to include at least one instance of every possible symbol in each position. He derived functions that indicated population size as a function of chromosome length, alphabet size, and probability. These functions suggest that small populations suffice when chromosomes are binary, while coding over alphabets of higher cardinality requires larger populations.

4.2 Chromosomal Representation

Traditionally, chromosomes are a simple binary string used to encode simple optimization problems where there is no ordered arrangement of genes within a chromosome. The problem when the binary representation is applied to the TSP is shown in Figure 4.1, where it can lead to an illegal tour for a 4-city tour.

<table>
<thead>
<tr>
<th>The Chromosome</th>
<th>Binary Encoding</th>
</tr>
</thead>
<tbody>
<tr>
<td>Parent 1</td>
<td>001 010 011 100</td>
</tr>
<tr>
<td>Parent 2</td>
<td>010 001 100 011</td>
</tr>
</tbody>
</table>

Figure 4.1: The Limitation of Binary Encoding Scheme.
If the crossover occurs at the 11\textsuperscript{th} point of the binary string it can lead to a city with the value of 111 which is City 7, not existing in the tour.

Several schemes have been proposed for the TSP, among them permutation representation and random key representation which are proposed by (Cheng and Gen, 2000).

4.2.1 Permutation representation

This is the most natural representation of a TSP tour, where cities are listed in order, which they are visited. The search space is the permutation of the cities. For example a 9 city TSP:

1-2-3-4-5-6-7-8-9

Is simply represented as

\[ [1 \ 2 \ 3 \ 4 \ 5 \ 6 \ 7 \ 8 \ 9] \]

This representation is also called a path representation or order representation. This representation might lead to illegal tours if the traditional one-point crossover is used.

An illegal tour from a traditional one-point crossover is as depicted in Figure 4.2, where there is illegal tours on cities number 1, 2, 3 and 4.
A modification must be made to the during the crossover procedure in order to use the Permutation Representation to solve the TSP using Genetic Algorithms.

### 4.2.2 Random Keys Representation

This representation encodes a solution with random numbers commonly from (0,1) or any real numbers, which can be sorted in an ascending or descending manner. These values are used for sort keys to decode the solution. For example, a chromosome to a 9-city problem may as in Figure 4.3.
Figure 4.3: The Random Keys Representation.

The representation is such where position i in the list represent city i. The random number in position i determines the visiting order of city i in a TSP tour. We sort the random keys in ascending order to get the following tour:

\[6 - 1 - 3 - 7 - 8 - 4 - 9 - 2 - 5\]

Random keys eliminate the infeasibility of the offspring by representing solutions in a soft manner. Soft manner can be defined as providing an offspring selection method without any additional illegal tours made and any additional steps taken to rectify any illegal tours. Any type of crossover can be applied to the chromosome or any type of representation can be used as the traditional crossover can produce any random number, which can be rearranged to produce a tour of a TSP.

This representation is applicable to a wide variety of sequencing optimization problem including machine scheduling, resource allocation, vehicle routing, quadratic assignment problem and so on.
4.3 Selection Mechanism

There are many selection mechanisms that could be used for the TSP problem. With the selection mechanism of random stochastic selection, roulette wheel selection, tournament selection and many more could be used effectively.

The selection mechanism used for the TSP problem is the stochastic random selection. This method is used to reduce the effects of elitism, which is highly associated with the roulette wheel selection.

A possible drawback of normal roulette wheel selection is the proliferation of studs. Studs are individuals that have much higher fitness than the rest of the population and thus get to mate more often. The genes might “infect” the entire population. Normalize the fitness before these are used for basis for parent selection. This might lead to premature convergence of the solution; while in some cases the global optimum is reached in a much faster manner (Stender, 2002).

4.4 Crossover Operators

Several crossover operators have been proposed for permutation representation such as partial-mapped crossover (PMX), order crossover (OX), cycle crossover (CX), position-based crossover and so on.

Roughly, these operators can be classified into two classes:
• Canonical approach
• Heuristic approach

The Canonical approach can be viewed as an extension of a two-point or multipoint crossover of binary strings to permutation representation. Generally, permutation representation will yield illegal offspring by two-point or multipoint crossover in the sense that some cities might be missed while some cities may be duplicated in the offspring. Repairing procedures is embedded in this approach to resolving illegitimacy of offspring. The essence of Canonical approach is the blind random mechanism. There is no guarantee that an offspring produced by this kind of crossover is better than their parents. The application of heuristics in crossover intends to generate an improved offspring.

Heuristic crossover was first presented by Grefenstette (Grefenstette et al, 1985). In conventional heuristics for TSP, there are two basic construction approaches: nearest neighbor and best insertion heuristics. Grefenstette's crossover was implemented with the mechanism of nearest-neighbor heuristic. (Cheng and Gen, 2000) designed a crossover with the mechanism of best insertion for the vehicle routing and scheduling problem. The heuristic crossover works as follows:

1. For a pair of parents, pick a random city for the start

2. Choose the shortest edge (that is represented in the parents) leading from current city which does not lead to a cycle. If two edges lead to a cycle, choose a random city that continues the tour
3. If the tour is complete, stop; otherwise go to step 2

Much work has been done on the modification of the crossover process to enhance the time taken to solve the problem. In (Sengoku and Yoshihara, 1993), the 2-opt method was implemented to solve the TSP with GA and is met with much success with the solution popping up from local minima more effectively than using Simulated Annealing methods. The mutation is implemented with the 2-opt method, where a percentage of individuals are randomly chosen and improved using the 2-opt method. The elite individual or the individual with the best fitness value is always chosen. Figure 4.4 illustrates the 2-opt method used. This method is better than Simulated Annealing because it considers one only solution at a time and random steps away from the solution are taken, as the optimization continues, the average size of the step gradually decreases this contrasts against every generation in the 2-opt method which guarantees improvement as it chooses only the best individuals for enhancement.

![Figure 4.4: The 2-opt Method.](image-url)
4.4.1 Partial-Mapped Crossover (PMX)

PMX was proposed by (Goldberg and Lingle, 1985). PMX is a canonical approach to crossover. PMX can be viewed as an extension of two-point crossover binary string to permutation representation. It uses a special repairing procedure to resolve illegitimacy cause by the simple two-point crossover. Thus the essentials of PMX are a simple two-point crossover plus a repairing procedure. PMX works as follows:

1. Select two positions along the string uniformly at random. The sub strings defined by the two positions are called mapping selections.

2. Exchange the two sub strings between parents to produce proto-children

3. Determine the mapping relationship between two mapping selections

4. Legalize offspring with the mapping relationship
4.4.2 Order Crossover (OX)

Order Crossover was proposed by Davis (Davis, 1985). OX is a canonical approach to crossover. It can be viewed as a kind of variation of PMX with a different repairing procedure.

1. Select a sub string from a parent at random

2. Produce a proto-child by copying the sub string into the corresponding position of it
3. Delete the cities, which are already in the substring from the 2nd parent. The resulted sequence of cities contains the cities, which the proto-child needs.

4. Place the cities into the unfixed positions of the proto-child from left to right according to the order of the sequence to produce an offspring.

![Figure 4.6: Ordered Crossover for a 9 City TSP.](image)

**4.4.3 Cycle Crossover (CX)**

Was proposed by Oliver, Smith and Holland (Oliver, Smith, Holland, 1987). CX is a canonical approach to crossover. As in position-based crossover, it takes symbols from one parent and the remaining symbols from the other parent. The difference is that symbols from the first parent are not selected randomly and only those symbols are selected that define a cycle according to the corresponding positions between parents. CX works as follows:

1. Find the cycle that is defined by the corresponding positions of symbols between parents.
2. Copy the symbols in the cycle to a child with positions corresponding to those of one parent

3. Determine the remaining symbols for the child by deleting those symbols that are already in the cycle from the other parent

4. Fulfill the child with the remaining symbols

In (Vassil and Penev), the cycle crossover operator is used in solving the TSP successfully eliminating the illegal tour possibility in addition to providing a good solution to the TSP.

**Cycle Crossover**

2 Parents for the Tour of a 9 city TSP  
\[ p_1 = (1 2 3 4 5 6 7 8 9) \]
\[ p_2 = (4 1 2 8 7 6 9 3 5) \]

Take the first city from Parent 1 \((p_1)\), \(\sigma_1\) denotes Children 1  
\[ \sigma_1 = (1 x x x x x x x) \]

The next city must be from Parent 2 \((p_2)\) and from the same position (City 4)  
\[ \sigma_1 = (1 x 4 x x x x x) \]

In \(p_2\), the same position as 4 in \(p_1\), there is city 8  
\[ \sigma_1 = (1 x 4 x x x x 8 x) \]

In \(p_2\), the same position of 8 is City 3  
\[ \sigma_1 = (1 x 3 4 x x x 8 x) \]

In \(p_2\), the same position of 3 is City 2  
\[ \sigma_i = (1 2 3 4 x x x 8 x) \]

Selection of 2 will cause City 1 and there will be a cycle & use second parent to fill in \(\sigma_i\)  
\[ \sigma_i = (1 2 3 4 7 6 9 8 5) \]

If \(p_2\) is selected, the outcome will be:  
\[ \sigma_i = (4 1 2 8 x x x 3 x) \]

At the first cycle and thus the complete chromosome will be:  
\[ \sigma_i = (4 1 2 8 5 6 7 3 9) \]

Figure 4.7: An Example of Cycle Crossover.
4.4.4 Greedy Crossover (GX)

The Greedy Crossover operator was first introduced by (Grefenstette et al, 1985). GX is a heuristic approach to crossover. The Greedy Crossover is sometimes referred to as Greedy-swap. The basic idea of Greedy Crossover is to randomly select two cities from one chromosome and swap them if the new (swapped) tour length is shorter than the old one. This leads to an always-converging solution pointing towards the shortest tour length of a TSP. The comparison is made prior to crossover, which brings to the concluding factor that only offspring with high fitness values are created.

The Greedy Crossover is a specific type of crossover. It can only be is applied if

1. All genes in the chromosome are different and
2. The set of genes for both chromosomes is identical and only they order in the chromosome can vary.
Figure 4.8 graphically shows the effect of the Greedy Crossover. Tour 2 is the possible output of the Greedy Crossover of a tour with a much shorter distance if compared to Tour 1.

Greedy Crossover selects the first city of one parent, compares the cities leaving that city in both parents, and chooses the closer one to extend the tour. If one city has already appeared in the tour, we choose the other city. If both cities have already appeared, we randomly select a non-selected city.
4.5 Mutation Operators

In order to use the mutation operator, the traditional bit inversion method could lead to illegal tours and thus slight modifications need to be facilitated to suit the TSP. The mutation operator must be able to conform to the TSP rules.

The mutation might be a pair-wise exchange of genetic material. Mutation in the TSP problem is generally to perform random exchange of genes between two randomly selected positions on target chromosomes (Bagchi, 1999).

The mutation operators for TSP will involve the trade of cities in a chromosome. This might lead to a better fitness child. Some of the mutation operators are considered are:

- Inversion Mutation - Inversion mutation selects two positions within a chromosome at random and then inverts the sub-string between these two positions.

![Figure 4.9: The Inversion Mutation.](image)
• Insertion Mutation - Insertion mutation selects a city at random and inserts it in a random position.

![Insertion Mutation Diagram]

Figure 4.10: The Insertion Mutation.

• Displacement Mutation - Displacement mutation selects a sub-tour at random and inserts it in a random position. Insertion can be viewed as a special case of displacement in which the sub-string contains only one city. In essence 2 cities are exchanged.

![Displacement Mutation Diagram]

Figure 4.11: The Displacement Mutation.
4.6 The Visual Genetic Algorithm Tool

Genetic Algorithm tools such as Matlab’s Genetic Algorithm and Direct Search Toolbox provides a complete tool that is complex to learn and is not intuitive to individuals with no prior knowledge of computer science to properly understand genetic algorithms and furthermore utilize the tool as an alternative to solve problems such as in finance.

![Figure 4.12 Screen Capture of Matlab’s Genetic Algorithm Tool.](image)

In addition, Matlab provides a platform for users to program a visual tool but in-depth knowledge is needed to fully utilize the platform. However textual information is much readily available to the users and knowledge of computing is needed to interpret the results.
obtained from the model. Figure 4.12 illustrates the overview of Matlab’s Genetic Algorithm Tool. The interface design is suitable for mathematical optimization and does not provide an intuitive overview for visualization techniques in certain problems. This compels further if the optimization problem is the TSP. Matlab tool does not provide a user interface ready to solve the TSP.

The visual tool will attempt to simplify understanding genetic algorithms where it can serve as an educational tool for individuals with no prior knowledge of computer science to properly understand genetic algorithms how it is used to solve a TSP. The visualization aspect of the tool enables the user to visualize the effects of genetic algorithm in each generation, improving the tour distance by finding the shortest route. This is not available to textual tools, where only figures are displayed without any visual affirmation on the results obtained.

In general, visual tools are symbols graphically linked by mental associations to create a pattern of information and a form of knowledge about an idea (Hartley, 1996). The visual tool must be able to provide a graphical translation of genetic algorithms in action. For this, the visual tool must graphically depict using genetic algorithms to solve the TSP. In (Vassil and Penev, 2005), a genetic algorithm tool was designed to solve TSP with focus on effects of varying the crossover methods and mutations on solving TSP and provides an abstraction of the graphical interface.
Chapter 5. Design and Implementation

The Visual Genetic Algorithm Tool for TSP will be designed and implemented using the best-known methods researched in Chapter 2, Chapter 3 and Chapter 4. For this tool, the Java programming language is chosen to code the software.

There are many inherent benefits of the Java programming language for general science and engineering applications. They are:

- **Platform Independence & Portability**
  The application is Operating System (OS) independent. Java codes can be exchanged without requiring rewrites and recompilation saves time and effort.

- **Object Oriented**
  The Java programming language is an Object Oriented Programming language. This encourages reuse of codes and reducing complexity in programs.

- **Networking**
  Java comes with many networking capabilities that allow distributed systems and applications across the Wide Area Network (WAN).

- **Visual Support & Simplicity**
  The Java programming language comes with an extensive support for Graphical User Interface (GUI) and is able to provide a much better user interaction. Java has
access to a set of standard components through the Abstract Windowing Toolkit (AWT). The AWT provides standard components such as buttons, text boxes and list boxes, a set of container components and an environment for creating event handlers for the components. The set of existing components provides an added benefit in designing graphical tool by simplifying the design and planning of the software.

In (Liao and Sun, 2001), the main areas of concern for an educational genetic algorithm tool are representation, evaluation function, initial population, operators and parameters. The areas of concern will be addressed in the design of the Visual Genetic Algorithm Tool for TSP. The design also will take into consideration the good principles of software engineering and interaction design to ensure the software produced is of good quality.

5.1 The Algorithm Design

For the Visual Genetic Algorithm Tool for TSP a mixture of algorithms and methods is used to produce a tool where it can effectively solve the TSP. The best method introduced in Chapter 4 will be used in the design of the tool. For the fitness evaluation, the route with the shortest distance is considered to be the fitter solution.

5.1.1 Population Size

For the Visual Genetic Algorithm Tool, the population size can be varied. For the tool, it is proposed that the size of the population used to be similar to the number of cities in the tour. This step ensures that there is enough genetic material and diversity to ensure that the
fitness of the chromosomes increases in a much quicker manner each generation, as there are a larger number of candidates for reproduction. The population size is maintained constant throughout the genetic algorithm search.

5.1.2 The Fitness Evaluation

The fitness of each chromosome is the total Euclidean distance traveled between all the cities. The Euclidean distance is in a two-dimensional space and will involve the x and y coordinates. All in all, the higher the fitness the shorter Euclidean distance is.

The fitness measurement is based on pixel units in the map. Figure 5.1 illustrates how the distance between two cities is calculated with the results based on pixel units.

\[
\text{Distance} = \sqrt{(x_2-x_1)^2 + (y_2-y_1)^2}
\]

Figure 5.1: The Fitness Evaluation Based on Pixels.

5.1.3 Random Keys Representation
The visual tool will use the random keys representation. This can be attributed to the
elegance and simplicity of its application and use. There is no need for additional code to
perform effort correction efforts to rectify illegal tours. A minimum of 2 dimensional arrays
is used to represent each city. The array will hold information about the city as well as the
random key generated. The random keys are then sorted in ascending order to produce the
tour.

5.1.4 Selection Mechanism

For the selection mechanism, the random stochastic selection is used to truly select the
parents randomly for reproduction.

5.1.5 Greedy Crossover

The greedy crossover is selected for the reproduction method. The greedy operator is best
suited for the TSP problem as the tour route is exchanged only if they improve upon the
fitness of the solution and will lead quickly to a solution. In the visual tool, a statement that
compares fitness of the offspring and parents is conducted prior to completing the
crossover, where the crossover occurs if the fitness improves.
Figure 5.2 shows the greedy crossover in action for a tour. The map on the left shows the entangled tour route and the map on the right shows the effect of greedy crossover in untangling the tour route.

5.1.6 Inversion Mutation

The inversion mutation is used in the visual tool for the offspring. The array in the chromosome is randomly selected and then the string is inverted to produce a mutated offspring.
Figure 5.3 depicts the effects of insertion mutation in inverting the order of the cities in producing an optimal tour route. The black box shows the ordering of the cities in the tour change after the mutation process.

5.2 User Interface Design & Implementation

The design of the Visual Genetic Algorithm Tool to solve TSP will be based on good user interface design principles. The tool must explicitly show users that it is easy to learn, effective and efficient to use and provides an enjoyable experience. In this situation, the Direct Manipulation (DM) conceptual model is used as it uses physical action and button pressing instead of issuing commands with complex syntax. In addition to that, DM provides rapid reversible actions with immediate feedback on object of interest (Preece, Rogers et al. 1994).

The Visual Genetic Algorithm Tool for TSP will have the properties of a well-designed user interface design. Figure 5.4 depicts the user interface outlook of the tool. The Java applet’s design is meant to be intuitive to the user providing simple point and click method for the tool.
Figure 5.4: The User Interface of The Visual Genetic Algorithm Tool for TSP.

The green background is the map for the TSP. The cities will be initialized on the green canvas randomly based on the number of cities selected for the problem. The box below titled “Time history of shortest distance” tracks the distance of the tour per generation throughout the search by GA. The meaning of each data field associated in the visual tool is as follows:
• Population size
  Indicates the population size of the genetic algorithm search

• Initial best distance
  The Initial best distance shows the initial tour distance of the TSP. The cities are randomly generated along with the tour path.

• Shortest distance now
  It is the shortest tour distance of the TSP obtained

• Generation
  The current generation of the GA search cycle

• Select number of cities
  Select the number of cities in the TSP. The cities can be varied in groups of 5.

The user is given a choice to choose the number of cities in the TSP. To randomly generate cities within the map, click on the “Initialize Map” button.
Figure 5.5: The Randomly Generated Cities in The Map.

Figure 5.5 shows the effects of initializing the map and generating 20 random cities. By clicking the “Start” button, it will then randomly create a tour path through all the 20 cities and record the initial Euclidean distance in the text box labeled “Initial best distance:”. The application will then utilize GA to optimize the tour of the cities to produce a shortest path for the salesman to travel. The tour distance will also be recorded in a two dimensional
graph in the white box below. Using red to label the cities will ensure that the cities are visible to the user taking into consideration good design principles. The tour path will be denoted by a black line running from each city in the map, connecting the cities involved. The fitness will be the distance traveled in the tour.

5.3 Software Architecture Design & Implementation

The Visual Genetic Algorithm Tool for solving TSP, the Pipes and Filters architecture is used to design the software. Having well-defined software architecture ensures simplicity during troubleshooting, reduction in program complexity, well-defined program structures and improving on code reuse.

In pipe and filter architecture, each component has a set of inputs and a set of outputs. A component reads streams of data in its inputs and produces streams of data on its outputs; the analogy is similar to pouring water in a pipe, hence the name. This is done usually by applying a local transformation to the input streams of data in its output. Figure 5.6 shows the generic pipes and filters architecture.
The pipes and filter architecture restricts the topologies to a linear sequence of filters; bounded pipes which in turn restricts the amount of data that can reside on a pipe; and typed pipes, which require that the data passed between two filters have a well defined type. The architecture is more of a batch sequential system.

The benefits according to (Shaw and Garlan, 1996) are as follows:

- Better understanding & Simplicity
  Understand the overall input/output behavior of the system as a simple composition of the behaviors of the individual filters
- Support reuse
  Any two filters can be hooked together provided they agree on the data being transmitted
- Easy to maintain and enhance
New filters can be added to existing systems and old ones replaced by improved ones

- Permit certain kind of specialized analysis
  
  Due to the architecture design, the execution can be analyzed component by component

- Support concurrent execution
  
  Each filter can be implemented as a separate task and potentially executed in parallel

For the Visual Genetic Algorithm Tool for TSP, due to the unique nature of GA where the algorithm is sequentially executed enables it to be designed effectively using the pipes and filters architecture.
Figure 5.7 depicts the architecture of the visual tool based on the pipes and filters architecture. The prime objective of this design is to ensure that the application is simple taking into consideration the algorithm of the tool. The pipes and filter design ensures that the visual tool can be implemented in stages where each filter can be independently coded with no arrangement or precedence taking into consideration the pipes inputs and outputs.
Figure 5.8 shows the collaboration diagram for the Visual Genetic Algorithm Tool. The diagram shows the set of classes together interacting to deliver the functionality of the tool. The user will first get the tool’s user interface to initialize upon starting the program. The map of the tour will be initialized enabling the user to have a visual overview of the TSP. After that the population of the TSP is generated. The initial tour route is randomly initialized creating a tour path and fitness is then evaluated. The initial tour distance is then recorded and the point is then plotted in the graph. The parents of best fitness values are then selected based on the population for crossover and an offspring is created. The offspring is then mutated. The new tour distance is then recorded and the values updated in the graph. Figure 5.9 illustrates the class diagram for the Visual Genetic Algorithm Tool for solving the Traveling Salesman Problem.
5.3.1 The Algorithm for The Visual Genetic Algorithm Tool

For the Visual Genetic Algorithm Tool, the random keys representation is used for the chromosome, the greedy crossover is used for the reproduction and the inversion mutation is selected to mutate the offspring.

To encode the chromosome, the user first defines the number of cities and then the cities defined by the user are randomly inserted in a 3 dimensional array consisting of the coordinates. The pseudocode for encoding the chromosome into the random keys representation is as follows:
initialize the map of the TSP
reset x coordinates
reset y coordinates

for (i=2; i < number of cities set by user; i++)
{
    randomly insert city
    insert coordinates into array xx[i]
    insert coordinates into array yy[i]
draw city in map
draw the tour path
    map.drawLine(xx[1], yy[1], xx[2], yy[2])
}

a is defined as a 3 dimensional array and is for used during crossover
gene is defined as a 3 dimensional array and is to the information stored in the
chromosome

for(j = 2; j < number of cities set by user + 1; j++)
{
    k = (int) random number
    a[x][y][z] = gene [xx][yy][k]
}

sort chromosome in ascending order for based on k
The pseudocode for greedy crossover is as follows:

randomly select 2 parents from top 5% of array[n]

parent1 = [xx1][yy1][k1]

parent2 = [xx2][yy2][k2]

crossover gene at position k

new tour = [xx3][yy3]

if (fitness new tour > fitness parent 1 && fitness new tour > fitness parent 2)
{
    perform the crossover
    insert new tour into population
}

else

discard new tour

The inversion mutation is achieved with the following pseudocode:

mutation rate = 10%

produce random number <= 10

if (random number == 10)
{
    generate random number, a1 where a1 must be an integer number less than

    the number of cities by user
generate random number, a2 where a2 must be an integer number less than
the number of cities by user
exchange cities in position noted by a1 and a2
redraw tour map
insert mutated tour into population

}

else
    redraw tour map
Chapter 6. Results and Observations

The results and observations are based on the Visual Genetic Algorithm Tool designed specially to solve the Traveling Salesman Problem. The tool is a java applet and is platform independent. The tool can be launched via a java enabled web browser.

6.1 Results for a 20 City TSP

A 20-city map is initialized as in Figure 6.1. A 20-city TSP will have a possible solution search space of 20! or 2,432,902,008,176,640,000 choices.
Upon starting the search for the optimal tour, the initial distance of the tour is randomly generated and is recorded as 2697 pixels for the 20-city tour. At the 14th generation, the shortest tour recorded is 2524 pixels, an improvement over the initial distance of 2697 pixels. A histogram records the tour distance per generation showing a decreasing graph, where the fitness improves throughout the run. This is as shown in Figure 6.2.
Figure 6.2: The Tour of the 20 City TSP at the 14\textsuperscript{th} Generation.

The Visual Genetic Algorithm Tool is able to visually depict the improvement of the tour per generation from the initial tour distance obtained from generation 0. Figure 6.3 further shows the effects that genetic algorithm can bring to the tour distance.
Figure 6.3: Shows the Tour Distance Reducing per Generation.

Figure 6.4 is the screen capture at generation number 107, which brings the current distance to 1473. Visually, the user is shown the improvement by changes in the tour route. Using greedy crossover, the routes are often shorten per generation as the name implies, the reproduction occurs only if it brings any improvement over the current generation.
Figure 6.4: The Tour of the 20 City TSP at Generation Number 107.
Figure 6.5: The Tour of the 20 City TSP at Generation Number 185.
Figure 6.6: The Optimal Tour of the 20 City TSP with Tour Distance of 1254 Pixels.

Figure 6.5 and Figure 6.6 shows screen captures at increasing generation numbers. Figure 6.6 depicts the optimal tour of the 20-city TSP. The distance of the 20-city TSP is 1254 pixels. This is an improvement of 1443 pixels from the initial tour distance of 2697 pixels. The tool visually shows the shortest route possible for the 20-city TSP at around 394 generations. This shows that genetic algorithm is successfully able to solve a TSP.
Furthermore; the effects of genetic algorithm can be visually depicted precisely, showing the decrement of tour distance per generation.

![Figure 6.7: Tour Route With Nearest Neighbor Heuristics.](image)

If the obtained results from the Visual Genetic Algorithm Tool for TSP are compared with nearest neighbor heuristic in Figure 6.7, the visual tool is able to provide a good tour solution for the 20-city TSP. Since both solutions are acquired from approximation algorithms, the comparison is done visually for tour route comparison. For the nearest neighbor heuristics, city 1 is chosen as the starting point and the cities are labeled in according to the search sequence or sequence of visit based on the visiting the nearest city from the current city.
6.2 Results for a 50 City TSP

To further test the Visual Genetic Algorithm Tool designed for solving TSP, the number of cities in the map is increased. Generally, the search space for a 50-city TSP is around 50! or 3.0414 x 10^{64} possible tour routes. This proves to be a challenge for any method used to solve the problem.
Figure 6.8 shows the tool randomly generating a 50-city TSP map. Upon initialization, the initial best distance recorded is 8267. Figure 6.9 shows the route at generation number 26, which has the distance of 7310 pixels. This clearly shows an improvement of 957 pixels over the course of 26 search iterations.
Figure 6.9: The Tour of the 50 City TSP at Generation Number 26.

Figure 6.10 shows the optimal tour for the 50-city TSP, which is obtained roughly around generation 2751 with a distance of 2226 pixels. The improvement made to the route fitness is the reduction of 5516 pixels from the initial distance. This translates to around 66.7% improvement to the tour distance.
Figure 6.10: The Optimal Tour of the 50 City TSP with a Tour Distance of 2226 Pixels.

In short, the Visual Genetic Algorithm Tool has successfully solved small scale and large scale TSP and graphically depicts the improvements made throughout the genetic algorithm search.
6.3 The Visual Aspect of the Visual Genetic Algorithm Tool

The Visual Genetic Algorithm Tool developed for the TSP is successful in visually depicting on how genetic algorithm is used to optimize the tour route in a TSP. The tool successfully visualizes the evolution of the chromosomes and the effect of the genetic operators.

![Figure 6.11: The Map of the 50-City TSP.](image)

Figure 6.11 depicts the tool’s ability to visually generate random TSP maps, which visually depict the path traveled initially by the salesman. The black lines denote the path traveled by the salesman. This tool is able to show improvements in the map visually by the length of the black line reducing and the tour pattern changing. Figure 6.12 shows the optimized tour route using the Visual Genetic Algorithm Tool. This is depicted visually and is able to show improvement overall using genetic algorithm.
6.4 Overall Results and Observation

This section summarizes the results obtained from the Visual Genetic Algorithm Tool for TSP. The Visual Genetic Algorithm tool is used to solve a 20-city TSP and 50-city TSP. The performance of varying population size is recorded for both 20-city and 50-city problems. The objective is to measure the performance in obtaining the shortest tour route as reflected in the reduction in distance per generation with the varying of population size.

20 City TSP

The results for the 20-city TSP are as obtained and tabulated in Table 6.1. The population size is varied from 20, 30, 40 and 50. The results shows that the performance of the TSP run with a population size of 50 brings the best results for a total run of 300 generation. The GA run converges at around 190 generations while the GA with population size of 20 only reaches to the best tour route at around 300 generations.
Table 6.1: The Results for 20 City TSP With Varying Population Size.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Distance (Pixels)</td>
<td>Distance (Pixels)</td>
<td>Distance (Pixels)</td>
<td>Distance (Pixels)</td>
</tr>
<tr>
<td>0</td>
<td>3257</td>
<td>3518</td>
<td>3526</td>
<td>3217</td>
</tr>
<tr>
<td>10</td>
<td>2946</td>
<td>2868</td>
<td>3184</td>
<td>2303</td>
</tr>
<tr>
<td>20</td>
<td>2312</td>
<td>2719</td>
<td>2856</td>
<td>2007</td>
</tr>
<tr>
<td>30</td>
<td>2245</td>
<td>2575</td>
<td>2795</td>
<td>1987</td>
</tr>
<tr>
<td>40</td>
<td>2185</td>
<td>2284</td>
<td>2748</td>
<td>1943</td>
</tr>
<tr>
<td>50</td>
<td>2165</td>
<td>2183</td>
<td>2492</td>
<td>1943</td>
</tr>
<tr>
<td>60</td>
<td>2165</td>
<td>2038</td>
<td>2407</td>
<td>1898</td>
</tr>
<tr>
<td>70</td>
<td>2165</td>
<td>1830</td>
<td>2270</td>
<td>1760</td>
</tr>
<tr>
<td>80</td>
<td>2119</td>
<td>1830</td>
<td>2008</td>
<td>1751</td>
</tr>
<tr>
<td>90</td>
<td>2119</td>
<td>1830</td>
<td>1898</td>
<td>1751</td>
</tr>
<tr>
<td>100</td>
<td>2004</td>
<td>1748</td>
<td>1882</td>
<td>1663</td>
</tr>
<tr>
<td>110</td>
<td>1853</td>
<td>1613</td>
<td>1882</td>
<td>1527</td>
</tr>
<tr>
<td>120</td>
<td>1853</td>
<td>1574</td>
<td>1829</td>
<td>1443</td>
</tr>
<tr>
<td>130</td>
<td>1786</td>
<td>1574</td>
<td>1735</td>
<td>1443</td>
</tr>
<tr>
<td>140</td>
<td>1768</td>
<td>1574</td>
<td>1735</td>
<td>1443</td>
</tr>
<tr>
<td>150</td>
<td>1720</td>
<td>1574</td>
<td>1684</td>
<td>1436</td>
</tr>
<tr>
<td>160</td>
<td>1720</td>
<td>1566</td>
<td>1684</td>
<td>1436</td>
</tr>
<tr>
<td>170</td>
<td>1720</td>
<td>1507</td>
<td>1684</td>
<td>1436</td>
</tr>
<tr>
<td>180</td>
<td>1720</td>
<td>1507</td>
<td>1684</td>
<td>1436</td>
</tr>
<tr>
<td>190</td>
<td>1652</td>
<td>1507</td>
<td>1565</td>
<td>1271</td>
</tr>
<tr>
<td>200</td>
<td>1652</td>
<td>1507</td>
<td>1565</td>
<td>1271</td>
</tr>
<tr>
<td>210</td>
<td>1626</td>
<td>1482</td>
<td>1565</td>
<td>1271</td>
</tr>
<tr>
<td>220</td>
<td>1626</td>
<td>1482</td>
<td>1565</td>
<td>1271</td>
</tr>
<tr>
<td>230</td>
<td>1626</td>
<td>1482</td>
<td>1565</td>
<td>1271</td>
</tr>
<tr>
<td>240</td>
<td>1626</td>
<td>1482</td>
<td>1565</td>
<td>1271</td>
</tr>
<tr>
<td>250</td>
<td>1626</td>
<td>1482</td>
<td>1565</td>
<td>1271</td>
</tr>
<tr>
<td>260</td>
<td>1626</td>
<td>1482</td>
<td>1565</td>
<td>1271</td>
</tr>
<tr>
<td>270</td>
<td>1626</td>
<td>1482</td>
<td>1565</td>
<td>1271</td>
</tr>
<tr>
<td>280</td>
<td>1626</td>
<td>1482</td>
<td>1565</td>
<td>1271</td>
</tr>
<tr>
<td>290</td>
<td>1593</td>
<td>1482</td>
<td>1565</td>
<td>1271</td>
</tr>
<tr>
<td>300</td>
<td>1593</td>
<td>1482</td>
<td>1565</td>
<td>1271</td>
</tr>
</tbody>
</table>

Figure 6.13 is the graph of Table 6.1 and clearly shows that for the GA search with 20-city TSP takes the longest search time in generations to produce a shorter tour route compared
to the search with population size of 30, 40 and 50. This might be attributed to the larger population size as in the work of (Patton, Dexter and Punch III, 1997) and (Alander, 1996).

The performance for population of size 40 is much less than the performance for population of size 30. This can be seen from the tour distance reduction, an improvement of 2036 pixels is observed for a population size 30 and 1961 pixel is observed for a population size of 40. Since much of genetic algorithm is random from the parent selection, crossover to mutation, the experiment with population size of 40 could not be ascertained to produce the best solution but generally the performance improves and the tour distance reduces per generation.

![Graph of 20 City TSP](image)

Figure 6.13: Graph of 20 City TSP.
The results for the 50-city TSP are as obtained and tabulated in Table 6.2 and the graph is in Figure 6.14. The population size is varied from 40, 50, 60 and 70. The GA search is run for 300 generations for all population size instances. As in (Patton, Dexter and Punch III, 1997) and (Alander, 1996), the results of the GA search of population size of 70 bring the best performance in terms of reducing tour distance for a total of 300 generations.

Table 6.2: The Results for 50 City TSP With Varying Population Size.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Distance (Pixels)</td>
<td>Distance (Pixels)</td>
<td>Distance (Pixels)</td>
<td>Distance (Pixels)</td>
</tr>
<tr>
<td>0</td>
<td>9150</td>
<td>8710</td>
<td>8377</td>
<td>9234</td>
</tr>
<tr>
<td>10</td>
<td>8704</td>
<td>7907</td>
<td>7803</td>
<td>8932</td>
</tr>
<tr>
<td>20</td>
<td>8597</td>
<td>7444</td>
<td>7215</td>
<td>8648</td>
</tr>
<tr>
<td>30</td>
<td>8205</td>
<td>7134</td>
<td>7514</td>
<td>8164</td>
</tr>
<tr>
<td>40</td>
<td>7744</td>
<td>6422</td>
<td>7130</td>
<td>7843</td>
</tr>
<tr>
<td>50</td>
<td>7708</td>
<td>6380</td>
<td>6946</td>
<td>7208</td>
</tr>
<tr>
<td>60</td>
<td>7036</td>
<td>5911</td>
<td>6734</td>
<td>6987</td>
</tr>
<tr>
<td>70</td>
<td>6803</td>
<td>5656</td>
<td>6340</td>
<td>6892</td>
</tr>
<tr>
<td>80</td>
<td>6303</td>
<td>5574</td>
<td>5514</td>
<td>6202</td>
</tr>
<tr>
<td>90</td>
<td>6336</td>
<td>5387</td>
<td>5115</td>
<td>5911</td>
</tr>
<tr>
<td>100</td>
<td>6290</td>
<td>5164</td>
<td>5001</td>
<td>5367</td>
</tr>
<tr>
<td>110</td>
<td>6187</td>
<td>5126</td>
<td>4930</td>
<td>5217</td>
</tr>
<tr>
<td>120</td>
<td>6046</td>
<td>4993</td>
<td>4803</td>
<td>4931</td>
</tr>
<tr>
<td>130</td>
<td>5970</td>
<td>4902</td>
<td>4803</td>
<td>4842</td>
</tr>
<tr>
<td>140</td>
<td>5970</td>
<td>4814</td>
<td>4611</td>
<td>4766</td>
</tr>
<tr>
<td>150</td>
<td>5715</td>
<td>4788</td>
<td>4598</td>
<td>4766</td>
</tr>
<tr>
<td>160</td>
<td>5588</td>
<td>4653</td>
<td>4521</td>
<td>4659</td>
</tr>
<tr>
<td>170</td>
<td>5662</td>
<td>4576</td>
<td>4521</td>
<td>4498</td>
</tr>
<tr>
<td>180</td>
<td>5441</td>
<td>4561</td>
<td>4521</td>
<td>4326</td>
</tr>
<tr>
<td>190</td>
<td>5631</td>
<td>4561</td>
<td>4366</td>
<td>4299</td>
</tr>
<tr>
<td>200</td>
<td>5061</td>
<td>4372</td>
<td>4216</td>
<td>4001</td>
</tr>
<tr>
<td>210</td>
<td>5029</td>
<td>4283</td>
<td>4192</td>
<td>4001</td>
</tr>
<tr>
<td>220</td>
<td>4805</td>
<td>4283</td>
<td>4082</td>
<td>3789</td>
</tr>
<tr>
<td>230</td>
<td>4716</td>
<td>4132</td>
<td>4025</td>
<td>3688</td>
</tr>
<tr>
<td>240</td>
<td>4716</td>
<td>3976</td>
<td>3950</td>
<td>3579</td>
</tr>
<tr>
<td>250</td>
<td>4716</td>
<td>3887</td>
<td>3819</td>
<td>3511</td>
</tr>
<tr>
<td>260</td>
<td>4736</td>
<td>3824</td>
<td>3723</td>
<td>3474</td>
</tr>
<tr>
<td>270</td>
<td>4627</td>
<td>3715</td>
<td>3618</td>
<td>3455</td>
</tr>
<tr>
<td>280</td>
<td>4537</td>
<td>3622</td>
<td>3567</td>
<td>3414</td>
</tr>
<tr>
<td>290</td>
<td>4524</td>
<td>3590</td>
<td>3531</td>
<td>3414</td>
</tr>
</tbody>
</table>
Figure 6.14: Graph of 50 City TSP.
Chapter 7. Conclusion

The objective of the dissertation is to design a Visual Genetic Algorithm Tool. The genetic algorithm tool must be able to show the effects of the genetic algorithm in solving an optimization problem. In this case, the problem is the Traveling Salesman Problem, which is a NP-Complete problem that has a large search space. The Traveling Salesman Problem is an ideal problem that is readily solved visually using Genetic Algorithms. The Genetic Algorithm tool is then able to show the improvements that Genetic Algorithms can make in each generation, slowly converging to the answer or the tour with the shortest distance.

7.1 Discussions and Conclusion

To successfully design and implement the Visual Genetic Algorithm Tool for the Traveling Salesman Problem, a through study of Genetic Algorithms and the Traveling Salesman Problem is concluded where the algorithm will be modified to suit the problem domain. During this phase, one of the most important factors to be considered is the chromosomal representation, which differs from each problem to avoid having illegal tours. Slight modifications are to be made to accommodate the unique data representation of the Traveling Salesman Problem. The second point to consider is the reproduction function or the crossover function. Traditional crossover method such as the single point crossover might result in an illegal tour and adaptation to the algorithm must be made to successfully solve the problem. The third and final factor of concern is the mutation of the chromosome. Inverting bits of a chromosome string is not feasible due to the data representation and thus
other methods to swap cities in a tour are adapted. Upon that, finally the Java programming language is then used to code the program taking into consideration the modifications.

In summary, the genetic algorithm is able to successfully produce a solution for the Traveling Salesman Problem. The genetic algorithm is able to visually show improvements to the length of the tour in each generation showing the tour is converging to the shortest tour distance. In addition to that, genetic algorithm is successfully shown to optimize the Rosenbrock function and Rastrigin’s function based on varying genetic algorithm parameters. The effects of varying the genetic operators are compared and contrasted solidifying the literature review conducted for genetic algorithms and further understands the underlying concepts of genetic algorithms.

7.2 Further Work

In the dissertation, the Traveling Salesman Problem was successfully solved by a Visual Genetic Algorithm Tool, which is specially designed for that purpose. A generalized tool can be designed to incorporate a Symmetrical TSP where the distance of travel from A to B is not similar to the distance from B to A. The Symmetrical TSP is widely used in the logistics industry where an example would be to move cargo from A to B using cargo ship might have the time of 1 hour while moving cargo in the reverse fashion might occur faster using the plane with the time of 45 minutes.

Furthermore, the TSP can be adapted to solve a routing problem. Adding weights to the route enables the differences in path traveled and it apparently is widely used in routing protocols such as EIGRP or BGP. Genetic Algorithms can be used to solve the traffic
routing problem and then it could be compared and contrasted. The route optimization
could also be graphically visualized similarly to the TSP problem.

Finally, some further work could be conducted to investigate the efficiency and
effectiveness of the various crossovers and mutation operators introduced in the earlier
chapters. With that, a final recommendation on the most effective method to solve a TSP
can be concluded.
References


