MULTI OFFSPRING GENETIC ALGORITHM WITH DYNAMIC MUTATION AND ITS APPLICATION TO THE TRAVELING SALESMAN PROBLEM

BY

Fredgie Louis B. Aquino

A special problem submitted to the

Department of Mathematics and Computer Science

College of Science

The University of the Philippines

Baguio, Baguio City

as partial fulfillment of the

requirements for the degree of

Bachelor of Science in Computer Science

June 2022

This is to certify that this Special Problem entitled **"Multi Offspring Genetic Algorithm with Dynamic Mutation and its Application to the Traveling Salesman Problem"**, prepared and submitted by **Fredgie Louis B. Aquino** to fulfill part of the requirements for the degree of **Bachelor of Science in Computer Science**, was successfully defended and approved on June 11, 2022.

Lee Javellana
Special Problem Adviser

The Department of Mathematics and Computer Science endorses the acceptance of this Special Problem as partial fulfillment of the requirements for the degree of Bachelor of Science in Computer Science .

Jerico B. Bacani, Ph.D.
Current Dept. Chair
Department of Mathematics and
Computer Science

Table of Contents

Acknowledgments

The pandemic has taken a toll on everyone, including me. Anxiety has left me hanging and made me postpone anything I would want to do, including this research. Now, I have decided to stand up and finish the tasks I should be doing.

First of all, I would like to extend my sincerest gratitude to my family, especially to my Mamang Regie and Papang Freddie. It has been a long and unconventional path, but you guys stayed by my side all the way. Your support meant a lot to me and I honestly couldn't have done it without you. Shoutout also to my brothers, Al Francis and Mark Lee, for always supporting your kuya. You guys are the best.

I also sincerely thank all the people who have been instrumental in my daily survival for the past years. As a student, it is difficult to get opportunities that will help you get by. To the local fashion industry, thank you for introducing me to my potentials and to all the amazing people I have met and worked with. To my MarkeTeam, I am beyond grateful for all the experiences and the consistent shoot gigs you have given me. To the SG team, thank you for giving me the chance to practice my degree for the common good. To the other people I have worked with for the past years, I am grateful for all your existence.

I would also want to send my appreciation to all my friends who stayed with me, even during the darkest of times. Life has taught me that it could always be better if you have a nice set of people around you. The times we spent together, face to face or virtual, were vital in maintaining my sanity through the darkest times. I would forever cherish every laughter and tears that we all shared together. Labyu all!

To the people who pushed me to finally finish this, thank you for the pressure and for the constant reminders on the things that need to be done. Your words and actions were the extra push I needed to motivate myself into completing this research.

A big thanks also to my research adviser, Sir Lee Javellana, who is always an email away. Thank you for the kindness, guidance, and patience while I am doing this work.

Lastly, I would like to thank God, who made all these things possible.

Abstract

Multi Offspring Genetic Algorithm with Dynamic Mutation and its Application to the Traveling Salesman Problem

Fredgie Louis B. Aquino
University of the Philippines, 2022

Adviser:
Lee Javellana

In this paper, solutions for the Traveling Salesman Problem (TSP) are solved using a modified version of the Genetic Algorithm (GA). The Multi-Offspring Genetic Algorithm with Dynamic Mutation (MO-GA with DM) finds optimized solutions for the TSP by applying a modification to the parameters to a currently existing algorithm. By obtaining solutions of different maps using Matlab, results have shown that the proposed algorithm has promising results as it also outperforms the preexisting algorithm.

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Background of the Study

Given a set of locations, what is the shortest path that will take you to visit all of this locations and go back to your starting point? This question is answered by the Traveling Salesman Problem (TSP), a combinatorial problem that was defined in the 1930s by Karl Menger [6,28,32]. Aside from a simple calculation of distance, its methods and foundations are also applied in logistics, planning, scheduling, routing problems, and other related fields [27].

John Holland introduced the Genetic Algorithm (GA) in 1975 [29]. It is one of the most commonly used Evolutionary Algorithms and is mainly based on Darwin's Theory of Evolution and is used in optimization problems. Its importance in the field of computer science include scheduling applications, image processing, travelling salesman problem, and other related subjects [9, 22, 24]. GA seems to have no limits as it also applies on other fields and problems that include design of anti-terrorism systems, estimation of heat flux between the atmosphere and sea ice, and even on the detection of cancer [7, 11, 36].

Simulated Annealing was the first heuristic used to solve TSP until Robert Brady utilized Genetic Algorithm for TSP [5, 22]. Within the same year in 1895, John Grefenstette introduced heuristic crossover while David Goldberg and Robert Lingle introduced partially-mapped crossover [14, 15]. There are other improvements made on the application of GA on TSP throughout the years including the matrix crossover by Abdollah Homaifar and the implementation of adaptive capabilities of crossover and mutation by Srinivas and Patnaik [20, 34]. In 2016, Jiquan Wang introduced the use of a Multi-Offspring Genetic Algorithm (MO-GA) that doubles the size of the offspring produced from crossover as compared to the other GAs that solve TSP [40].

1.2 Statement of the Problem

Converging into an optimum solution and providing diverse solutions are two of the factors that affect the effectivity of a Genetic Algorithm [34]. A balance between both is an ideal way of implementing GA. Multiple-Offspring Genetic Algorithm (MO-GA) utilizes such method on its selection and crossover method [40]. However, it fails to do so on its mutation operator as it uses the traditional way of setting a fixed parameter for its mutation rate. With that, a trial and error is needed to find the optimal parameter, which could lead to tedious work as simulations are to be done every time a parameter is changed.

1.3 Objective of the Study

1.3.1 General Objective of the Study

The general objective of this study is to formulate an algorithm that could aid in finding solutions to Traveling Salesman Problem. Given a set of points, the method should be able to provide the shortest path to visit each point.

With the modifications made on this research, fields that utilize the foundations of TSP could be improved. Better solutions to problems could be obtained, at the possibility of obtaining them at a faster rate.

1.3.2 Specific Objective of the Study

The research aims to improve the Multiple Offspring Genetic Algorithm (MO-GA) by modifying the parameter for mutation. Aside from yielding better solutions, it should also be solved faster by the proposed algorithm as compared to the Basic Genetic Algorithm and the MO-GA. By comparing results of each agorithm side-by-side, assesment on the improvement could be made, as to whether the modification is impactful or not.

1.4 Significance of the Study

Multi-Offspring Genetic Algorithm with Dynamic Mutation (MO-GA with DM) introduces a dynamic mutation rate with the MO-GA could aid in creating the balance required in optimizing and diversifying solutions not only on the crossover method, but also on the mutation method. The effectivity of this algorithm is expected to be at par, or better, than the MO-GA.

The algorithm focuses on providing solution for TSP. It also aims to introduce a hybrid GA that exhausts the adaptive capabilities on two of its methods: selection and mutation.

1.5 Scope and Limitation

This study is focused on obtaining optimum solutions of different maps for the presented algorithm, as well as for Basic Genetic Algorithm and Multiple Offspring Genetic Algorithm. Every computed optimum solution per iteration will be compared for all the algorithms in order to assess which algorithm works best in obtaining a solution efficiently. Also, the program only works in 2d as the data points given are presented with only the x and y coordinates. The algorithm only works on symmetrical TSP as it is assumed that the distances going back and forth are equal.

Chapter 2

Review of Related Literature

This chapter provides an overview on the concepts used in this research. Each subtopic is a discussion on the different concepts that are necessary and vital to the research.

First of all, the literature on combinatorics is discussed. This is to provide context on the possible number of solutions for the problem that is about to be solved. Texts on graph theory follows next. It is necessary as it shows how the problem could be illustrated and presented visually. After this is a discussion on the traveling salesman problem, which discusses the problem that is about to be solved in the paper. The Darwin's Theory of Evolution is also briefly explored as some of the concepts in this theory are used in providing for the solution to the problem. Genetic Algorithm (GA) is presented in this section to provide context on how algorithms like this one finds solutions on specific problems using concepts in biology, like the Darwinian theory. In this literature, the different steps and multiple variations of GA are also discussed. Following this, the literature on Genetic Algorithm on Traveling Salesman Problem (TSP) is presented. This is done in order to show how the Genetic Algorithm is used to solve prolems like the TSP: how it is presented, and the process on how solutions are generation. Lastly, the Multiple Offspring Genetic Algorithm (MOGA) is also shown to introduce a specific type of GA that is utilized in the research.

2.1 Combinatorics

Combinatorics is a young field of mathematics that started its own independence in the 20th century [38]. It deals with the combination and arrangement of sets into patterns that comply with the given rules.

2.1.1 Enumeration

Enumeration deals with counting the number of elements in a finite set [37]. Two major concepts involved in enumeration are permutation and combination.

Permutation is a particular ordering of objects [38]. It frequently asks "Given n objects, how many ways can you order k objects?". Given n, the number of permutations if k is taken at a time is denoted by Equation 2.1. In particular, where n=k, it is denoted by Equation 2.2.

$$P(n,k) = \frac{n!}{(n-k)!} \tag{2.1}$$

where:

n = total number of objects
k = number of objects selected

$$P(n,k) = \frac{n!}{(n-k)!} = \frac{n!}{(n-n)!} = \frac{n!}{0!} = n! \tag{2.2}$$

In particular, Equation 2.2 shows Equation 2.1 if n=k. It is assumed that 0! = 1. For example, given the set $N = \{1,2,3,4,5\}$, how many ways can you arrange the elements if you take 5 elements at a time?

$$P(n,k) = P(5,5) = \frac{5!}{(5-5)!} = \frac{5!}{(0)!} = \frac{5!}{1} = 5!$$

Combination, on the other hand, deals not with the order of the objects, but whether an object is chosen or not [26]. It involves picking items in a given set and disregarding the order. The number of combinations with r objects given an n population is solved by

$$C\binom{n}{k} = \frac{n!}{k!(n-k)!} \tag{2.3}$$

where:

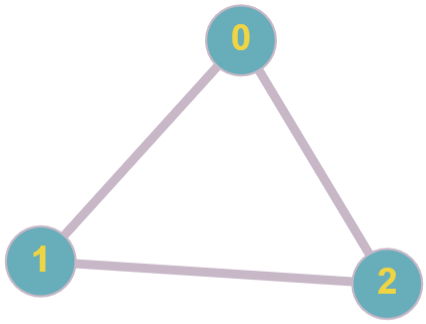
n = total number of objects

k = number of objects selected

2.2 Graph Theory

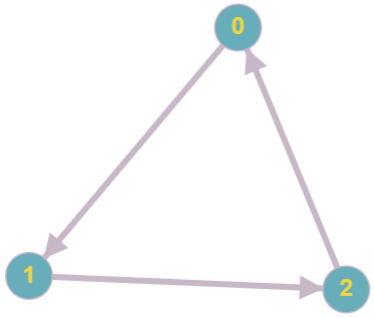
A graph is a set of points called vertices connected by lines known as edges or arcs [1].

Figure 2.1: An example of a graph with 3 vertices and edges.



There are two types of graphs that differ according to the direction of the edges: directed and undirected [1]. A directed graph, as shown on Figure 2.2 is formed by directed edges or arcs, meaning, an edge has a specified direction to where it goes. [30].

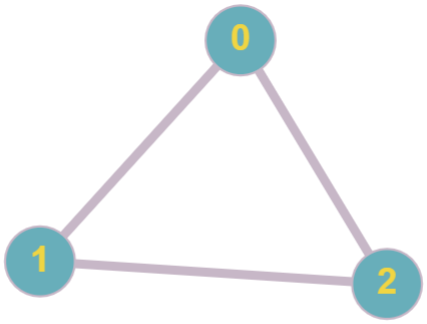
Figure 2.2: An example of a directed graph.



Meanwhile, an undirected graph is formed by edges without a specified direction. It

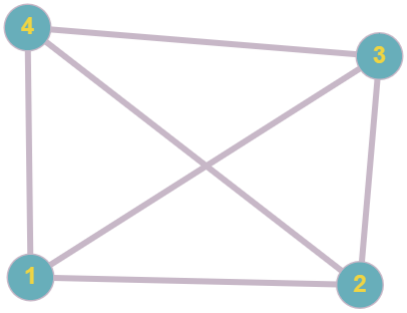
is illustrated on Figure 2.3.

Figure 2.3: An example of an undirected graph with 3 vertices.



Another helpful type of graph is the complete graph. It is formed when all of the possible edges between the vertices are created in the graph [30] and an example is shown on Figure 2.4.

Figure 2.4: An example of a complete graph with 4 vertices.



Some fundamental concepts in graph theory involve paths and cycles [6]. A path is a walk within a graph where no vertex is repeated. For example, in Figure 2.4, $1 \rightarrow 2 \rightarrow 3$ is a path. Since no vertex is repeated, it automatically follows that no edge is visited more than once. Meanwhile, a cycle is a walk whose starting vertex will be its ending vertex. From Figure 2.4, $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 1$ is a cycle.

More advanced definitions in the graph theory include the Hamiltonian path and the Hamiltonian cycle. A Hamiltonian path is a path that contains all the vertices given

in a graph [6]. Based on Figure 2.5, $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5$ is a hamiltonian path and it is shown on Figure 2.6. Meanwhile, a Hamiltonian cycle is a cycle that contains all the vertices in a given graph [6]. Using Figure 2.5, $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 1$ is a hamiltonian cycle and is illustrated on Figure 2.7.

Figure 2.5: A graph with 5 edges and vertices.

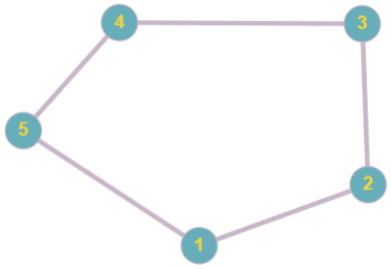


Figure 2.6: A hamiltonian path.

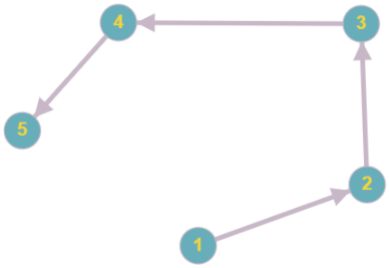
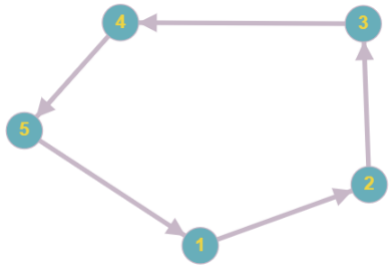


Figure 2.7: A hamiltonian cycle.



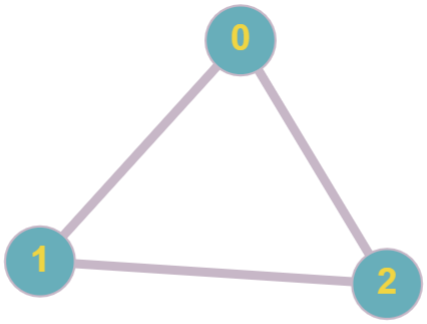
Note that $1 \rightarrow 2 \rightarrow 3 \rightarrow 4 \rightarrow 5 \rightarrow 1$ and $3 \rightarrow 4 \rightarrow 5 \rightarrow 1 \rightarrow 2 \rightarrow 3$ are the same Hamiltonian cycles.

2.3 Traveling Salesman Problem

The Traveling Salesman Problem (TSP) is a combinatorial problem that was defined in the 1930s by Karl Menger [32]. It answers the question "Given a set of cities, what is the shortest possible route to be taken in order to visit all the cities and return to the origin?" [28]. Its solution is a hamiltonian cycle [6].

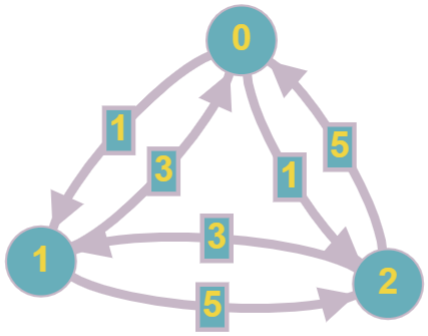
There are two major types of TSP, the symmetric and asymmetric [25]. In a symmetric TSP, the path between two cities has no specified directions. It means that the cost of traveling between two cities, even if the route is on a different direction, is the same. It is denoted by undirected graphs. In Figure 2.8, path $0 \rightarrow 1$ is equal to path $1 \rightarrow 0$.

Figure 2.8: An example of a Symmetric TSP with 3 cities.



Meanwhile, in asymmetric TSP, the cost of traveling between cities on different directions are not equal. It is illustrated by directed graphs. In Figure 2.9, path $0 \rightarrow 1$ costs 1 unit. However, path $1 \rightarrow 0$ costs 3 units.

Figure 2.9: An example of an Asymmetric TSP with 3 cities.



The TSP is a permutation problem [17]. It deals with the arrangement or order of the cities and therefore, the search space increases gradually every time a city is added. Some say it is NP-Hard while others claim it is an NP-Complete [19, 32].

Using brute-force approach, the running time will lie within a polynomial factor of $\mathcal{O}(n!)$ because given a list of n cities, the size of the search space in a general case is $(n-1)!$ [18, 21]. However, $\frac{(n-1)!}{2}$ is used to solve for the search space of symmetric TSP [33]. It is because in symmetric TSP, a route can be represented in two ways.

2.4 Darwin's Theory of Evolution

Charles Darwin's "Theory of Evolution" is a widely accepted concept which states that all life descended from a common ancestor, and are therefore related [2]. It presumes that descendants evolve with more complexities compared to their ancestors. Variation and the struggle for existence are the two characteristics that result in biological changes [41].

Variations occur within individuals in the population [41]. The varying traits of these individuals are passed on to the next generations. Existential competition preserves the advantageous traits and removes the inferior ones in order to improve their potential to exist.

The concept of Natural Selection is included within this theory [41]. It was stated

that with the varying traits of individuals, those with slight advantageous traits will have a higher chance of survival as compared to the individuals who do not posses such traits. Furthermore, these advantageous traits can be passed on to their offspring.

2.5 Genetic Algorithm

In computer science, there are some known algorithms that follow the laws of natural evolution [22]. These probabilistic search algorithms are known as Evolutionary Algorithms (EA) and were proposed in the late 1960s. After a few years, these algorithms were used to optimize solutions for NP-Hard problems which were applied in different fields such as Biology, Chemistry, Cryptoanalysis, Pattern Recognition, etc.

The Genetic Algorithm (GA), introduced by John Holland in 1975, is one of the most commonly used EAs [29]. It was mainly based on Darwin's Theory of Evolution and is used in optimization problems. It is often characterized by the following [34]:

- genetic representation
- population of generated solutions
- fitness function that evaluates each solution
- genetic operators that creates new population based on the existing one
- control parameters

In this algorithm, one solves a solution in the form of a string of numbers, which is referred to as chromosomes [22]. The main goal here is to find the best chromosome that will give us the optimal solution to the given problem. The group of chromosomes that will be worked on will be known as the population.

Just like in Darwin's Theory of Evolution, the population will compete, mate, and sometimes, mutate, through time [23]. Due to natural selection, advantageous traits and individuals will be preserved for the next generations. This will result in more fit individuals and eventually, will lead to the domination of superior individuals within the population.

Pseudocode for GA is as presented on Algorithm 1 [4]:

Algorithm 1 Psudocode for Genetic Algorithm

```
BEGIN GA
Initialize Population
while Terminating Condition/s do
    Select Parents from Population
    Produce offspring from selected parents
    Mutate individuals
    Add offspring to the population
    Reduce the population to the original size
end while
Show Output
END GA
```

2.5.1 Initializing the Population

First, an initial population will be generated. This population is composed of a set of strings or individuals known as chromosomes [29]. A chromosome is composed of genes. These chromosomes are represented in different ways, the most popular are the binary and numerical or non-binary representations [12].

In Binary representation, the chromosomes are given by genes that contain the numbers 0,1. For example:

010 110 101

Meanwhile, in a non-binary representation, the real values of the genes are used.

2 6 5

2.5.2 Selection

After initializing the population, the selection process follows [23]. The selection process is a way of determining which solutions are to be preserved and will yield offspring [3] [23]. Its main goal is to preserve the better solutions while discarding the bad ones while maintaining the population size. A fitness function is described in order to determine the good solutions within the population. This varies depending on the problem. The fitness value is calculated for each individual, and will be compared with

each other to determine the most fit individuals or as Darwin calls it, "the survival of the fittest.

There are several ways to implement the selection process, and it includes roulette wheel, ranking, tournament, and steady-state [43]. In Roulette Wheel, all individuals within the population are placed in a roulette wheel wherein their fitness value or rank is proportional to their portion in the wheel. Individuals with the higher fitness values have a better chance of getting picked. The roulette is spun until the desired number of individuals is picked.

Common processes in the different selection implementations are [43]:

- Identifying the best individuals within the population.
- Duplicating the best solutions.
- Disregarding the least fit solutions to the problem.

Sometimes, some individuals from the original population are carried out to be a part the next population [23]. This is known as elitism.

2.5.3 Crossover

After the parents will be chosen for each iteration in the selection, they will produce an offspring that will be added to the population. This process is known as the crossover. This process is mimics the reproduction process of living things as it is the step where there is an exchange of genes within individuals [35]. For example, given the individuals:

$P_1 = [4, 1, 3, 2]$ and $P_2 = [6, 2, 7, 3]$

as parents, a crossover would mean that they will each exchange genes in order to create an offpring, depending on the given process. For this one, we will assume that they will exchange their last digits to create their offspring. Replacing the last digit of P_1 into P_2 will yield $C_1 = [6, 2, 7, 2]$, and replacing the last digit of P_2 into P_1 will yield $C_2 = [4, 1, 3, 3]$.

1-point crossover, k-point crossover, and shuffle crossover are some of the variations in implementing crossover.

2.5.4 Mutation

After the offspring are generated, these new individuals has a chance of mutation [12] [16]. The mutation process randomly alters an individual's genes. It is done so to prevent the population from converging fast and to avoid getting stuck in a local optima.

An example of a mutation is the flip mutation. It is applicable when binary representation is used. It is done by flipping the genes from 0 to 1, and vice versa. For example, given the string

010 110 101

If it undergoes flip mutation, it will be

101 001 010

After a series of crossovers and mutations, the new population will be evaluated and the most fit individuals based on the criteria given will proceed to the next iteration [12]. An iteration will be known as a generation.

Stopping criteria varies on the problem. There are three criteria for stopping that are commonly used in GA [31]:

- a limit to the number of generations reached
- a limit to the number of evaluations of the fitness function is reached
- or low significant changes with the next generations

2.5.5 Parameter Setting

There are some parameters to be considered in evolutionary algorithms like the GA. Selection, crossover, and mutation rates are the most common parameters needed in the said process.

The traditional genetic algorithm uses constant values for each parameter [8]. While constant parameters are proven effective to problems, optimum values may be require a

lot of tests as they have to be experimentally plugged into the system to figure out which values work best. With this, parameter controls are introduced [10].

An example of parameter control is the dynamic mutation where the mutation rate is not represented by a fixed value, but rather, an equation involving the fitness values of the population [42]. On Equation 2.4, m is given as a function involving the average fitness of the population, F_{ave}, and the current optimum solution of the population, $F_{optimum}$. The mutation rate will be high if there is minimum diversity among the population and it is low of the population is diverse.

$$m = (1 - \frac{F_{ave} - F_{optimum}}{F_{optimum}})^k \tag{2.4}$$

where:

m = mutation rate
F_{ave} = average fitness of the population
$F_{optimum}$ = average fitness of the population
k = control for change in amplitude

2.6 Genetic Algorithm on Travelling Salesman Problem

Numerous heuristics were used to solve TSP [22]. Simulated Annealing was the first heuristic used to solve it. Robert Brady was the first researcher to tackle TSP with GA in 1985.

2.6.1 Representations

There are varying representations used to solve TSP using GA: Binary, Path, Adjacency, Ordinal, and Matrix [22].

Binary representation uses binary numbers in representing the cities. For example, if a path is 3-5-2-7, the chromosome will be given by 011-101-010-111. In Path representation,

given an n number of cities, a chromosome will be of size n. The order of the path will be the order of the numbers representing the cities in the chromosome. For example, if we have cities 1-9, and given the string [3 1 4 2 6 8 5 7 9], city 3 will be visited first, followed by 1, then 4, and so on until a cycle is complete.

2.6.2 Crossover

For the crossover, there are some variations used throughout history [22]. One example is the Order crossover. It was proposed by Davis in 1985 and it stresses that the order of cities, not their positions, are important. For example

P_1 = [1 2 3 4 5 6 7 8 9]
P_2 = [9 3 4 2 1 5 6 8 7]

First, it selects two random cut points. For example, the first cut will be done in between 3rd and 4th element, while the second cut will be made between the 7th and 8th element. It will look like this.

P_1 = [1 2 3 | 4 5 6 7 | 8 9]
P_2 = [9 3 4 | 2 1 5 6 | 8 7]

Where the middle sections will be preserved, and be put into the offprings.

C_1 = [x x x | 4 5 6 7 | x x]
C_2 = [x x x | 2 1 5 6 | x x]

And then, starting from the second cut point of a parent, the cities are copied in order from the other parent, also starting from the second cut point while removing the cities that already exist within the string.

C_1 = [x x x | 4 5 6 7 | 8 x]

Since the second cut of the second parent starts with 8, and 8 does not yet exist within the future offspring, we add it to the string.

C_1 = [x x x | 4 5 6 7 | 8 9]

Since the next number of the second parent is 7, and it exists within the future offspring, we do not add it. Since it is the end of the string for P_2, we proceed to the first number on the string. Since 9 is the 1st number on P_2, and 9 is not yet present within the child string, we add 9 to the last vacant position.

C_1 = [3 x x | 4 5 6 7 | 8 9]

After 9, the next number on P_2 if we traverse it in a linear way is 3, which is non existent within the child string. Since we are done filling the numbers on the second cut of C_1, we will not fill the empty slots before the 1st cut. The process goes on until we get

C_1 = [3 2 1 | 4 5 6 7 | 8 9]

The same process will be done for the second offspring. It's middle chunk will be from P_2 and its remaining digits will come from P_1 and will be placed in a way the digits of C_1 is filled. it will result in:

C_2 = [3 4 7 | 2 1 5 6 | 8 9]

The resulting children will be:

C_1 = [3 2 1 4 5 6 7 8 9]

C_2 = [3 4 7 2 1 5 6 8 9]

Aside from this, other crossover methods include Partial-Mapping by Goldberg and Lingle , Cycle crossover by Oliver et. al, Position Based Crossover by Syswerda, and some more [22].

2.6.3 Mutation

For the mutation, different variations are also present like the Simple Inversion Mutation by Holland, Displacement mutation by Michalewizc and Exchange mutation by Banchaf [22].

The Simple Inversion Mutation (SIM) [22] was proposed by Holland in 1975. It chooses two random cuts from an individual, and reverses the order of the middle chunk. For example.

$C_1 = [3\ 4\ 7\ 2\ 1\ 5\ 6\ 8\ 9]$

Assuming that the first cut point is between the 3rd and 4th numbers, and the second cut point is between the 7th and 8th numbers.

We now have

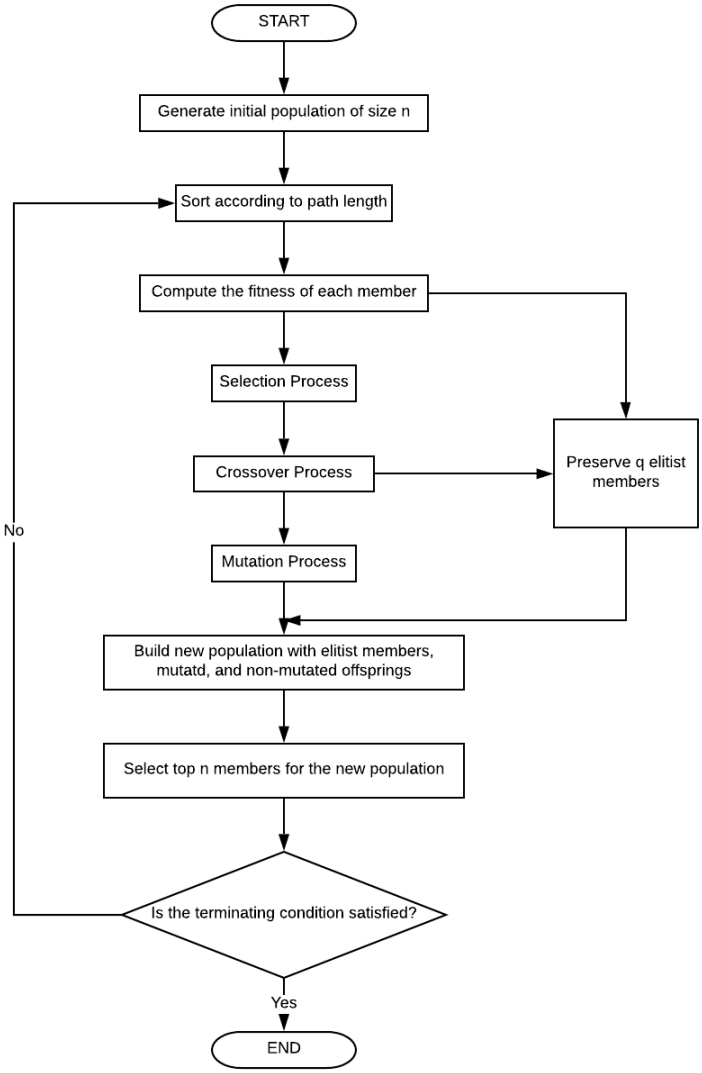
$C_1 = [3\ 4\ 7 \mid 2\ 1\ 5\ 6 \mid 8\ 9]$

Reversing the order of the middle chunk will give us

$C_1 = [3\ 4\ 7 \mid 6\ 5\ 1\ 2 \mid 8\ 9]$
$C_1 = [3\ 4\ 7\ 6\ 5\ 1\ 2\ 8\ 9]$

The processes repeats itself until the terminating condition is satisfied.

Figure 2.10: Flowchart for the GA on TSP



2.7 Multiple Offspring Genetic Algorithm

There are three conditions for natural selection: variation, inheritance, and competition [13]. Variation refers to the differing characteristics of individuals within the population. Inheritance refers to the traits that are passed on from parents to offspring. Competition refers to the occasion where more fit and competitive individuals survive.

With that being said, we can infer that a larger population contains a lot of variations.

And with the existence of a greater variety of individuals, a more competitive population will exist. This is the biological theory that led to the foundation of the Multiple Offspring Genetic Algorithm (MO-GA) from the paper titled "Multi-offspring genetic algorithm and its application to the traveling salesman problem" by Jiquan Wang, Okan Ersoy, Mengying He, and Fulin Wang.

The MO-GA is a special Genetic Algorithm wherein instead of creating 2 offspring from 2 parents, 4 offspring will be generated.

An individual will be represented by a string of numbers. Path representation will be used in this method. The numbers indicate the city number and the order within the string represent the order of the cities. An individual, which represents the route, will be randomly generated and it will contain a string of non-repeating numbers representing the cities. For example, if there are 5 cities, a string of size 5 of non-repeating numbers from 1-5 will be created, e.g. 3, 2 ,4, 1, 5.

2.7.1 Initializing the Population

For the initial population, 100 individuals will be randomly generated. There is no guarantee that an individual is unique as duplicates may also be randomly generated.

2.7.2 Selection

In determining the individuals who will become parents, the roulette wheel method is used. For this method, the individual fitness and probability of each individual to be picked is established.

There are two models used in solving the fitness of an individual. The first one is

$$F_1(X_i') = \beta(1-\beta)^{i-1}, i = 1, 2, ...n \tag{2.5}$$

where:

n = the population size

X_i = the ith member of the population when arranged in ascending order

β = parameter that $\in (0, 1)$ and is usually $\beta = (0.01, 0.3)$ [39]

The second model is given by

$$F_2(X_i') = \frac{n - i + 1}{n}, i = 1, 2, ...n \tag{2.6}$$

where:

n = the population size

If the number of generation is odd, Equation 2.6 will be used, else, Equation 2.5 will be utilized. In choosing the crossover members or parents, given the fitness value $F(X_i')$, the probability of an ith member to be selected is denoted by Equation 2.7 where PP_0 is given by Equation 2.8 and PP_i is given by Equation 2.9.

$$P_i = \frac{F(X_i')}{\sum_{i=1}^{n} F(X_i')} \tag{2.7}$$

$$PP_0 = 0 \tag{2.8}$$

$$PP_i = \sum_{j=1}^{i} P_i, i = 1, 2, ...n \tag{2.9}$$

where:

$F(X_i')$ = the fitness value of an individual
n = the population size
P_i = the probability of an individual to be selected.

A random number n_k is selected from (0,1) for 2n times. If the random number satisfies Equation 2.10, the ith member will be selected for crossover.

$$PP_{i-1} < n_k < PP_i \tag{2.10}$$

2.7.3 Crossover and Mutation

In the basic GA, 2 parents generate 2 offspring. In MO-GA, 2 parents generate 4 offspring during the crossover method. Since the offspring will be twice the original GA can produce, two crossover methods will be used. The first one is the Order crossover described in Section 2.6.2, while the second one will be as follows :

Consider the two strings

$P_1 = [\ 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9]$
$P_2 = [\ 9\ 3\ 4\ 2\ 1\ 5\ 6\ 8\ 7]$

Again, there will be two cut points.

$P_1 = [\ 1\ 2\ 3\ |\ 4\ 5\ 6\ 7\ |\ 8\ 9]$
$P_2 = [\ 9\ 3\ 4\ |\ 2\ 1\ 5\ 6\ |\ 8\ 7]$

From this, we switch the first and second subtours of each parent

$PP_1 = [\ 4\ 5\ 6\ 7\ |\ 1\ 2\ 3\ |\ 8\ 9]$
$PP_2 = [\ 2\ 1\ 5\ 6\ |\ 9\ 3\ 4\ |\ 8\ 7]$

Swapping the 3rd subtours of each parent while copying the rest in order and omitting the duplicates, we get

$C_3 = [\ 4\ 5\ 6\ 1\ 2\ 3\ 9\ 8\ 7]$
$C_4 = [\ 2\ 1\ 5\ 6\ 3\ 4\ 7\ 8\ 9]$

After obtaining offspring, a mutation rate established earlier will be used to determine if certain offspring will mutate or not. Each offspring will be given a random number $m = [0, 1]$. If the number assigned to an individual is less than or equal to m, then that individual will mutate. The mutation process used is the Inversion as described on

Section 2.6.3.

After the crossover and mutation process, the selection for the next generation follows. Elitism is implemented on this, wherein a specified number of most fit individuals from the parent population, will not be replaced and be carried on to the next generation [44].

The process repeats itself until a terminating condition is satisfied. The step-by-step process for the MO-GA is presented on Algorithm 2.

Algorithm 2 Multi Offspring Genetic Algorithm

BEGIN MOGA
Initialize random population with size n
while Terminating Condition/s **do**
  for i=0; $i < n$; i++ **do**
    Compute for the path length of each individual
  end for
  Arrange individuals in ascending order based on path lengths
  Select the q highest ranking individuals in the population
  for i=0; $i < n$; i++ **do**
    Using the selection and crossover operations, generate offspring
  end for
  Create a new population by combining the q elitist members selected and $2n$ offpsrings generated.
  Compute for the path lengths of each individual and arrange them in ascending order
  From the new population, select new q individuals with the highest rank
  for i=0; $i < 2n$; i++ **do**
    Modify the offsprings using the mutation operator
  end for
  Create a new population consisting of the $2n$ mutated and unmutated offspring and the latest q individuals
  Arrange the members of the new population in ascending order according to their path lengths
  Select the n highest ranking individuals as the new population
end while
Show Output
END MOGA

Chapter 3

Methodology

A balance between converging into an optimum solution and providing diverse solutions is needed for a GA to work properly [34]. While MO-GA could provide both with its crossover method, its mutation rate could be considered weak at maintaining the said balance due to it being set as a fixed value.

With this proposed algorithm called "Multi-offspring Genetic Algorithm with Dynamic Mutation (MO-GA with DM)", the concepts and procedures of MO-GA are utilized, however, the constant mutation rate will be replaced by one that changes depending on the stability of the population. It follows the step-by-step process of MO-GA as discussed on section 2.7, with the insertion of the computation for a mutation rate, instead of using a fixed rate.

The step-by-step process for the MO-GA with Dynamic Mutation is summarized on Algorithm 3.

Algorithm 3 Multi Offspring Genetic Algorithm with Dynamic Mutation

BEGIN MOGA with DM

Initialize random population with size n

while Terminating Condition/s **do**

for i=0; $i < n$; i++ **do**

Compute for the path length of each individual

end for

Arrange individuals in ascending order based on path lengths

Select the q highest ranking individuals in the population

for i=0; $i < n$; i++ **do**

Using the selection and crossover operations, generate offspring

end for

Create a new population by combining the q elitist members selected and $2n$ offpsrings generated.

Compute for the path lengths of each individual and arrange them in ascending order

From the new population, select new q individuals with the highest rank

Based on the current population, compute for the mutation rate.

for i=0; $i < 2n$; i++ **do**

Modify the offsprings using the mutation operator

end for

Create a new population consisting of the $2n$ mutated and unmutated offspring and the latest q individuals

Arrange the members of the new population in ascending order according to their path lengths

Select the n highest ranking individuals as the new population

end while

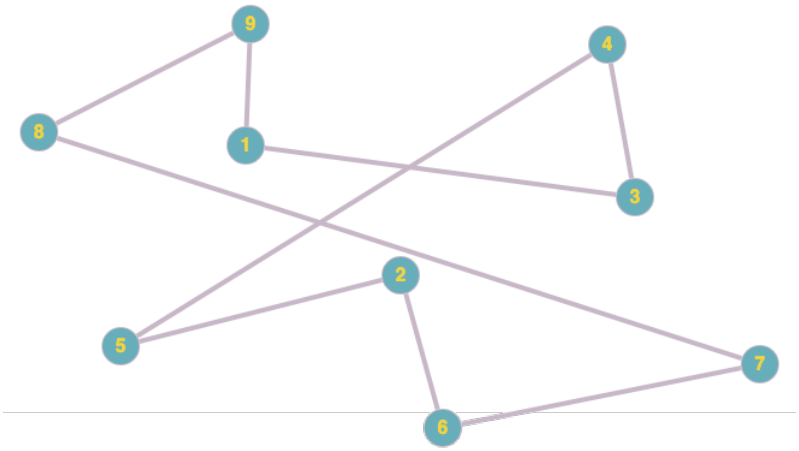
Show Output

END MOGA with DM

3.1 Representation

Path representation will be used for this method. The cities are assigned a number that represents them and each city contains x and y coordinates. The individuals of each population contain an array of cities and they represent the route taken. For example if an individual is denoted by $A_1 = [1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9]$, its path can be visualized in Figure 3.1.

Figure 3.1: A sample representation for an individual in MO-GA with DM.



3.2 Initialization

The initial number of the individuals in the population will be given by n, which is a non-negative constant number defined at the beginning of the algorithm. The order of the cities in every individual is generated randomly. For a map with c number of cities, numbers 1 to c are generated without repetition for c times. In the earlier example from Section 3.1, for a map with 9 cities, A_1 is generated randomly. The random generation of individuals is repeated n times to complete the population number of n. For example, we generate a population where n=10 and c=9:

$A_1 = [\ 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9]$

$A_2 = [\ 9\ 34\ 2\ 5\ 1\ 6\ 8\ 7]$

$A_3 = [\ 2\ 3\ 1\ 4\ 5\ 6\ 7\ 8\ 9]$

A_4 = [3 4 1 2 5 6 7 8 9]
A_5 = [5 1 6 3 2 4 7 8 9]
A_6 = [9 2 4 3 5 6 1 8 7]
A_7 = [8 2 3 4 7 6 5 1 9]
A_8 = [4 2 3 1 5 9 7 8 6]
A_9 = [9 3 4 2 1 5 6 8 7]
A_{10} = [7 1 5 4 3 6 2 8 9]

3.3 Computation for Path Length and Arrangement

The path length of each individual is the main basis for its fitness. Its path length is the summation of the distances between the cities. With the current representation, the distance between two cities i and j with coordinates given by (x_i, y_i) and (x_j, y_j) is given by Equation 3.1.

$$D_{ij} = ((x_i - x_j)^2 + (y_i - y_j)^2)^{0.5} \tag{3.1}$$

where:

D_{ij} = distance between cities i and j
x_i = x-coordinate for city i
x_j = x-coordinate for city j
y_i = y-coordinate for city i
y_j = y-coordinate for city j

With the example individual N_1 = [1 3 4 5 2 6 7 8 9] from section 3.1 with 9 cities, its total path length can be computed by Equation 3.2.

$$D_{Total} = D_{12} + D_{23} + D_{34} + D_{45} + D_{56} + D_{67} + D_{78} + D_{89} + D_{91} \tag{3.2}$$

Assuming that we have computed the path lengths of the individuals given in section 3.2, we arrange them from shortest to longest path, we then have Table 3.1.

Individual	Route Length
A_1	35
A_2	39
A_3	42
A_4	43
A_5	50
A_6	55
A_7	56
A_8	64
A_9	70
A_{10}	71

Table 3.1: Path Lengths for Example

3.4 Preservation of q elitist members

In order to preserve the best individuals of the population, selecting and saving them for later is done. The number of individuals to be chosen is given by q which is an arbitrary number that is less than n. For the example, we let $q = 5$, therefore, the best 5 individuals will be selected and saved for later use and they comprise of A_1 to A_5.

$A_1 = [\ 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9]$

$A_2 = [\ 9\ 3\ 4\ 2\ 5\ 1\ 6\ 8\ 7]$

$A_3 = [\ 2\ 3\ 1\ 4\ 5\ 6\ 7\ 8\ 9]$

$A_4 = [\ 3\ 4\ 1\ 2\ 5\ 6\ 7\ 8\ 9]$

$A_5 = [\ 5\ 1\ 6\ 3\ 2\ 4\ 7\ 8\ 9]$

3.5 Selection and Crossover

The selection and crossover process comes next. Two individuals will be selected at a time to act as parents. The fitness of each individual is given by 2 equations which are used alternately. Equation 3.3 is used when the current generation (iteration) is an

odd number. Meanwhile, Equation 3.4 is utilized when the current generation is an even number.

$$F_1(X_i') = \beta(1-\beta)^{i-1}, i = 1, 2, ...n \tag{3.3}$$

n = the population size
X_i = the ith member of the population when arranged in ascending order
β = parameter that $\in (0, 1)$ and is usually $\beta = (0.01, 0.3)$ [39]

$$F_2(X_i') = \frac{n-i+1}{n}, i = 1, 2, ...n \tag{3.4}$$

n = the population size

Given the fitness value $F(X_i')$, the probability of an ith member to be selected is given by Equation 3.5, letting PP_0 to be given by Equation 3.6 and PP_i to be given by Equation 3.7.

$$P_i = \frac{F(X_i')}{\sum_{i=1}^{n} F(X_i')} \tag{3.5}$$

$$PP_0 = 0 \tag{3.6}$$

$$PP_i = \sum_{j=1}^{i} P_i, i = 1, 2, ...n \tag{3.7}$$

where:

$F(X_i')$ = the fitness value of an individual
n = the population size
P_i = the probability of an individual to be selected.

A parent is chosen by generating a random number n_k from (0,1) and is determined by Equation 3.8. If n_k satisfies Equation 3.8,then the ith member will be chosen. 2 parents are selected at a time, so two n_k are generated.

$$PP_{i-1} < n_k < PP_i \tag{3.8}$$

Assume we have already computed for the P_i and PP_i for all the individuals in the population and it is shown on Table 3.2. Suppose we have n_k values 0.36, and 0.54, then we have individuals A_1 and A_2 as parents.

Individual	Route Length	$F(X_i')$	PP_i
A_1	35	0.300	0.309
A_2	39	0.210	0.525
A_3	42	0.147	0.676
A_4	43	0.103	0.782
A_5	50	0.072	0.856
A_6	55	0.050	0.908
A_7	56	0.035	0.944
A_8	64	0.025	0.970
A_9	70	0.017	0.988
A_{10}	71	0.012	1

Table 3.2: Fitness Values of Example

The first two children are generated by using the order crossover discussed on section 2.6.2. For this example, given that A_1 and A_2 are selected as parents, respectively, the offspring can be computed as follows:

1. First, it selects two random cut points. A cut point is a random number generated from 1 to c-1 that slices the individuals into subtours. Since we are selecting 2 random cut points, they must be non-repeating and it creates. For example, if we get random cut points 3 and 7, the first cut will be done in between 3rd and 4th element, while the second cut will be made between the 7th and 8th element. It will look like this.

$P_1 = A_1 = [\ 1\ 2\ 3 \mid 4\ 5\ 6\ 7 \mid 8\ 9]$

$P_2 = A_2 = [\ 9\ 3\ 4 \mid 2\ 1\ 5\ 6 \mid 8\ 7]$

2. The middle sections will be preserved, and be put into the offpring.

$C_1 = [\ x\ x\ x \mid 4\ 5\ 6\ 7 \mid x\ x]$

$C_2 = [\ x\ x\ x \mid 2\ 1\ 5\ 6 \mid x\ x]$

3. And then, starting from the second cut point of a parent, the cities are copied in order from the other parent, also starting from the second cut point while removing the cities that already exist within the string.

$C_1 = [\ x\ x\ x \mid 4\ 5\ 6\ 7 \mid 8\ x]$

Since the second cut of the second parent starts with 8, and 8 does not yet exist within the future offspring, we add it to the string.

$C_1 = [\ x\ x\ x \mid 4\ 5\ 6\ 7 \mid 8\ 9]$

Since the next number of the second parent is 7, and it exists within the future offspring, we do not add it. Since it is the end of the string for P_2, we proceed to the first number on the string. Since 9 is the 1st number on P_2, and 9 is not yet present within the child string, we add 9 to the last vacant position.

$C_1 = [\ 3\ x\ x \mid 4\ 5\ 6\ 7 \mid 8\ 9]$

After 9, the next number on P_2 if we traverse it in a linear way is 3, which is non existent within the child string. Since we are done filling the numbers on the second cut of C_1, we will not fill the empty slots before the 1st cut. The process goes on until we get

$C_1 = [\ 3\ 2\ 1\ |\ 4\ 5\ 6\ 7\ |\ 8\ 9]$

4. The same process will be done for the second offspring. It's middle chunk will be from P_2 and its remaining digits will come from P_1 and will be placed in a way the digits of C_1 is filled. it will result in:

   $C_2 = [\ 3\ 4\ 7\ |\ 2\ 1\ 5\ 6\ |\ 8\ 9]$

   The resulting children will be:

   $C_1 = [\ 3\ 2\ 1\ 4\ 5\ 6\ 7\ 8\ 9]$

   $C_2 = [\ 3\ 4\ 7\ 2\ 1\ 5\ 6\ 8\ 9]$

For the third and fourth children, the process utilized is the same as discussed on section 2.7.3. For this given example, it is generated as follows:

1. The same parents from the earlier crossover method will be used. Also, the same cutpoints will be utilized for this process. So we have the following parents with their cutpoints.

   $P_1 = [\ 1\ 2\ 3\ |\ 4\ 5\ 6\ 7\ |\ 8\ 9]$

   $P_2 = [\ 9\ 3\ 4\ |\ 2\ 1\ 5\ 6\ |\ 8\ 7]$

2. From this, we switch the first and second subtours of each parent

   $PP_1 =$ [4 5 6 7 | 1 2 3 | 8 9]

   $PP_2 =$ [2 1 5 6 | 9 3 4 | 8 7]

3. Swapping the 3rd subtours of each parent while copying the rest in order and omitting the duplicates, we get

   $C_3 =$ [4 5 6 1 2 3 9 8 7]

   $C_4 =$ [2 1 5 6 3 4 7 8 9]

After the crossover process, we have generated 4 offspring out of 2 parents. The crossover process will be done for $n/2$ times so it is ideal to have an n that is an even number. When the whole crossover process is finished, $2n$ offspring are generated from n parents. Assuming we have finished generating all offspring, we have the C list of offspring. par

$C_1 =$ [3 2 1 4 5 6 7 8 9]
$C_2 =$ [3 4 7 2 1 5 6 8 9]
$C_3 =$ [4 5 6 1 2 3 9 8 7]
$C_4 =$ [2 1 5 6 3 4 7 8 9]
$C_5 =$ [5 1 6 3 2 4 7 8 9]
$C_6 =$ [9 2 4 3 5 6 1 8 7]
$C_7 =$ [8 2 3 4 7 6 5 1 9]
$C_8 =$ [4 2 3 1 5 9 7 8 6]
$C_9 =$ [9 3 4 2 1 5 6 8 7]
$C_{10} =$ [7 1 5 4 3 6 2 8 9]
$C_{11} =$ [1 2 4 3 5 6 7 8 9]
$C_{12} =$ [2 3 1 6 5 4 7 8 9]

C_{13} = [2 1 3 4 5 6 7 8 9]
C_{14} = [3 1 4 2 5 6 7 8 9]
C_{15} = [5 6 1 3 2 4 7 8 9]
C_{16} = [2 9 4 3 5 6 1 8 7]
C_{17} = [3 2 8 4 7 6 5 1 9]
C_{18} = [7 2 3 1 5 9 4 8 6]
C_{19} = [9 3 4 2 1 5 6 7 8]
C_{20} = [7 5 1 4 3 6 2 8 9]

3.6 Preserving another set of q elitist members

After the $2n$ offspring have been generated, we combine the offspring population with the q elitist members that were selected on section 3.4, compute for their individual path length, and arrange them from shortest to longest route lengths. Assuming we have combined and arranged them, we have the following population:

C_1 = [3 2 1 4 5 6 7 8 9]
A_1 = [1 2 3 4 5 6 7 8 9]
C_2 = [3 4 7 2 1 5 6 8 9]
A_2 = [9 3 4 2 5 1 6 8 7]
C_3 = [4 5 6 1 2 3 9 8 7]
A_3 = [2 3 1 4 5 6 7 8 9]
C_4 = [2 1 5 6 3 4 7 8 9]
A_4 = [3 4 1 2 5 6 7 8 9]
C_5 = [5 1 6 3 2 4 7 8 9]
A_5 = [5 1 6 3 2 4 7 8 9]
C_6 = [9 2 4 3 5 6 1 8 7]
C_7 = [8 2 3 4 7 6 5 1 9]
C_8 = [4 2 3 1 5 9 7 8 6]
C_9 = [9 3 4 2 1 5 6 8 7]
C_{10} = [7 1 5 4 3 6 2 8 9]
C_{11} = [1 2 4 3 5 6 7 8 9]

C_{12} = [2 3 1 6 5 4 7 8 9]
C_{13} = [2 1 3 4 5 6 7 8 9]
C_{14} = [3 1 4 2 5 6 7 8 9]
C_{15} = [5 6 1 3 2 4 7 8 9]
C_{16} = [2 9 4 3 5 6 1 8 7]
C_{17} = [3 2 8 4 7 6 5 1 9]
C_{18} = [7 2 3 1 5 9 4 8 6]
C_{19} = [9 3 4 2 1 5 6 7 8]
C_{20} = [7 5 1 4 3 6 2 8 9]

From the newly generated population, we again select the q highest ranking members. In the example, q is 5, therefore, the five highest ranking individuals will be saved for later use. The five highest ranking members are:

$Q_1 = C_1$ = [3 2 1 4 5 6 7 8 9]
$Q_2 = A_1$ = [1 2 3 4 5 6 7 8 9]
$Q_3 = C_2$ = [3 4 7 2 1 5 6 8 9]
$Q_4 = A_2$ = [9 3 4 2 5 1 6 8 7]
$Q_5 = C_3$ = [4 5 6 1 2 3 9 8 7]

3.7 Computation for Mutation Rate

The mutation rate will be based on the stability of the population, meaning, it is dependent on the ratio of the average fitness of the population and the optimum fitness. It is given on Equation 3.10 [42]. F_{ave} is the average path length of the population given by Equation 3.9 where F_i is the path length of an individual and n is the number of individuals within the population. Meanwhile $F_{optimum}$ is the shortest path length in the population.

$$F_{ave} = \frac{\sum_{i=1}^{n} F_i}{n} \tag{3.9}$$

where:

F_{ave} = average fitness of the population

F_i = fitness of an individual
n = total number of individuals in the population

$$m = (1 - \frac{F_{ave} - F_{optimum}}{F_{optimum}})^k \tag{3.10}$$

where:

m = mutation rate
F_{ave} = average fitness of the population
$F_{optimum}$ = average fitness of the population
k = control for change in amplitude

Assuming that the route lengths for the new population on Section 3.6 is given on Table 3.3, F_{ave} is 50.76 and the $F_{optimum}$ is the path length of C_1 which is 34. Therefore m is 0.507.

Individual	Route Length
C_1	34
A_1	35
C_2	66
A_2	39
C_3	54
A_3	42
C_4	70
A_4	43
C_5	41
A_5	50
C_6	41
C_7	34
C_8	66
C_9	54
C_{10}	70
C_{11}	41
C_{12}	34
C_{13}	66
C_{14}	54
C_{15}	70
C_{16}	41
C_{17}	34
C_{18}	66
C_{19}	54
C_{20}	70

Table 3.3: Fitness Values of New Population

3.8 Mutation Process

The *2n* offspring will then undergo the mutation process. For this once, each individual generated a random number p_m from (0,1). If $p_m \leq m$, then the individual mutates. Else, the individual retains itself. The mutation process is only terminated once all the offspring have undergone mutation or retention depending on the p_m that was randomly assigned to them.

The mutation process used is the simple inversion mutation described on section 2.7.3. For the example, assuming that C_1 has generated p_m of 0.45, it will undergo mutation as the m computed earlier from section 3.7 is 0.5, and 0.45 ¡ 0.5. Its mutation will be as follows:

1. We have the offspring C_1 given by:

   C_1 = [3 2 1 4 5 6 7 8 9]

   We again generate 2 random cut points. Assuming that the first cut point is between the 3rd and 4th numbers, and the second cut point is between the 7th and 8th numbers. We now have

   C_1 = [3 2 1| 4 5 6 7 | 8 9]

2. Reversing the order of the middle chunk will give us

   C_1 = [3 2 1 | 7 6 5 4 | 8 9]

   C_1 = [3 2 1 7 6 5 4 8 9]

The mutation process will generate a new offspring population composed of mutated and nonmutated offspring. Assume that the new offspring population has been generated fully by the mutation process, we then have:

C_1 = [3 2 1 7 6 5 4 8 9]
C_2 = [3 4 7 2 1 5 6 8 9]
C_3 = [4 5 6 1 2 3 9 8 7]
C_4 = [2 1 5 4 3 6 7 8 9]
C_5 = [5 1 6 3 2 4 7 8 9]
C_6 = [9 4 2 3 5 6 1 8 7]
C_7 = [8 2 3 4 7 6 5 1 9]
C_8 = [4 2 3 1 5 9 7 8 6]
C_9 = [9 3 4 2 8 6 5 1 7]
C_{10} = [7 1 5 4 3 6 2 8 9]
C_{11} = [1 2 4 3 5 6 7 8 9]
C_{12} = [2 3 1 5 6 4 7 8 9]
C_{13} = [2 1 3 4 5 6 7 8 9]
C_{14} = [3 2 4 1 5 6 7 8 9]
C_{15} = [5 6 1 3 2 4 7 8 9]
C_{16} = [2 9 4 3 5 6 1 8 7]
C_{17} = [3 2 8 4 7 6 5 1 9]
C_{18} = [7 4 9 5 1 3 2 8 6]
C_{18} = [7 2 3 1 5 9 4 8 6]
C_{19} = [9 3 4 2 1 5 6 7 8]
C_{20} = [7 5 1 4 3 6 2 8 9]

3.9 Create New Population

The final process is the creation of the new population that will be used in the next generation. To get the new population, the $2n$ nonmutated and mutated offspring will be combined with the q members selected from section 3.6. All the individual route lengths will be computed and they will be ordered from shortest to longest route length. The n highest ranking individuals will make up the new population. Assuming we have combined and ordered the $2n$ offspring and q individuals, we have:

C_1 = [3 2 1 7 6 5 4 8 9]

$C_2 = [\ 3\ 4\ 7\ 2\ 1\ 5\ 6\ 8\ 9]$

$C_3 = [\ 4\ 5\ 6\ 1\ 2\ 3\ 9\ 8\ 7]$

$Q_1 = [\ 3\ 2\ 1\ 4\ 5\ 6\ 7\ 8\ 9]$

$Q_2 = [\ 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9]$

$C_4 = [\ 2\ 1\ 5\ 4\ 3\ 6\ 7\ 8\ 9]$

$C_5 = [\ 5\ 1\ 6\ 3\ 2\ 4\ 7\ 8\ 9]$

$Q_3 = [\ 3\ 4\ 7\ 2\ 1\ 5\ 6\ 8\ 9]$

$C_6 = [\ 9\ 4\ 2\ 3\ 5\ 6\ 1\ 8\ 7]$

$C_7 = [\ 8\ 2\ 3\ 4\ 7\ 6\ 5\ 1\ 9]$

$Q_4 = [\ 9\ 3\ 4\ 2\ 5\ 1\ 6\ 8\ 7]$

$C_8 = [\ 4\ 2\ 3\ 1\ 5\ 9\ 7\ 8\ 6]$

$C_9 = [\ 9\ 3\ 4\ 2\ 8\ 6\ 5\ 1\ 7]$

$Q_5 = [\ 4\ 5\ 6\ 1\ 2\ 3\ 9\ 8\ 7]$

$C_{10} = [\ 7\ 1\ 5\ 4\ 3\ 6\ 2\ 8\ 9]$

$C_{11} = [\ 1\ 2\ 4\ 3\ 5\ 6\ 7\ 8\ 9]$

$C_{12} = [\ 2\ 3\ 1\ 5\ 6\ 4\ 7\ 8\ 9]$

$C_{13} = [\ 2\ 1\ 3\ 4\ 5\ 6\ 7\ 8\ 9]$

$C_{14} = [\ 3\ 2\ 4\ 1\ 5\ 6\ 7\ 8\ 9]$

$C_{15} = [\ 5\ 6\ 1\ 3\ 2\ 4\ 7\ 8\ 9]$

$C_{16} = [\ 2\ 9\ 4\ 3\ 5\ 6\ 1\ 8\ 7]$

$C_{17} = [\ 3\ 2\ 8\ 4\ 7\ 6\ 5\ 1\ 9]$

$C_{18} = [\ 7\ 4\ 9\ 5\ 1\ 3\ 2\ 8\ 6]$

$C_{18} = [\ 7\ 2\ 3\ 1\ 5\ 9\ 4\ 8\ 6]$

$C_{19} = [\ 9\ 3\ 4\ 2\ 1\ 5\ 6\ 7\ 8]$

$C_{20} = [\ 7\ 5\ 1\ 4\ 3\ 6\ 2\ 8\ 9]$

Selecting n members to be the new population A, we then have:

$A_1 = C_1 = [3\ 2\ 1\ 7\ 6\ 5\ 4\ 8\ 9]$

$A_2 = C_2 = [\ 3\ 4\ 7\ 2\ 1\ 5\ 6\ 8\ 9]$

$A_3 = C_3 = [\ 4\ 5\ 6\ 1\ 2\ 3\ 9\ 8\ 7]$

$A_4 = Q_1 = [\ 3\ 2\ 1\ 4\ 5\ 6\ 7\ 8\ 9]$

$A_5 = Q_2 = [\ 1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9]$

$A_6 = C_4 = [\ 2\ 1\ 5\ 4\ 3\ 6\ 7\ 8\ 9]$
$A_7 = C_5 = [\ 5\ 1\ 6\ 3\ 2\ 4\ 7\ 8\ 9]$
$A_8 = Q_3 = [\ 3\ 4\ 7\ 2\ 1\ 5\ 6\ 8\ 9]$
$A_9 = C_6 = [\ 9\ 4\ 2\ 3\ 5\ 6\ 1\ 8\ 7]$
$A_10 = C_7 = [\ 8\ 2\ 3\ 4\ 7\ 6\ 5\ 1\ 9]$

3.10 Test Cases and Experimental Setup

There are 3 data sets used in testing the described algorithm each containing different number of data points (cities): Burma14 with 14 cities, EIL51 with 51 cities, and KROB100 with 100 cities, where each city is given by their respective x and y coordinates. Raw data sets are obtained from TSPLIB and presented on Tables A.1, A.2, and A.3 from Appendix B. Figure 3.2 illustrates the maps with their cities.

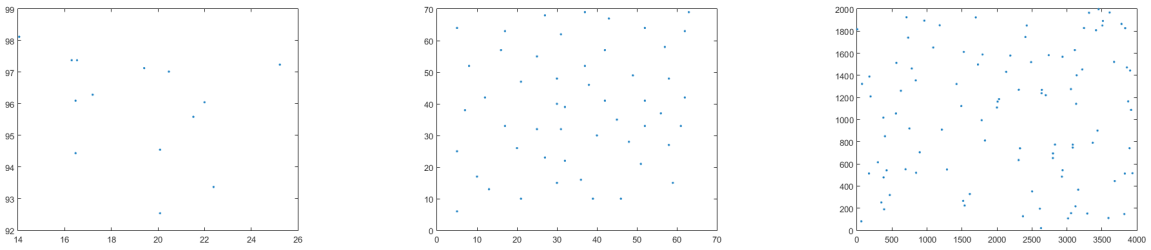


Figure 3.2: Illustration of the maps with the cities as data points. (L-R) Burma 14, EIL51, and KROB100. Larger illustrations of these maps can be seen on Figures B.1, B.2, and B.3 from Appendix B.

Three algorithms are used in this project: Basic Genetic Algorithm (BGA), Multi-Offspring Genetic Algorithm (MO-GA), and the Multi-Offspring Genetic Algorithm with Dynamic Mutation (MO-GA DM). The test cases are generated randomly, for each map, and those randomly generated values will be the initial values for all the three algorithms all throughout the different runs. The number of iterations will be the terminating condition: 100 for Burma14, and 1000 for both EIL51 and KROB100.

Each map is saved on a csv file which is imported on Matlab. The algorithm described on section 2.7 is used to implement MO-GA. For the BGA, the algorithm exhausted is

almost the same as the one used for MO-GA, however, the number of offspring is reduced into two per crossover as it is the main difference between the BGA and MO-GA [40]. An initial population is generated at random once for all the algorithms for each map to give each method an equal starting line. All the algorithms are all coded in Matlab R2021a Update 3 (64-bit). It runs on MacBook Pro (2020) with Apple M1 Chip with 16 GB RAM.

Chapter 4

Results and Discussion

Each algorithm was run 100 times per map. The shortest and longest path lengths out of those 100 runs were obtained and their relative error compared to the known solution was computed. Ideal solutions are expected to have lower path lengths. Additionaly, the relative error of these solutions are minimal. The relative error is computed by using Equation 4.1.

$$\%Error = \frac{S_O - S_K}{S_K} * 100\% \tag{4.1}$$

where:

S_K = the known optimal solution
S_O = the obtained solution from the algorithm

Algorithm	Obtained Solution	Known Optimal Solution	% Error
BGA	30.8785	30.8785	0
MO-GA	30.8785	30.8785	0
MO-GA with Dynamic Mutation	30.8785	30.8785	0

Table 4.1: The comparison of the obtained best optimum solutions for Burma14.

For Burma14, all the three algorithms arrived at the same optimal solution, yielding a path length of 30.8785, which is the same as the known optimal solution. Table 4.1 shows the data of the best solutions for each algorithm alongside the knows solution.

With the worst solutions for Burma14, BGA yielded the longest path length having 33.7082 at 9.16% error, while MO-GA with DM had the shortest length at 0% error from the optimal solution. Meanwhile, MOGA had a length of 31.8283 with 3.08% error. The comparison of the worst solutions can be seen closely on Table 4.2.

Algorithm	Obtained Solution	Known Optimal Solution	% Error
BGA	33.7082	30.8785	9.16
MO-GA	31.8283	30.8785	3.08
MO-GA with Dynamic Mutation	30.8785	30.8785	0

Table 4.2: The comparison of the obtained worst optimum solutions for Burma14 after 100 runs.

Algorithm	Obtained Solution	Known Optimal Solution	% Error
BGA	437.5542	426	2.71
MO-GA	437.4470	426	2.7
MO-GA with Dynamic Mutation	429.1179	426	0.73

Table 4.3: The comparison of the obtained best optimum solutions for EIL51 after 100 runs.

In the case of EIL 51, the best solutions of the three algorithms differ. The known optimal solution has a length of 426. MOGA with DM produced the solution with the lowest relative error at 0.73% having a length of 429.1179. BGA and MOGA came close to each other, yielding 437.5542 at 2.71% and 437.4470 at 2.7% respectively. This comparison is summarized on Table 4.3.

With its worst solutions, the MO-GA with DM still had the best one having 459.7808 at 7.93% relative error. MO-GA had 476.5986 at 11.88% error while BGA had the worst performance among the three with 493.3572 at 15.81% error. The worst solutions are presented on Table 4.4.

For the largest of the three maps, the KROB100, MO-GA with DM outperformed the other algorithms. For the best solutions presented on Table 4.5, MO-GA with DM had a path length of 22,585.9404 with 2.01% error compared to the known optimal solution which is at 22,141. MO-GA performed the second best with 23,283.9925 at 5.16% error.Meanwhile, BGA resulted in 24,3684.5235 with 10.04% error.

With regards to the worst performances on KROB 100, which is presented on Table 4.6, MO-GA with Dynamic Mutation had the best results among the three algorithms with 24852.6995 at 12.25% error. MO-GA yielded 26777.1512 with 20.94% error and

Algorithm	Obtained Solution	Known Optimal Solution	% Error
BGA	493.3572	426	15.81
MO-GA	476.5986	426	11.88
MO-GA with Dynamic Mutation	459.7808	426	7.93

Table 4.4: The comparison of the obtained worst optimum solutions for EIL51 after 100 runs.

Algorithm	Obtained Solution	Known Optimal Solution	% Error
BGA	24364.5235	22,141	10.04
MO-GA	23283.9925	22,141	5.16
MO-GA with Dynamic Mutation	22585.9404	22,141	2.01

Table 4.5: The comparison of the obtained best optimum solutions for KROB100.

BGA had 27640.0017 at 24.27% error.

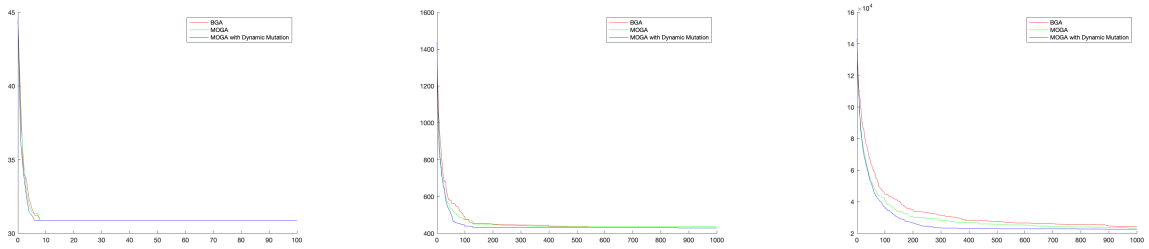


Figure 4.1: Comparison of the best path lengths in every generation for each map for the different algorithms. (L-R) Burma 14, EIL51, and KROB100.

Figure 4.1 shows the comparison of the best path lengths for every generation of the different algorithms. A closer look on the illustrations can be seen on Figures D.1, D.2, and D.3 on the Appendix. For Burma 14 all the algorithms arrived at the same optimum path, however, MOGA-DM arrived at that solution at an earlier generation as compared to the BGA and MOGA. For both the EIL51 and KROB100, MOGA-DM arrived at better solutions than both the BGA and MOGA. Table 4.7 summarizes the average optimum solutions for each algorithm.

The average performances of each of the algorithms on the three maps are presented

Algorithm	Obtained Solution	Known Optimal Solution	% Error
BGA	27640.0017	22,141	24.37
MO-GA	26777.1512	22,141	20.94
MO-GA with Dynamic Mutation	24852.6995	22,141	12.25

Table 4.6: The comparison of the obtained worst optimum solutions for KROB100.

		BGA		MOGA		MOGA DM	
	Known	Average	% Error	Average	% Error	Average	% Error
Burma14	30.8785	31.3042	1.38	31.0775	0.64	30.8785	0
EIL51	426	457.4377	7.38	452.6436	6.25	442.4443	3.86
KROB100	22,141	26199.2686	18.33	25096.3903	13.35	23610.1956	6.64

Table 4.7: The average optimum solutions for each algorithm

on 4.7. MOGA-DM clearly outperformed BGA and MOGA on all the three maps even after averaging the 100 runs done. For Burma 14, it yielded a 0% error while MOGA and BGA had 0.64% and 1.38% errors respectively. MOGA-DM resulted in 3.86% error on EIL51 while MOGA and BGA each had 6.25% and 7.38% errors. For the largest map, the KROB100, MOGA-DM had 6.64% error on average while MOGA and BGA came up with 13.35% and 18.33% errors.

The comparison of the average performance of the algorithms on each of the paths can be seen on the Figure 4.2. Bigger illustrations can be seen on Figures E.1, E.2, and E.3 from the Appendix. For Burma 14, note that the MOGA-DM has the same average optimum solution as the known solution which means that all of the 100 runs resulted in the real optimum solution. On the other maps, MOGA-DM also yielded better average solutions than both the BGA and MOGA.

The path of the best solutions are then obtained and a visualization of these paths are shown on Figure 4.3. Larger illustrations of these paths can be seen on Figures C.1, C.2, and C.3 within the Appendix.

As observed from the results obtained, the solutions for MO-GA with DM on all maps arrive to the optimum solutions faster and converges to a shorter path length as

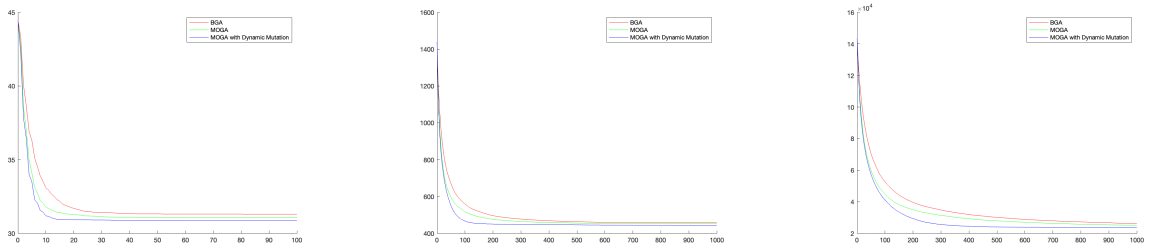


Figure 4.2: Comparison of the average path lengths in each generation for each map for the different algorithms. (L-R) Burma 14, EIL51, and KROB100.

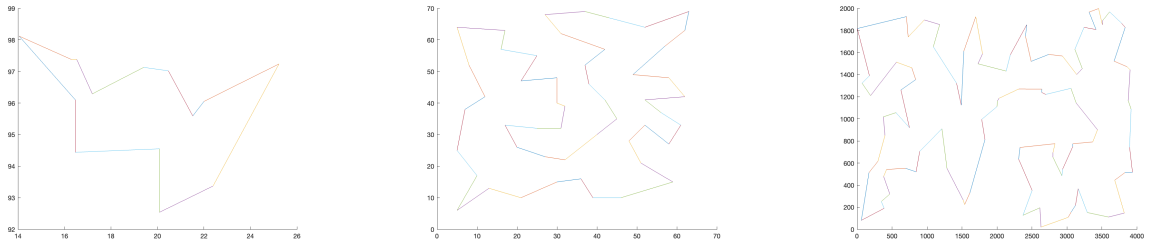


Figure 4.3: Illustration of the best paths for each map from the MOGA-DM. (L-R) Burma 14, EIL51, and KROB100.

compared to the solutions for BGA and MO-GA. This means that MO-GA with DM yielded better results compared to the BGA and MO-GA. It is also noted that the end of each run, MO-GA had better results than the BGA, as expected. MO-GA with DM also outperformed MO-GA on all the three maps where MO-GA with DM had better results at the end of each run, and it also arrived at these results on earlier generation or iteration as compared to the other two algorithms. This shows that MO-GA with DM is superior to BGA and MO-GA when it comes to effectivity and speed.

Chapter 5

Conclusion and Recommendation

The Basic Genetic Algorithm could aid in optimizing solutions for problems like the TSP. Improvements such as increasing the number of offspring, such as in Multi-Offspring Genetic Algorithm, could provide better solutions compared to the BGA which only bears single child by creating a more diverse search space. However, if MO-GA is combined with another strategy like modifying the mutation rate and turning it into a dynamic one, solutions could then be more optimized by creating a balance between optimizing the population and diversifying it with the mutation operator.

The dynamic capabilities of the mutation rate in MO-GA with DM created a more optimized mutation capabilities without trial and error of constant mutation values, and it yielded better solutions by preventing currently fit individuals into occupying the whole population, and thus, encouraging diversification which aided into evading from getting stuck into a local optima. The solutions obtained from the MO-GA DM yielded low percentage errors from the known optimum solutions. With the significant differences in percentage errors of the MO-GA and MO-DA with DM from the known solutions, the modification made with the new algorithm is indeed an improvement.

Other ways of initializing parameters could be studied to create comparisons as to which greatly affects the changes on the behaviour of the population. Another interesting way to explore is tweaking the number of offspring, whether to increase it or to create an adaptive method where parents have different number of offspring. This MO-GA with Dynamic Mutation could be studied further in order to explore more possible ways of obtaining optimized solutions faster and better. Other applications for this algorithm aside from TSP could also be an interesting way to explore its solution searching capabilities.

List of References

[1] V. Adamchick, *Graph theory*, (2005).

[2] A. M. Alzohairy, *Darwin's theory of evolution*, (2014).

[3] R. K. Bhattacharjya, *Introduction to genetic algorithms*, IIT Guwahati, 12 (2012).

[4] U. Bodenhofer, *Genetic algorithms: theory and applications*, 2003.

[5] R. Brady, *Optimization strategies gleaned from biological evolution*, Nature, 317 (1985), pp. 804–806.

[6] C. Brucato, *The Traveling Salesman Problem*, PhD thesis, University of Pittsburgh, 2013.

[7] J. Buurman, S. Zhang, and V. Babovic, *Reducing risk through real options in systems design: the case of architecting a maritime domain protection system*, Risk Analysis: An International Journal, 29 (2009), pp. 366–379.

[8] K. A. De Jong, *An analysis of the behavior of a class of genetic adaptive systems.*, University of Michigan, 1975.

[9] A. C. dos Santos-Paulino, J.-C. Nebel, and F. Flórez-Revuelta, *Evolutionary algorithm for dense pixel matching in presence of distortions*, in European Conference on the Applications of Evolutionary Computation, Springer, 2014, pp. 439–450.

[10] Á. E. Eiben, R. Hinterding, and Z. Michalewicz, *Parameter control in evolutionary algorithms*, IEEE Transactions on evolutionary computation, 3 (1999), pp. 124–141.

[11] J. M. Fitzgerald, C. Ryan, D. Medernach, and K. Krawiec, *An integrated approach to stage 1 breast cancer detection*, in Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation, 2015, pp. 1199–1206.

[12] M. Gen and L. Lin, *Genetic algorithms*, Wiley Encyclopedia of Computer Science and Engineering, (2007), pp. 1–15.

[13] P. Godfrey-Smith, *Conditions for evolution by natural selection*, The Journal of Philosophy, 104 (2007), pp. 489–516.

[14] D. E. Goldberg, R. Lingle, et al., *Alleles, loci, and the traveling salesman problem*, in Proceedings of an international conference on genetic algorithms and their applications, vol. 154, Carnegie-Mellon University Pittsburgh, PA, 1985, pp. 154–159.

[15] J. Grefenstette, R. Gopal, B. Rosmaita, and D. Van Gucht, *Genetic algorithms for the traveling salesman problem*, in Proceedings of the first International Conference on Genetic Algorithms and their Applications, vol. 160, Lawrence Erlbaum, 1985, pp. 160–168.

[16] J. J. Grefenstette, *Optimization of control parameters for genetic algorithms*, IEEE Transactions on systems, man, and cybernetics, 16 (1986), pp. 122–128.

[17] A. Gupta and S. Khurana, *Study of traveling salesman problem using genetic algorithm*, International Journal of Managment, IT and Engineering, 2 (2012), pp. 575–588.

[18] M. Hahsler and K. Hornik, *Tsp-infrastructure for the traveling salesperson problem*, Journal of Statistical Software, 23 (2007), pp. 1–21.

[19] T. Haist and W. Osten, *An optical solution for the traveling salesman problem*, Optics Express, 15 (2007), pp. 10473–10482.

[20] A. Homaifar, S. Guan, and G. E. Liepins, *Schema analysis of the traveling salesman problem using genetic algorithms*, Complex Systems, 6 (1992), pp. 533–552.

[21] V. Jain and J. S. Prasad, *Solving travelling salesman problem using greedy genetic algorithm gga*, International Journal of Engineering and Technology, 9 (2017), pp. 1148–54.

[22] P. Larranaga, C. M. H. Kuijpers, R. H. Murga, I. Inza, and S. Dizdarevic, *Genetic algorithms for the travelling salesman problem: A review of representations and operators*, Artificial Intelligence Review, 13 (1999), pp. 129–170.

[23] M. M. Maad, *Genetic algorithm optimization by natural selection*, 2016.

[24] O. Maimon and D. Braha, *A genetic algorithm approach to scheduling pcbs on a single machine*, International Journal of Production Research, 36 (1998), pp. 761–784.

[25] R. Matai, S. P. Singh, and M. L. Mittal, *Traveling salesman problem: an overview of applications, formulations, and solution approaches*, Traveling salesman problem, theory and applications, 1 (2010).

[26] J. Morris, *Combinatorics: Enumeration, graph theory, and design theory*, (2017).

[27] T. Narwadi and Subiyanto, *An application of traveling salesman problem using the improved genetic algorithm on android google maps*, in AIP Conference Proceedings, vol. 1818, AIP Publishing LLC, 2017, p. 020035.

[28] C. H. Papadimitriou and P. CH, *The euclidean traveling salesman problem is np-complete.*, (1977).

[29] C. Reeves, *Genetic algorithms*, in Handbook of metaheuristics, Springer, 2003, pp. 55–82.

[30] K. Ruohonen, *Graph theory. tampereen teknillinen yliopisto. originally titled graafiteoria, lecture notes translated by tamminen, j., lee, k*, C. and Piché, R, (2013).

[31] M. Safe, J. Carballido, I. Ponzoni, and N. Brignole, *On stopping criteria for genetic algorithms*, in Brazilian Symposium on Artificial Intelligence, Springer, 2004, pp. 405–413.

[32] T. Salvador, *The traveling salesman problem: a statistical approach*, Report within the scope of the program "Novos Talentos em Matemática"—Fundação Calouste Gulbenkian, Portugal, (2010).

[33] S. Singh and E. A. Lodhi, *Study of variation in tsp using genetic algorithm and its operator comparison*, International Journal of Soft Computing and Engineering (IJSCE), 3 (2013), pp. 264–267.

[34] M. Srinivas and L. M. Patnaik, *Adaptive probabilities of crossover and mutation in genetic algorithms*, IEEE Transactions on Systems, Man, and Cybernetics, 24 (1994), pp. 656–667.

[35] ——, *Genetic algorithms: A survey*, computer, 27 (1994), pp. 17–26.

[36] K. Stanislawska, K. Krawiec, and T. Vihma, *Genetic programming for estimation of heat flux between the atmosphere and sea ice in polar regions*, in Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation, 2015, pp. 1279–1286.

[37] R. P. Stanley, *What is enumerative combinatorics?*, in Enumerative combinatorics, Springer, 1986, pp. 1–63.

[38] S. Walzer, *Combinatorics*, (2016).

[39] F. Wang, J. Wang, C. Wu, and Q. Wu, *Research on improvement of real number genetic algorithm*, (2006).

[40] J. Wang, O. K. Ersoy, M. He, and F. Wang, *Multi-offspring genetic algorithm and its application to the traveling salesman problem*, Applied Soft Computing, 43 (2016), pp. 415–423.

[41] S. Waseda University, *Darwin's theory of evolution by natural selection a new theory of the origins of life*, Introduction to History and Philosophy of Science.

[42] J. Xu, L. Pei, and R.-z. Zhu, *Application of a genetic algorithm with random crossover and dynamic mutation on the travelling salesman problem*, Procedia computer science, 131 (2018), pp. 937–945.

[43] S. L. Yadav and A. Sohal, *Comparative study of different selection techniques in genetic algorithm*, International Journal of Engineering, Science and Mathematics, 6 (2017), pp. 174–180.

[44] S. Yang, *Genetic algorithms with elitism-based immigrants for changing optimization problems*, in Workshops on Applications of Evolutionary Computation, Springer, 2007, pp. 627–636.

Appendix A

Table for raw data of the maps.

Table A.1: The raw city data for EIL51.

City Number	X- coordinate	Y-coordinate
1	37	52
2	49	49
3	52	64
4	20	26
5	40	30
6	21	47
7	17	63
8	31	62
9	52	33
10	51	21
11	42	41
12	31	32
13	5	25
14	12	42
15	36	16
16	52	41
17	27	23
18	17	33
19	13	13
20	57	58
21	62	42
22	42	57

23	16	57
24	8	52
25	7	38
26	27	68
27	30	48
28	43	67
29	58	48
30	58	27
31	37	69
32	38	46
33	46	10
34	61	33
35	62	63
36	63	69
37	32	22
38	45	35
39	59	15
40	5	6
41	10	17
42	21	10
43	5	64
44	30	15
45	39	10
46	32	39
47	25	32
48	25	55
49	48	28
50	56	37
51	30	40

Table A.2: The raw city data for KROB100.

City Number	X- coordinate	Y-coordinate
1	3140	1401
2	556	1056
3	3675	1522
4	1182	1853
5	3595	111
6	962	1895
7	2030	1186
8	3507	1851
9	2642	1269
10	3438	901
11	3858	1472
12	2937	1568
13	376	1018
14	839	1355
15	706	1925
16	749	920
17	298	615
18	694	552
19	387	190
20	2801	695
21	3133	1143
22	1517	266
23	1538	224
24	844	520
25	2639	1239
26	3123	217
27	2489	1520
28	3834	1827

29	3417	1808
30	2938	543
31	71	1323
32	3245	1828
33	731	1741
34	2312	1270
35	2426	1851
36	380	478
37	2310	635
38	2830	775
39	3829	513
40	3684	445
41	171	514
42	627	1261
43	1490	1123
44	61	81
45	422	542
46	2698	1221
47	2372	127
48	177	1390
49	3084	748
50	1213	910
51	3	1817
52	1782	995
53	3896	742
54	1829	812
55	1286	550
56	3017	108
57	2132	1432
58	2000	1110
59	3317	1966

60	1729	1498
61	2408	1747
62	3292	152
63	193	1210
64	782	1462
65	2503	352
66	1697	1924
67	3821	147
68	3370	791
69	3162	367
70	3938	516
71	2741	1583
72	2330	741
73	3918	1088
74	1794	1589
75	2929	485
76	3453	1998
77	896	705
78	399	850
79	2614	195
80	2800	653
81	2630	20
82	563	1513
83	1090	1652
84	2009	1163
85	3876	1165
86	3084	774
87	1526	1612
88	1612	328
89	1423	1322
90	3058	1276

91	3782	1865
92	347	252
93	3904	1444
94	2191	1579
95	3220	1454
96	468	319
97	3611	1968
98	3114	1629
99	3515	1892
100	3060	155

Table A.3: The raw city data for Burma14.

City Number	X- coordinate	Y-coordinate
1	16.47	96.1
2	16.47	94.44
3	20.09	92.54
4	22.39	93.37
5	25.23	97.24
6	22	96.05
7	20.47	97.02
8	17.2	96.29
9	16.3	97.38
10	14.05	98.12
11	16.53	97.38
12	21.52	95.59
13	19.41	97.13
14	20.09	94.55

Appendix B

Illustrations for the raw map

Figure B.1: Burma14 Map

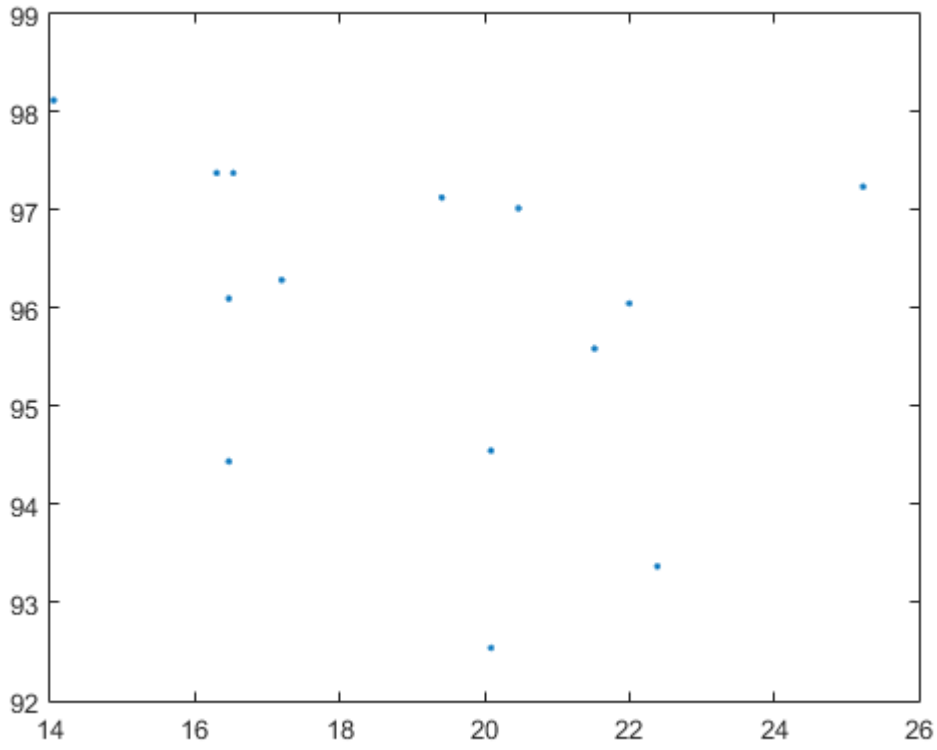


Figure B.2: EIL51 Map

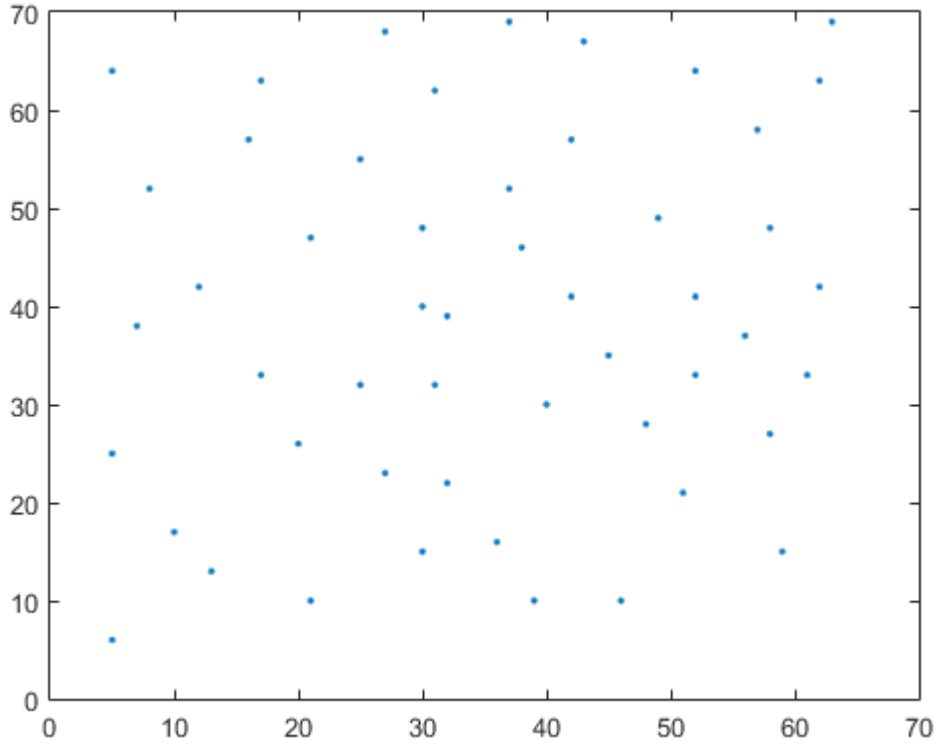
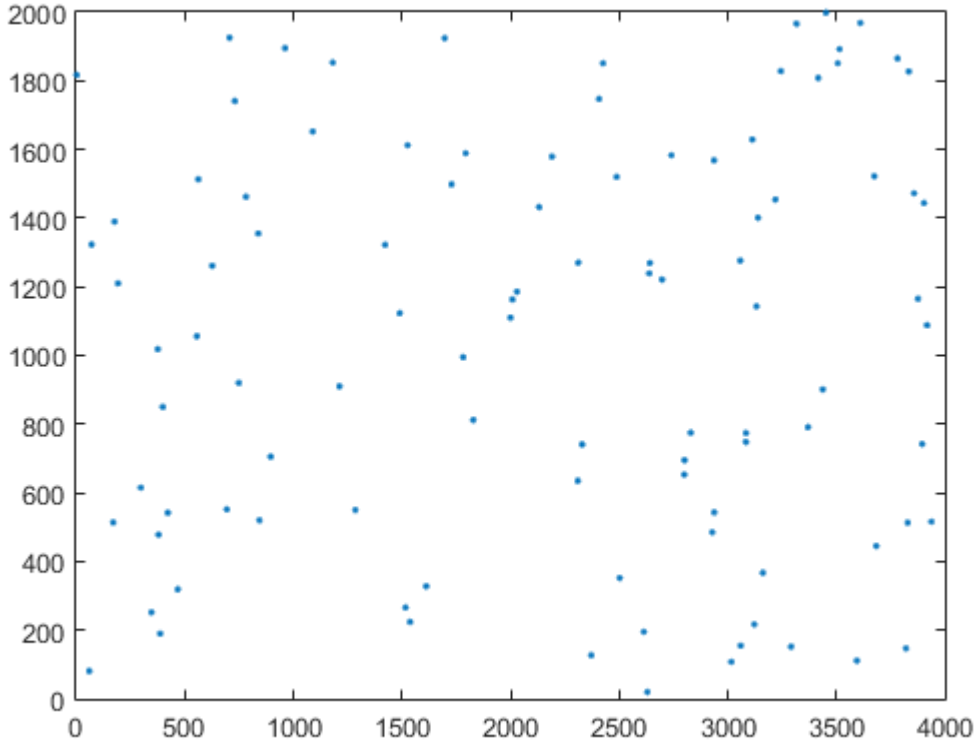


Figure B.3: KROB100 Map



Appendix C

Illustration of the best path lengths

Figure C.1: Illustration for the obtained optimal solution for Burma14.

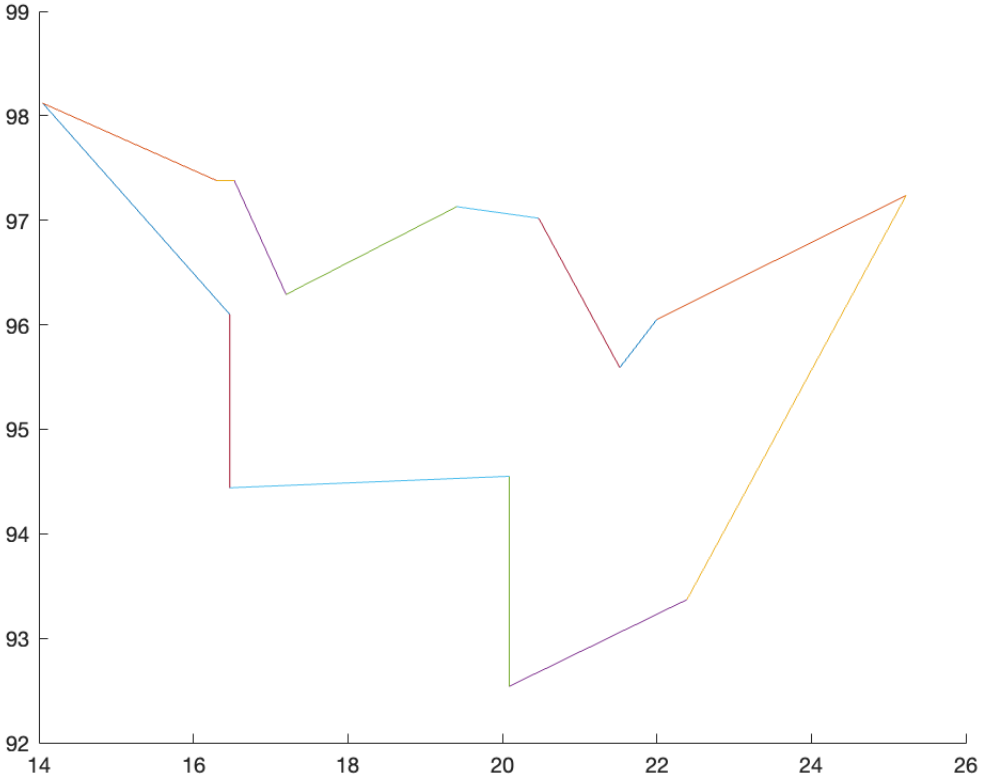


Figure C.2: Illustration for the obtained optimal solution for EIL51.

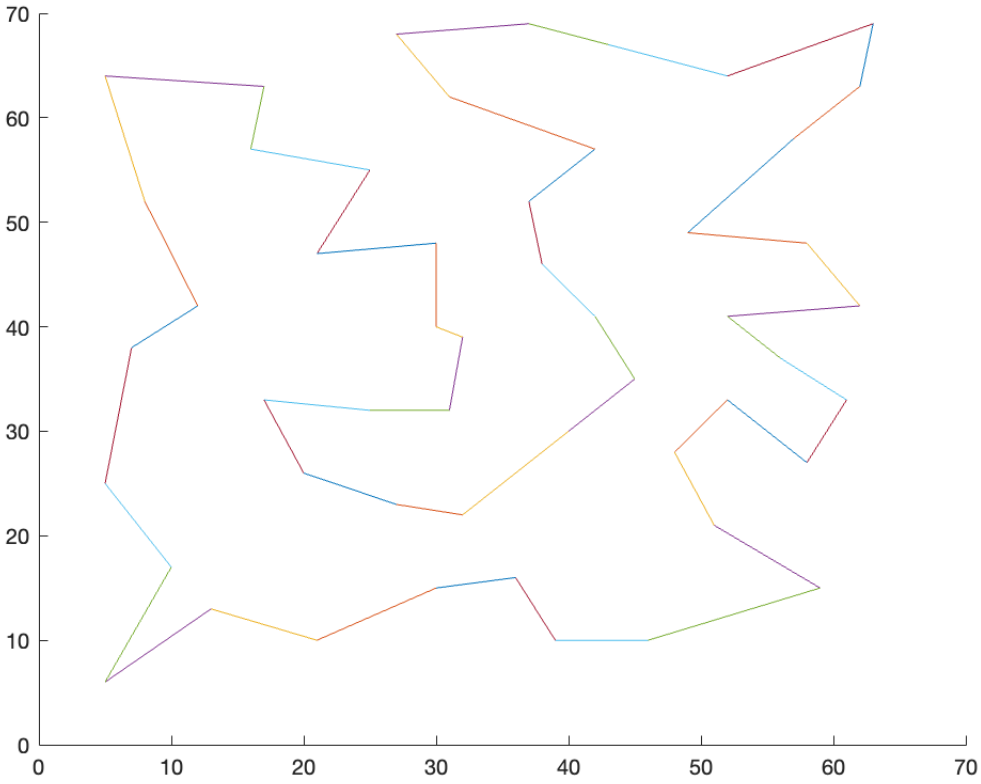
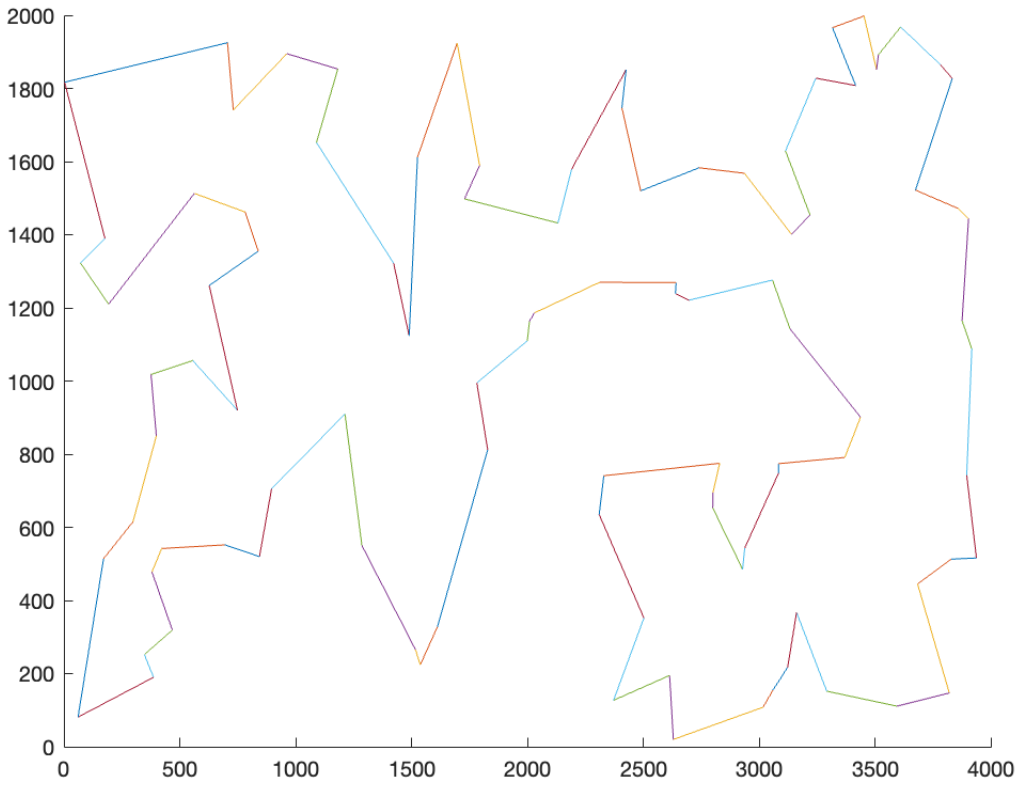


Figure C.3: Illustration for the obtained optimal solution for KROB100.



Appendix D

Graph of best path lengths per generation

Figure D.1: Graph of the path length of the algorithms for Burma14 in every generation for the best solution.

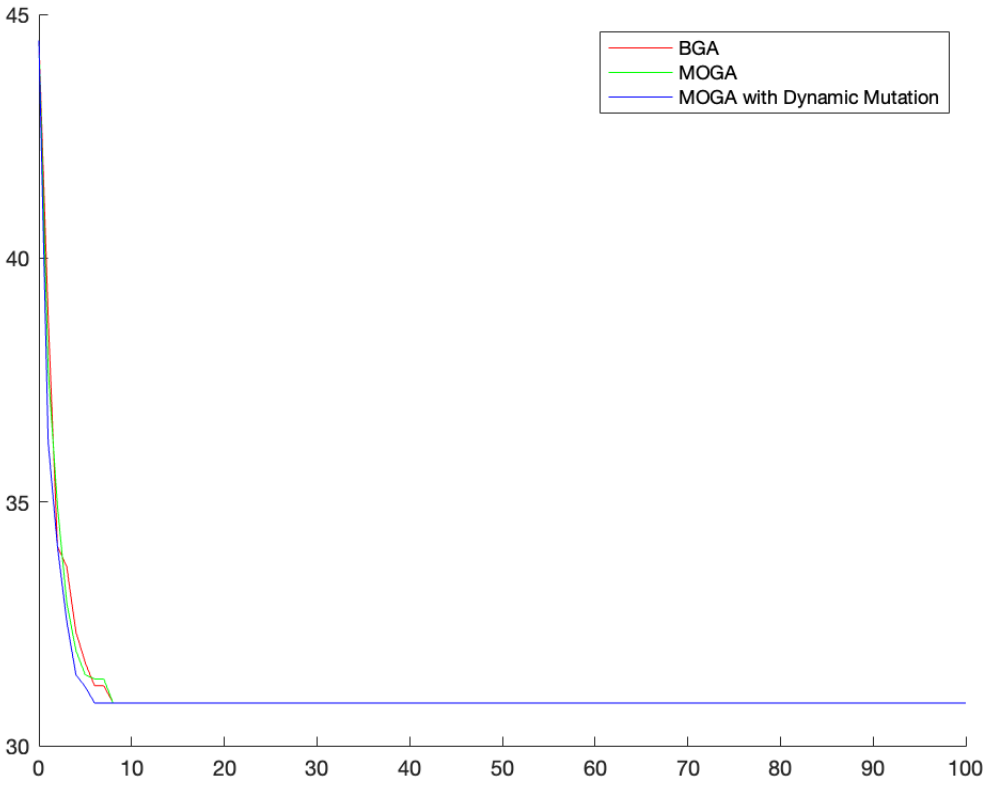


Figure D.2: Graph of the path length of the algorithms for EIL51 in every generation for the best solution.

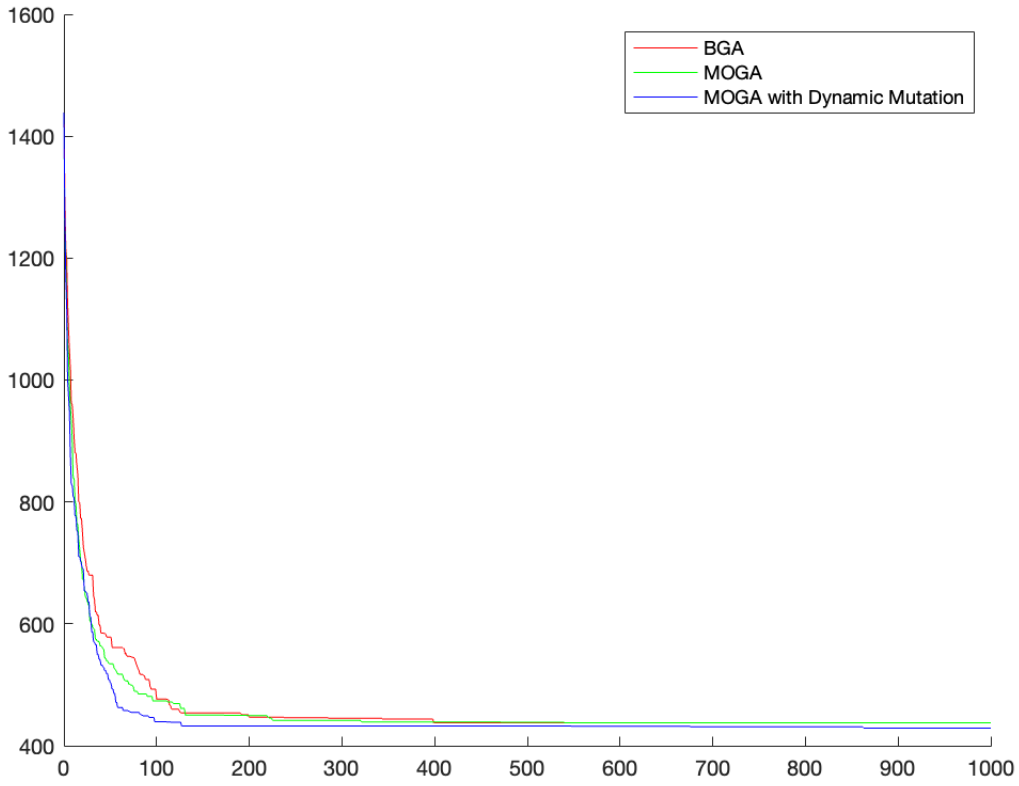
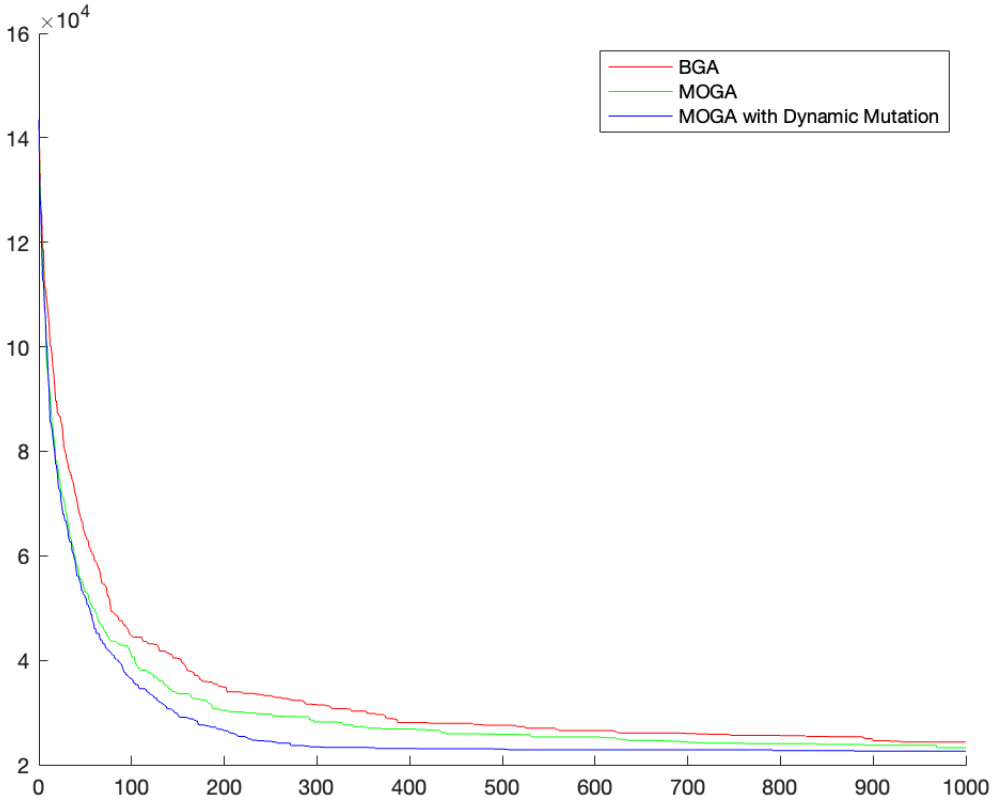


Figure D.3: Graph of the path length of the algorithms for KROB100 in every generation for the best solution.



Appendix E

Graph of average lengths per generation

Figure E.1: Graph of the average path length of the algorithms for Burma14 in every generation.

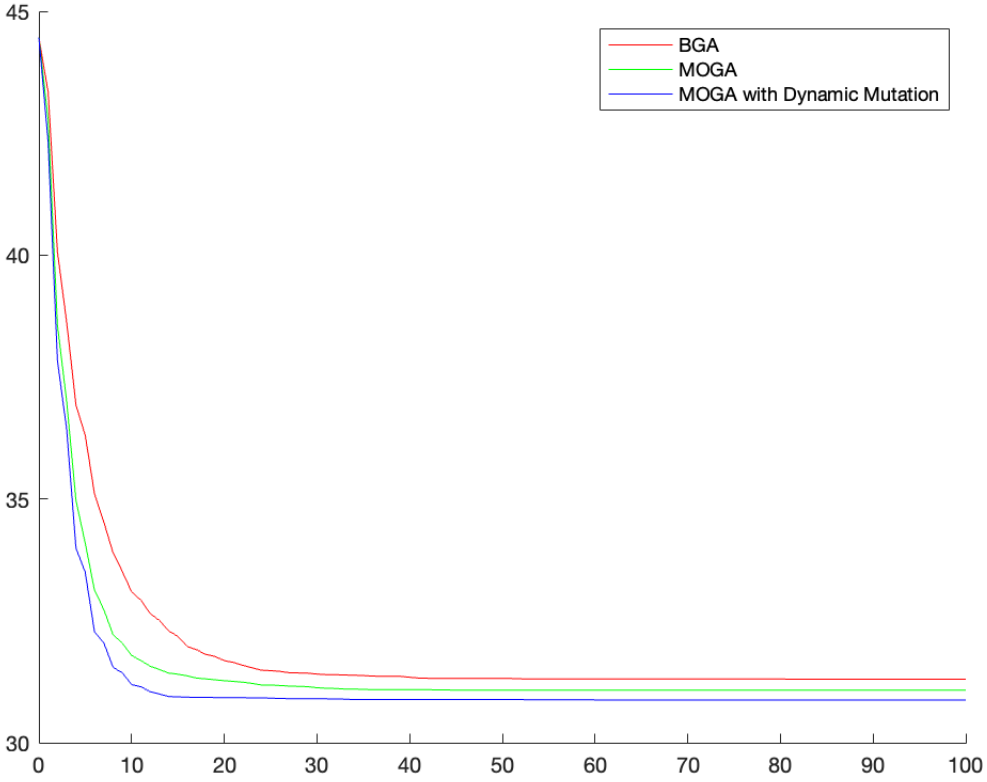


Figure E.2: Graph of the average path length of the algorithms for EIL51 in every generation.

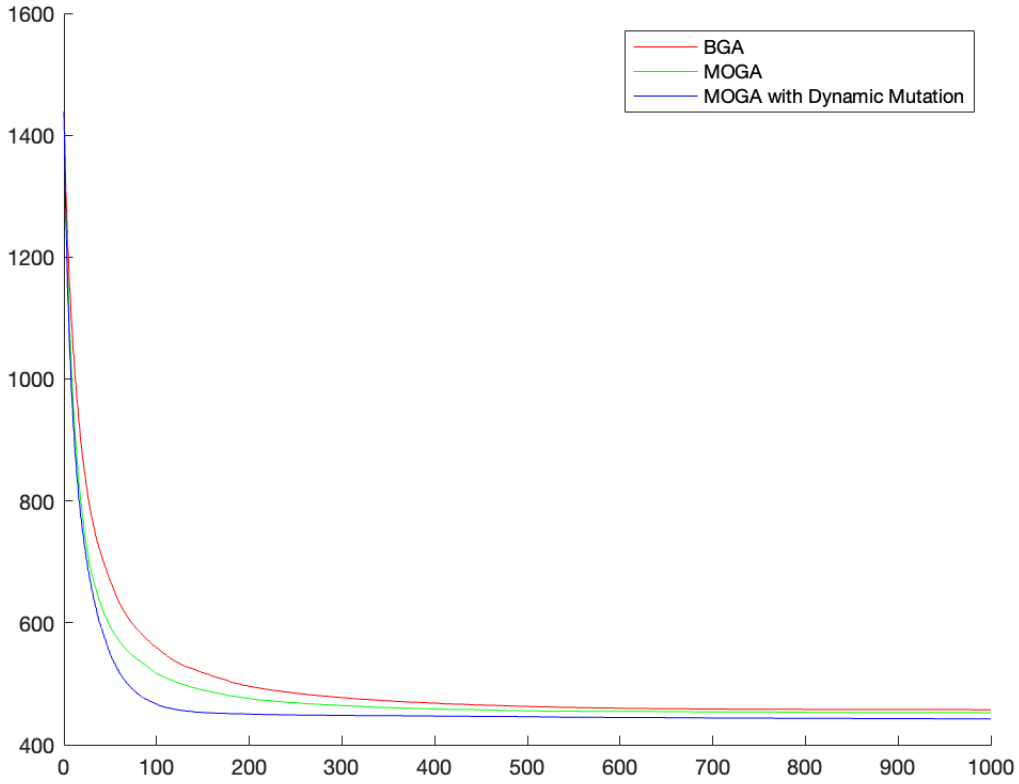
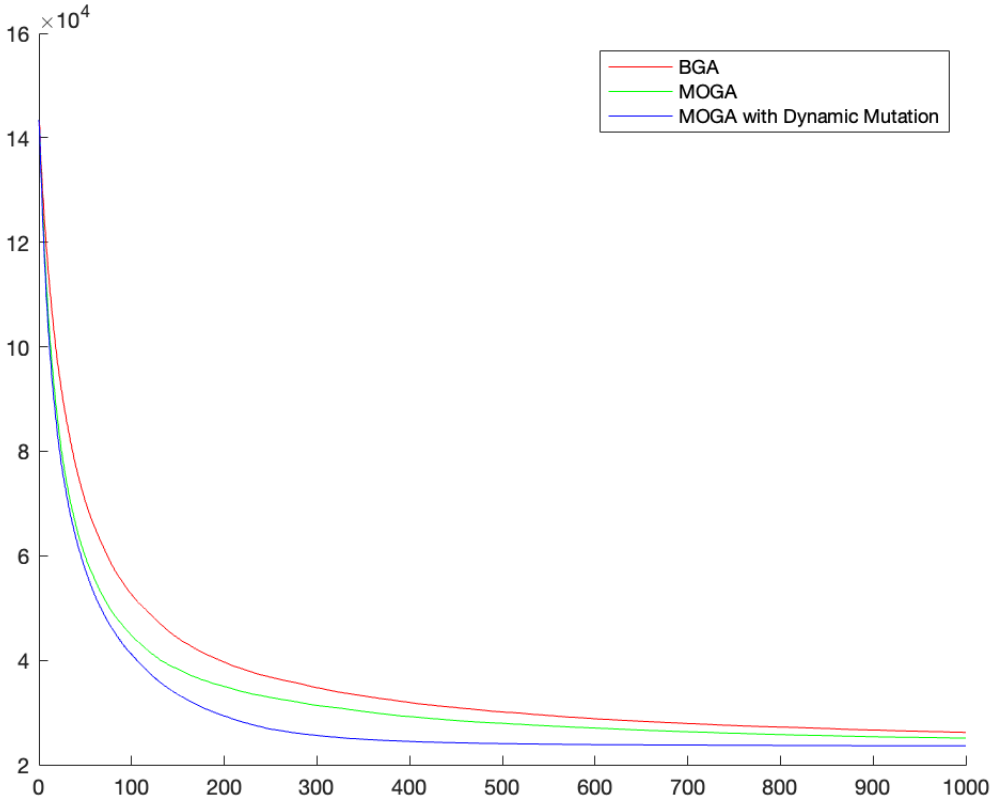


Figure E.3: Graph of the average path length of the algorithms for KROB100 in every generation.



Appendix F

Source Code

```
%% Import data from text file.
% Script for importing data from the following text file:
%
%    /Users/FieryCloud/Documents/190/Data/EIL51.csv
%
% To extend the code to different selected data or a different text file,
% generate a function instead of a script.
% Auto-generated by MATLAB on 2019/08/27 21:27:15
%% Initialize variables.
filename = '/Users/FieryCloud/Documents/190/Data/EIL51.csv';
delimiter = ' ';

%% Format string for each line of text:
%   column1: double (%f)
%   column2: double (%f)
%   column3: double (%f)
% For more information, see the TEXTSCAN documentation.
formatSpec = '%f%f%f%[^\n\r]';

%% Open the text file.
fileID = fopen(filename,'r');

%% Read columns of data according to format string.
% This call is based on the structure of the file used to generate this
% code. If an error occurs for a different file, try regenerating the code
% from the Import Tool.
dataArray = textscan(fileID, formatSpec, 'Delimiter', delimiter, 'MultipleDelimsAsOne', true,  'ReturnOnError', false);

%% Close the text file.
fclose(fileID);

%% Post processing for unimportable data.
% No unimportable data rules were applied during the import, so no post
% processing code is included. To generate code which works for
% unimportable data, select unimportable cells in a file and regenerate the
% script.

%% Allocate imported array to column variable names
```

```
a = dataArray{:, 1};
b = dataArray{:, 2};
c = dataArray{:, 3};


%% Clear temporary variables
clearvars filename delimiter formatSpec fileID dataArray ans;



%% count the number of cities from data and save them in a variable
si=size(c);
si=si(1,1);


%% initialize variables and parameters to be used
param=0.2; %the beta parameter for the selection process
q=5;       %the number of elitist members to be chosen
optimumArray=0; %temporary variable for the array of optimum values per generation
pop_size = 100; %initialize population size
maxGen = 1000; %the maximum number of generations to be done

pop_temp= zeros (pop_size,si);  %temporary initialization for the population

%initializing the population by randomizing based on the cities
for y = 1:1:pop_size
pop_temp(y,:)= randperm(si,si);
end

%run for 100 times

%save the population on a new variable
pop = pop_temp;

%% solving for the fitness of each individual
fit= zeros (1, pop_size);
for y= 1:1:pop_size
for x=1:1:si
if x==si
fit(1,y)= fit(1,y) + ((b(pop(y,x)) - b(pop(y,1)))^(2) + (c(pop(y,x)) - c (pop(y,1)))^2)^(0.5);
else
fit(1,y)= fit(1,y) + ((b(pop(y,x)) - b(pop(y,(x+1))))^(2) + (c(pop(y,x)) - c (pop(y,(x+1))))^2)^(0.5);
end

end

end
```

```
%% sorting from best to least fit
sort_pop= cat(2,pop,(fit.'));
sort_pop= sortrows(sort_pop,(si+1));
optimum= sort_pop(1,si+1);
pop= sort_pop(:,1:si);
optimumArray(1) = optimum; %save the best gene in the array

%% Compute for selection probabilities
fitval= zeros (pop_size,2);
fitprob= zeros (pop_size,2);

for ctr=1:1:pop_size
fitval(ctr,1) = param * ((1-param)^(ctr-1));
fitval(ctr,2) = (pop_size - ctr + 1)/ pop_size;
end

sfit=sum(fitval);

temp_sum1=0;
temp_sum2=0;


for ctr=1:1:pop_size
temp_sum1 = temp_sum1 + (fitval(ctr,1));
fitprob(ctr,1)= temp_sum1/sfit(1);

temp_sum2 = temp_sum2 + (fitval(ctr,2));
fitprob(ctr,2)= temp_sum2/sfit(2);
end


% We will start with 2, as the first generation to be considered is the
% one from the initial population
gen=2;

%% Run the whole GA until terminating condition is satisfied.
% Terminating condition is the number of generations.
while gen <= maxGen

offspring= zeros((pop_size*2),si); %initialize temporary offpspring

%% selection process
rep=1;
while rep < (pop_size*2)
if mod(gen,2)==1
ctr=1;
```

```
par1=rand;

while par1 > fitprob(ctr,1)
ctr=ctr+1;
end
par1=pop(ctr,:);

ctr=1;
par2=rand;

while par2 > fitprob(ctr,1)
ctr=ctr+1;
end
par2=pop(ctr,:);

else
ctr=1;
par1=rand;

while par1 > fitprob(ctr,2)
ctr=ctr+1;
end
par1= pop(ctr,:);

ctr=1;
par2=rand;

while par2 > fitprob(ctr,2)
ctr=ctr+1;
end
par2=pop(ctr,:);
end

%% crossover
e_num=size(par1);
e_count= e_num(1,2);
c1=zeros(e_num);
c2=zeros(e_num);
c3=zeros(e_num);
c4=zeros(e_num);

position=randperm((e_count-1),2);

if position(1)>position(2)
p1=position(2);
mut_rate=position(1);
else
```

```
mut_rate=position(2);
p1=position(1);
end

b_sec1= par2(1:p1);
b_sec2= par2((p1+1):mut_rate);
b_sec3= par2(mut_rate+1:e_count);

mid_count= size(b_sec2);
m_c= mid_count(1,2);
r_c= e_count-m_c;

a_sec1= par1(1:p1);
a_sec2= par1((p1+1):mut_rate);
a_sec3= par1(mut_rate+1:e_count);

b_temp1= [b_sec3,b_sec1,b_sec2];
a_temp1= [a_sec3,a_sec1,a_sec2];

point= mut_rate+1;


checker= ismember(b_temp1,a_sec2);
for ctr=1:1:e_count

if checker(ctr)==0
if point>e_count
point=1;
c1(point)=b_temp1(ctr);
point=point+1;
else
c1(point)=b_temp1(ctr);
point=point+1;
end
else
end
end


for ctr=(p1+1):1:mut_rate
c1(ctr)= par1(ctr);
end
offspring(rep,:)=c1;
rep=rep+1;

point=mut_rate+1;
```

```
checker= ismember(a_temp1,b_sec2);
for ctr=1:1:e_count

if checker(ctr)==0
if point>e_count
point=1;
c2(point)=a_temp1(ctr);
point=point+1;
else
c2(point)=a_temp1(ctr);
point=point+1;
end
else
end
end


for ctr=(p1+1):1:mut_rate
c2(ctr)= par2(ctr);
end
offspring(rep,:)=c2;
rep=rep+1;

b_temp1= [b_sec2,b_sec1,b_sec3];
a_temp1= [a_sec2,a_sec1,a_sec3];


checker= ismember(b_temp1,a_sec3);

point=1;
for ctr=1:1:e_count
if checker(ctr)==0
c3(point)= b_temp1(ctr);
point=point+1;
else
end
end

for point=(mut_rate+1):1:e_count
c3(point)=par1(point);
end

offspring(rep,:)=c3;
rep=rep+1;

checker= ismember(a_temp1,b_sec3);
```

```
point=1;
for ctr=1:1:e_count
if checker(ctr)==0
c4(point)= a_temp1(ctr);
point=point+1;
else
end
end

for point=(mut_rate+1):1:e_count
c4(point)=par2(point);
end
offspring(rep,:)=c4;
rep=rep+1;
end

%% compute for the path length of each individual
fit= zeros (1, (pop_size*2));
for y= 1:1:(pop_size*2)
for x=1:1:si
if x==si
fit(1,y)= fit(1,y) + ((b(offspring(y,x)) - b(offspring(y,1)))^(2) + (c(offspring(y,x)) - c (offspring(y,1)))^2)^(0.5);
else
fit(1,y)= fit(1,y) + ((b(offspring(y,x)) - b(offspring(y,(x+1))))^(2) + (c(offspring(y,x)) - c (offspring(y,(x+1))))^2)^(0.5
end
end
end

%% merge the q individuals and the 2n offspring and sort
sort_off= cat(2,offspring,(fit.'));

merge=cat(1,sort_pop(1:q,:),sort_off(:,:));

sort_gen= sortrows(merge,(si+1));

sort_fitness = sort_gen(:, (si+1));

numberOf = numel(sort_fitness);

%% compute for the mutation rate
fitnessAverage = sum(sort_fitness)/205;
mut_rate = (1 - ((fitnessAverage - sort_fitness(1))/sort_fitness(1)));

%% Mutation process
for ctr=1:1:(pop_size*2)
m=rand;
```

```
if m<mut_rate
mut= offspring(ctr,:);
e_num= size(mut);
e_count=e_num(1,2);

position=randperm((e_count-1),2);

if position(1)>position(2)
p1=position(2);
mut_rate=position(1);
else
mut_rate=position(2);
p1=position(1);
end

sec1= mut(1:p1);
sec2= mut((p1+1):mut_rate);
sec3= mut(mut_rate+1:e_count);

sec_flip= flip(sec2);

offspring(ctr,:)= [sec1,sec_flip,sec3];

else


end
end

%% Recompute fitness values
fit= zeros (1, (pop_size*2));
for y= 1:1:(pop_size*2)
for x=1:1:si
if x==si
fit(1,y)= fit(1,y) + ((b(offspring(y,x)) - b(offspring(y,1)))^(2) + (c(offspring(y,x)) - c (offspring(y,1)))^2)^(0.5);
else
fit(1,y)= fit(1,y) + ((b(offspring(y,x)) - b(offspring(y,(x+1))))^(2) + (c(offspring(y,x)) - c (offspring(y,(x+1))))^2)^(0.5
end

end

end

%% Create new population. Merge the 2n mutated and unmutated offspring, and the q
% individuals. Sort them and select the n highest individuals.
sort_off2= cat(2,offspring,(fit.'));
```

```
merge= cat(1,sort_gen(1:q,:),sort_off2(:,:));

sort_gen= sortrows(merge,(si+1));

pop = sort_gen(1:pop_size,1:si);

sort_pop = sort_gen(1:100,:);

optimum = sort_gen(1,(si+1)); %the optimum path length for current generation

pop_ave = sum(sort_pop(:, (si+1)))/100;

pop_ave_array_MOGA2(gen)= pop_ave;

optimumArray(gen) = optimum; %the array holding optimum path length per generation

gen=gen+1; %increment for the number of generations

graphSol = pop(1,:); %the array holding the best individual (sequence of cities)
end

%% graphing the path taken of best individual
hold on
for x = 1:1:(si-1)

plot([ b(graphSol(x)) b(graphSol(x+1))], [c(graphSol(x)) c(graphSol(x+1))]);

end
plot([ b(graphSol(si)) b(graphSol(1))], [c(graphSol(si)) c(graphSol(1))]);
```