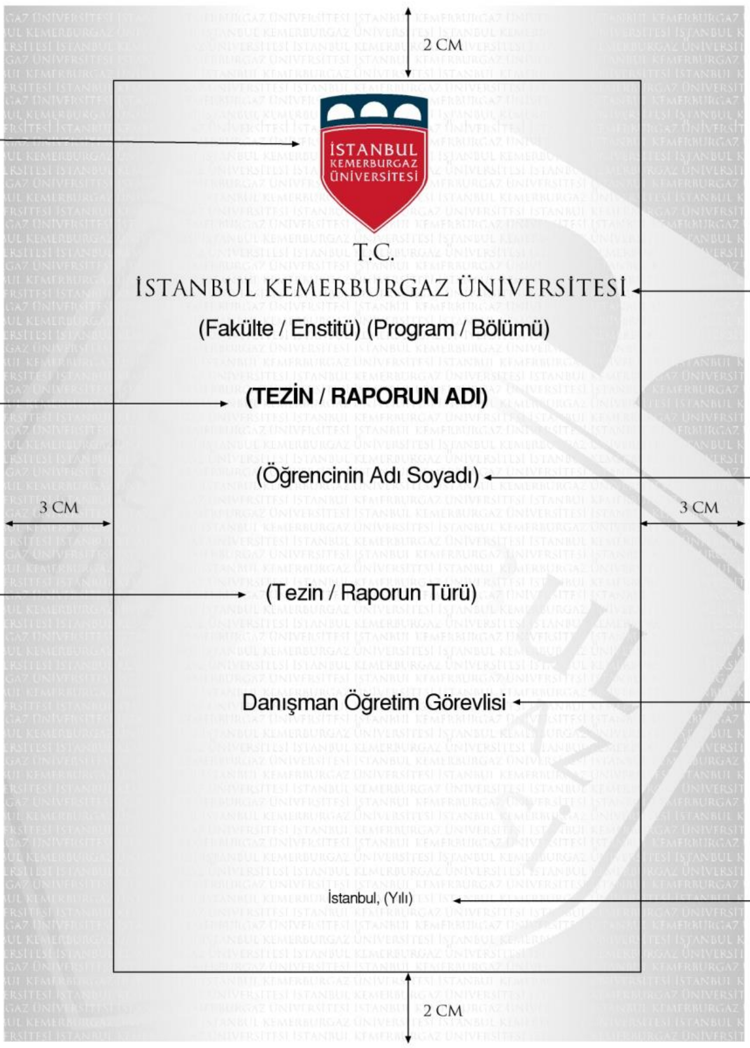


2 CM

T.C.

İSTANBUL KEMERBURGAZ ÜNİVERSİTESİ

(Fakülte / Enstitü) (Program / Bölümü)

(TEZİN / RAPORUN ADI)

(Öğrencinin Adı Soyadı)

3 CM

3 CM

(Tezin / Raporun Türü)

Danışman Öğretim Görevlisi

İstanbul, (Yılı)

2 CM

Logo
Logo 4 cm
ve ortalı olarak
kullanılmalıdır.

Tezin Adı
Büyük harflerle
bold ve 18 punto
olarak ortalanarak
yazılmalıdır.

Tezin Türü
Tezin Türü yazısı
baş harfi büyük
ve 18 punto olarak
yazılmalıdır.

Fakülte ve Bölüm Başlığı
Baş harfleri büyük
olarak 18 punto
yazılmalıdır.

Hazırlayanın Adı
Tezi hazırlayanın
adı soyadı ve
ünvanı (Dr.) baş
harfleri büyük
18 punto ve
ortalanarak
yazılmalıdır.

Danışman Yazısı
Danışman Yazısı
baş harfi büyük
ve 18 punto olarak
yazılmalıdır.

Sunulan Yıl
Tezin sunulduğu yıl
sunulan yerden ve
virgülden sonra
rakamla
yazılmalıdır.

COMPARISON OF OPTIMIZATION ALGORITHMS FOR THE SOLUTION OF TRAVELING SALESMAN PROBLEM

by

Waled Milad Abulsasem Alashheb

Department

Of

Information Technology

Submitted to the Graduate School of Science and Engineering

in partial fulfillment of the requirements for the degree of

Master of Science

ALTINBAŞ UNIVERSITY

2018

This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.

Asst. Prof. Adil Deniz Duru	Assoc. Prof. Dr. Oğuz Bayat
Co-Supervisor	Supervisor

Examining Committee Members (first name belongs to the chairperson of the jury and the second name belongs to supervisor)

Assoc. Prof. Dr. Oğuz Bayat	Altinbaş, University	________________
Asst. Prof. Dr.Adil Deniz Duru	Marmara, University	________________
Asst. Prof. Dr.Çağdaş Dogu Atilla	Altinbaş, University	________________
Asst. Prof. Dr.Çağatay AYDIN	Altinbaş, University	________________
Prof. Dr.Hasan Balik	Yildiz, University	________________

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

Asst. Prof. Dr. Oğuz ATA

Head of Department

Assoc. Prof. Dr.Oğuz Bayat

Director

Approval Date of Graduate School of Science and Engineering: ____/____/____

I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Waled Milad Abulsasem Alashheb

Signature

DEDICATION

This thesis is for my father(Milad), who taught me that the best kind of knowledge he has is what I learned for him and as long as he wishes to reconcile and pray for me to achieve the dream of obtaining a master's degree and continuous support and motivates me even in the most difficult moments. It is also dedicated to my mother who taught me that even a bigger step can be accomplished. And to my friends who have always supported and motivated me.

ACKNOWLEDGEMENTS

To everyone who helped me through study at every stage of education and learn. My deepest acknowledgements to my supervisor Assoc.Prof.Dr. Oguz Bayat , for his advice and supervision during my thesis prepare. My very thanks to University for providing me all the facilities during my study and work in my thesis. I would thank my country Libya for the unlimited support. I present my thanks to my family for their patience and encouragement, whose give love and guidance is with me in whatever I pursue. My final words go to express my deepest appreciation to my friends for their unconditional support to me.

ABSTRACT

COMPARISON OF OPTIMIZATION ALGORITHMS FOR THE SOLUTION OF TRAVELING SALESMAN PROBLEM

Author’s Alashheb Waled

M.S, [Information Technology],Istanbul Altınbaş University.

Supervisor: Assoc.Prof.Dr. Oguz Bayat

Co-Supervisor: Assist.Prof. Dr.Adil Deniz

Date: [May 2018]

Pages: 65

The Theory of computational complexity is an essential branch of study in the science of theoretical computing and mathematics, the resolution of P and NP problems is one of the main problems that have open solutions, for which no famous efficient algorithm exist. The Problem of Traveling Salesman (TSP) is an example of these problems. In this problem, a count of specified cities must be visited by traveling salesman, starting and ending point is the same city. In the (TSP) the aim is to get a tour of all nodes so that the complete distance or time is minimized. The application of Evolutionary algorithms is one of the famous methods to solving problems of TSP. These algorithms are usually simulates naturally occurring phenomena in nature, which are employed in modeling algorithms of computer. Currently there exist several of such algorithms; for example, Optimization of Ant Colony (ACO) and Genetic Algorithm (GA).
In this thesis, we analyzed the solution of TSP by GA and ACO and compared between the approaches after gathering solution results. The obtained results from our experiments showed that the ACO is better than GA since it requires less execution time for the same problem.

Keywords: Problem of the traveling Salesman, Optimization, Genetic Algorithm, Optimization of Ant Colony

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

ACA	Ant Colony Algorithm
ACO	Ant Colony Optimization
ACS	Ant Colony System
AIS	Artificial immune systems
AS	Ant System
B&B	Branch and bound
BCO	Bee Colony Optimization
CX	Cycle crossover
DM	Displacement Mutation
DTSP	Dynamic Traveling Salesman Problem
EAs	Evolutionary Algorithms
ERX	edge recombination crossover
ESEM	Ends Exchange Mutation
GA	Genetic Algorithm
IM	Insertion Mutation
IVM	Inversion Mutation
IWD	Intelligent water drops
LP	Linear Problem
MMAS	MAX−MIN Ant System
mTSP	Multiple Traveling Salesman Problem
NP	Non-Polynomial Problem
OX	Order Crossover
PMX	Partially Mapped Crossover
PP	Polynomial Problem
PSO	Particle swarm optimization
QAP	quadratic assignment problem
REM	Reciprocal Exchange Mutation

RESEM	Reverse Ends Exchange Mutation
RESM	Reverse Ends Mutation
RWS	Roulette Wheel Selection
SA	Simulated Annealing
SCP	Set Covering Problem
SCSP	Shortest Common Supersequence Problem
SIM	Simple Inversion Mutation
SM	Scramble Mutation
SMTWTP	single-machine total weighted tardiness problem
STSP	Symmetric Traveling Salesman Problem
TS	Traveling Salesman
TS	Tabu search
TS	Tournament selection
TSP	Traveling Salesman Problem
VRP	Vehicle Routing Problem
VRPTW	VRP with time window constraints

1 INTRODUCTION

Several groups of complexity are existing to differentiate problems according to their “difficulties”. Category of Polynomial (P) contains all of that problems which could be solved on a machine of deterministic Turing, in time that is polynomial from the size of the input. While group of Non-Polynomial (NP) involves all those problems, the algorithm that is exponential can be written for problems of category NP. Furthermore, group of NP-complete contains problems such that an algorithm of type polynomial for one of them could be transformed to polynomial algorithms for solving all other problems of kind NP.

Lastly, the group NP-hard can be assumed as the group of problems that are NP harder or complete. This class includes problems for which no methods at all can be applied.

TSP, is an example of the furthermost significant problems in combinatorial optimization and belongs to the problems of class of NP-Hard. The thought of the problem of traveling salesman (TSP) is to get a tour through a count of specified cities, going to every city just once and returning back to the city which he started from it where the distance of this tour is to be minimum as possible [1].

Presently the only identified method that assured to solve optimally the problem of traveling salesman of every size, is through computing every tour that is possible and inspection for the tour with lowest cost. Therefore , the TSP is one of the best intensively studied problems in mathematics of computation, commanded much attention of computer scientists and mathematicians for years and still remains an open question whether or not an effective common solution approach exists [2].

There are different solving algorithms to solve TSP, like Genetic Algorithm and Branch and Bound. Since it has a hard method to be solved. As a result, users should identify the kind of algorithms and approach of their solution so well to overcome of these types of problems [3].

In this concept, the analysis of two most common solution algorithms of the TSP which are Genetic Algorithm (GA) and Optimization of Ant colony (ACO) are studied and applied in this thesis.

1.1 TRAVELING SALESMAN PROBLEM

1.1.1 Definition

Given a group of specified cities together with the distance or cost of travel among every pair of them, the problem of traveling salesman, or TSP for short, is to get the shortest or cheapest way to return to the city which started from it after visiting all the cities. Many variations of this problem exist, but this is one of the most general forms. A predecessor to this formulation was specifically the distances between cities, but this was expanded to include costs. The path or order of the visited cities is called a tour [4].

The mathematical formulation of this problem assumes that there is a set of different cities $\{K_1, K_2, \ldots, K_m\}$ and an edge is there which corresponding for every couple of cities $\{K_i, K_j\}$ and a path that is closed $K_a = \{K_{a(1)}, K_{a(2)}, \ldots K_{a(M)}\}$. The aim is to get a collection of cities such that the total cost for the salesman is decreased. The minimum possible cost is called the optimal cost. The objective function is given in Eq. (1.1):

$$\sum_{j=1}^{M-1} d(K_{a(j)}, K_{a(j+1)}) + d(K_{a(M)}, K_{a(1)}) \tag{1.1}$$

It also can be known as specified group of m cities, named $\{c_1, c_2, \ldots, c_m\}$, and permutations, σ_1 ,..., ,σ_m !, the goal is to select σ_i such that the summation of totally distances of Euclidean in a tour among cities is reduced.

TSP topic can be solved through enumerating (m-1)!/2 to calculate number of possible routes, where m is the count of cities and then choose the route that has the length which is the shortest among all routes [5] .

The running time of the problem grows exponentially with respect to the count of cities, i.e. the count of possible solutions is n!. In a situation of a few cities (e.g. three or four) the problem will not be very difficult. While in situation of more cities (e.g. nine or ten) it is possible to utilize several simple approaches even it will attempt out all of solutions that could be possible. But what about the situation where there is thousands or tens thousands of cities, it will be very complicated.

1.1.2 History of TSP

The Euler was the beginner of TSP story. In 1766, his researches “Problem of Studied the Knight’s tour” was emerged point of TSP. After that, the significant famous study was accomplished by the two researchers of mathematic science Sir William Rowaw Hamilton from Ireland and Britain Thomas Penyngton Kirkman in 18th century. It was about getting circuits and paths on the dodecahedral graph, in which assured conditions was satisfied. Furthermost of research into history of TSP as a whole was achieved in the period from 1800 to 1900. It could be also pointed to Tucker’s, Menger’s or Kirkman’s researches in old history as well. Then, Lenstr, Lawler, Shmoys and Rinnoy Kan studied the TSP. Karl Menger offered notice to his colleagues in 1920. Afterward that, Merrill Meeks Flood obtained result associated to TSP in 1940. At the 1948 when Flood solved the problem of travelling salesman by awarding it at the Corporation of RAND, Rand is a procedure or function which can produce a number between 0 and 1 randomly. After those years the TSP expanded in popular. In 1950, the Programming of kind Linear was considered as a necessary force in computing solutions to problems of combinatorial optimization. Later on Dantzig, Fulkerson, and Johnson invented an approach for solving the problem of TS in 1950. They verified the success of their approach by solving a 49-city instance. However, it became obvious, as early as the 1960's mid, that the general case of the TSP could not be solved in polynomial time utilizing techniques of Linear Programming. Lastly, these classes of problem became identified as NP-hard. Unlimited progress was made in the 1970's and 1980's late, when Rinaldi, Padberg, Grötschen and others accomplished to exactly solve example with up to 2392 cities, using branch-and-bound and cutting planes [3].

1.1.3 Applications of TSP

Some common applications of TSP are:

a. Wiring of Computer: Some computer systems which were at earlier stages were as modules and pins. It is required to link the pins with series length of wires. The length of the wire must be minimized.

b. Punching Hole: In some manufacturing process, it will be required to punch hole on metallic sheets. The objective is to have minimum number of punches and maximum holes in it. Such a problem occurs in circuit board manufacturing industries. Problems are of large scale.

c. Simple job sequencing: If there are n jobs and it must be performed in a single machine with minimum change over time, the best sequence is figured out by TSP.

d. Design of Dartboard: Dartboards is a game with circular targets. It has concentric circles and 20 sections from 1 to 20. The aim of the game is to reduce the total value 301 to zero by scoring the values on the board. The application of TSP here is to design the dart board with scores from 1 to 20 depending on the toughness of the game.

e. Crystallography technique: The X-ray intensity measurements on a crystal is taken by means of detector for some experiments. This is called crystallography. The detection requires the sampling of the crystals and the placement of crystals in detector appropriately.

The order of measurement is the solution of TSP. The applications are very high and if the best solution are found, then it would be greatly reduce the time taken to undergo these measurements [6] .

1.1.4 Algorithms used to solve TSP

A number of variations exists for the TSP. In Symmetric TSP (STSP) each city pair have the same distance for both directions. Asymmetric TSP is similar to the symmetric one but no path exists for one direction or the distance between two cities is not equal [7]. We will focus on STSP in this thesis.

There are variation of computational tasks that exist and several of algorithms or methods developed to solve TSP. These can be put into separate classes taking into account the style of finding the results. The first class called exact algorithms that produces results which are provably correct. Second class, called heuristic algorithms, which either offer a solution not for all cases of the problem or produce nearly the right answer. These two classes are referred as classical algorithms.

Of course, the key difference between exact and heuristic algorithms is the speed of computation and quality of the results. Exact algorithms are very precious from the computational time point of view, but deliver absolutely correct results. On the other hand, heuristic algorithms are speedy, but their results are approximation to correct result and the correctness of the result is not provable [1].

Classical approaches for solving the TSP generally result in complexities of exponential computation. Therefore, group of algorithms are demanded to beat this shortcoming. These algorithms contain dissimilar categories of optimization algorithms that are nature based, techniques of optimization, algorithms of optimization that are depend on population and etc. In the following we will describe in summary some of these algorithms.

1.1.4.1 Exact algorithms

Approaches that present exact answer for the problem of TS are solution methods that can get solutions that is optimum through process of search which is enumerative. Exact approaches for problem of TS are not appeared to be suitable solution methods for the large count of cities since it needed much time for computation. They may find the optimal tour but the time complexity is not satisfactory especially when the count of cities is more than 1000, i.e. $N > 1000$.

a) Brute-force

This algorithm tries all the possible routes/all possible combinations of cities to visit while summing distances between the cities in particular route and finally find the route with shortest total distance. This solution is also called exhaustive search. It needs to generate (n-1)! permutations of n cities and calculate the total distance of it.

The count of points and permutations for solve TSP is shown in table 1.1, as n increases, the (n-1)! Increases to be greater than all functions of polynomials and exponential. This enormous growing of possible routes means enormous growing of time needed for solving TSP even for contemporary computers [1].

The data involve weights given to the edges of a graph that is finite complete, and the objective is to get a route crossing through all the vertices of the graph that have the lowest total weight. For instance, given the diagram illustrated in figure 1.1, the route that has cost with lowest value would be the one written (F, G, H, J, I, F), the cost is 31 [8].

Table 1.1: Number of permutations for solve TSP

Number of	**Number of possible**
4	24
5	120
6	720
7	5 040
8	40 320
9	362 880
10	3 628 800
11	39 916 800
14	87 178 291 200
15	1 307 674 368 00
16	20 922 789 888 000
18	6 402 373 705 728 000
19	121 645 100 408 832
20	2 432 902 008 176 640
25	15 511 210 043 330 985

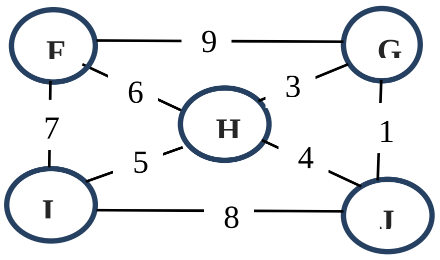


Figure 1.1: The tour with sequence F=>G =>H =>J =>I => F is the optimal tour

b) Branch and bound

Branch and bound is an approach that presented by Doig & Land around 1960s, it is an algorithm of exact kind used in solving problems of various optimization. This algorithm tries to get the solution that is the optimal with a process of splitting up known as "branching" and a stage of investigation known as "bounding". While the problem is divided into sub problems through the operation of branching, the branch to be completed is chosen through operation of bounding.

In this approach, the problem is tested with sub problems and bounds are specified for each sub problem not to be chosen that will give a solution result which is bad. The essential thought of the algorithm is that bounds perform the compare among new obtained solutions with the solutions that obtained previously and not permit for creating a search around a solution that will obviously not yield a solution that is optimal [8].

c) Cutting planes

Cutting planes approach attempts to get solution of the problem with the hypothesis that is depend on idea of getting integer values for objective function and variables. In this algorithm, in first stage, the problem of LP relaxation is solved to obtain a lower bound. After that a row of source of the solution from relaxation of LP is selected. A cutting plane then eliminates a group of solutions that have non-integer values which specified depending on the row of source. By inserting cutting plane to the simplex tableau, LP problem is solved. If all of the variables have integer values, this means that the solution which is optimum is found.

d) Branch and Cut

This approach arise from the combination of the cutting planes approach and the method of B&B. Bounds in the Branch and bound approach are specified considering to the obtained solution of the cutting planes; thus it guarantees attempting to get the solution that is optimum in space of a feasible solution which is smaller.

1.1.4.2 Heuristic algorithms

Heuristic methods generally advocate some estimates to the solution of problems of optimization. These algorithms have the ability to offer a satisfactory solution to a problem, but for which there is no formal proof of its exactness. Instead, it might be right, but might not be depend on it to offer a solution that is optimal. Two essential objectives in science of computer are getting methods with quality of provably good or solution that is optimal and with provably good run times. A heuristic is an approach that give up to one or both of these aims.

Each heuristic solution for TSP can be assessed in terms of two significant constraints: the goodness of the tours that it generates and its running time [1].

Heuristic solutions used for the TSP usually consist of two main parts:

• Tour construction mechanism

• Heuristic algorithm which runs iteratively until some stopping criterion has been satisfied

a. Closest neighbor

This is the most explicit and the simplest TSP heuristic. The fundamental to this method is to permanently go to the nearest city. The difficulty of polynomial related with this heuristic method is $O(n^2)$. The method of closest is so similar to algorithm of tree of minimum spanning. The stages of the nearest neighbor are specified as:

1. Choose a random.
2. Get the closest city that have been not visited and go there.
3. Are there any cities that are not visited yet? If answer is yes, go to step 2 and repeat.
4. Go to the first.

b. Greedy

The heuristic of Greedy progressively constructs a tour by repetitively choosing the edge that has the shortest distance and adding it with the distance of the tour as long as it doesn't make a cycle with less than N edges, or the degree of any node is incremented to more than 2. We ought to not add the same edge twice of course. Difficulty of the heuristic of greedy is $O(n^2 \log_2(n))$. Stages of Greedy method are [9]:

1. Arrange wholly edges.
2. Choose the edge that has short distance and sum it with total distance of our tour if it doesn't disrupt any of the above restrictions.
3. Is the tour has N edges in it? If answer is no, repeat from step 2.

c. Tabu search

It is an algorithm based on search of neighborhood which search the solution that is better in the neighborhood of the solution that is exist. In general, search of tabu (TS) utilizes 2-opt swap approach for searching the solution that is the best. A trouble with search of neighborhood method that is simple i.e. only 2-opt or 3-opt swap heuristic is that can simply obtain stuck in a local optimum. This can be averted straightforwardly in method of TS. To avert this, TS saves a tabu list having solution that is bad with bad swap. There are many approaches for executing the tabu list. The running time is major problem with the TS. Furthermost operations for the TSP mostly require $O(n^3)$, leading it to be sore slower than a 2-opt local search [9].

d. Simulated annealing

Simulated Annealing (SA) method has been effectively used and enhanced to produce an estimated solution for the problem of TS. this approach is mainly an algorithm having local search that is randomized like to TS, however it do not permit alteration of route that worsens the solution. For the TSP, [10] offered a baseline execution of SA. Researchers' utilized moves that is 2-opt to get solutions that is neighbor to optimum. Results that are better can be attained in SA, through increasing the time of running of the SA method, this particular operation needs $O(n^2)$ with a large constant of proportionality.

1.1.4.3 Population and nature based optimization algorithms

The optimization algorithms based on population and nature are the methods which are in the class of the recent dependent methods of optimization. The natural systems and creatures which are performing activities and progressing in nature consider as one of the exciting and precious sources of thoughts for constructing and developing algorithms and systems in various scopes of technology and science. Computation of Evolutionary, Optimization of Bee Colony, Maps of Self-Organizing for Adaptive Time, Neural Networks, Systems of Ant and Optimization of Particle Swarm are considering as the methods of problem solving stimulated depending on nature that could be observed.

a. Genetic algorithms (GA)

Genetic Algorithm (GA) works in manner that based on a population. A simple GA begins with a population that is randomly generated of solutions which are candidate. Several (or wholl) candidates after that are mated for offspring to be generated and a number of offspring go through a process of mutation. Every candidate has a value of fitness informing us the degree of their goodness. We will gain the increase in overall fitness of the population through choosing the most fit candidates for mating and mutation. Solving TSP by utilizing GA includes executing a routine of crossover, a measure of fitness, and similarly routine of mutation. The better measure to check fitness of the solution is represented by real length of that solution. There are many methods to perform the routines of crossover and mutation [9].

b. Optimization of Ant Colony (ACO)

The Optimization heuristic of Colony of Ant is stimulated through the real ant conduct in getting the route that is the shortest among the food and the nest. This is done through demonstrating the path of an ant by a element named pheromone. The heuristic knowledge used by ant in its searching which is its private knowledge of place of the food, according to the smell that comes from that food after that, the other ants' will choose the route that lead to the food by information of pheromone. In real in this method a group of individuals (artificial ants) utilized to work together in order to solve the problem through exchanging information by deposited pheromone on edges of graph. Thus the ACO method is utilized to imitate the conduct of real behavior of ants [1].

c. Optimization of Bee Colony (BCO)

The method of colonies of bee simulates the nature as following. At start, the feed is searched by each bee participate to colony in a separate manner. When the feed is founded by a bee, other bees are notified through dancing of that bee. The feed is carried to the hive after collecting it by other bees. After the feed is relinquished to the hive, the bee can perform three activities that are different from each other.

1. Give up previous source of the food and become uncommitted follower again.
2. Without recruiting the nestmates, it remains to be forage at the food source.
3. Before the return to the food source it dance and thus recruit the nestmates.

With a sure probability that is reliant on the gained quality of feed, it's far from the hive and the count of the bees which are currently engaged with this resource of feed, a bee chooses one of the specified activities and continues its job in a similar repetitive form. This conduct can be useful to several problems of complicated engineering containing optimization, transportation, computational control, etc… [11].

Several of such algorithms exist to solve problems such as artificial immune systems (AIS), intelligent water drops (IWD), Particle swarm optimization (PSO), etc... [11].

1.2 THE PROBLEM

The problem of this thesis is how to know the best algorithm for solve the TSP, what criteria must we take into account to know such algorithm, the count of ants when using ACO, count of generations when using GA, and number of cities to be visited affect to the simulation result.

1.3 LITERATURE REVIEWS

Many researchers have studied and proposed solutions to the TSP, here we will mention to some of them, [12] proposed An Improved Optimization of Ant Colony & Genetic Algorithm for Problem of TS. GA and ACO are two methods that consider bionic optimization, they are also two effective and powerful approaches for solving the problems of combination optimization, and furthermore they both were effectively employed in TSP. The proposed work syncretizes two approaches; meanwhile, a syncretic approach is put forward. The outputs of simulation demonstrate that the methods of Ant Colony Algorithm (ACA) and Genetic Algorithm (GA) are better at quickening the speed of convergence and improving global convergence. (GA) and (ACA) enlightened by biologic evolutionism are put forward as a type of bionic methods.

[13] Proposed Optimization of Ant Colony (ACO) and Genetic Algorithm (GA) as Mixture Algorithm to solve DTSP. In this, correctness of method and distance that travelled for DTSP has been measured. Attained results recommend that Mixture approach does not well-known easily in the local optimum and, in convergence for comprehensive answer it possesses a better speed. The goal function is proposed to minimize distance of the travelled cities. Hence, and according to travelled cities it is possible to select the correctness, in this method that for shorter distances more amount of appropriateness is taken into account.

[14] Presented optimizing delivery routes using Genetic Algorithms. The proposed methodology of route scheduling of optimizing delivery using GA to solve the Problem of Multiple Traveling Salesman (mTSP). The problem of traveling salesman (TS) is a problem that is combinatorial optimization where a salesman need to find the route with shortest distance to n cities and return back to start city. With procedures of genetic algorithm and robust clustering offers a best chance to solve this category of troubles.

[15] By using genetic algorithm they suggested flexible approach for solving the problem of travelling salesman. The proposed work gives a solution which contains execution of a genetic algorithm so as to give a maximal approximation of the problem with the decrease of cost. Crossover is an important operator in genetic algorithm for TSP. Lot of efforts in order to determine an appropriate crossover operator must be done. The efficacy of operator to be used in crossover is compared against some operators used in existing crossover. The aim of this proposed work was to compare the efficiency of the crossover operator against some crossover operators that are existing.

[16] Offered a proposed method which includes Optimization of Ant Colony, Genetic Algorithm and mixture of ACO and GA. In that paper route that is the best, its length, jitter factor and delay of propagation are attained for ACO, GA and Hybrid.

While [17] presented a paper which includes three intelligent methods, namely, Hopfield neural network, basic genetic algorithm, and elementary ant colony to solve the TSP. After that different algorithms are compared depending on the perspectives of complexity of time, complexity of space, the drawbacks and benefits of the calculated results, and hardness level of realization.

1.4 AIM

The objective of this thesis is to analyze and compare a solution for TSP using ACO and GA, since these two algorithms are recent techniques for approximate optimization, GA is belonging to algorithms of population based optimization while ACO is a type of optimization algorithm that is nature based. And also the aim is to show which one require less time and short distance to find the most approximate solution that approaches to optimal solution, supporting a number of cities to visit and produce the result (best possible route) with a random number of ants and generations in every execution.

1.5 ROADMAP OF THE THESIS

In Chapter 2, we describe several subjects that are related to the relevant definitions, major information, and the procedures of the two methods to solve TSP which are Genetic Algorithms and Ant Colony Optimization.

While in chapter 3, we will demonstrate the methods that are used for TSP together with their flowcharts.

After that, in chapter 4, we show the results of implementation of the two algorithms after applying real data, along with analyzing the implementation.

Lastly, in chapter 5, we talk about discussions, conclusions of this research and offer proposals for the future works and researches.

2 METHODS

Evolutionary Algorithms (EAs) are techniques of search which obtain their revelation from chose of nature and as exist in the world of nature the fittest will survive. Solutions that are represented as population used by EA to make a search process. Every repetition of an EA include chosen among wholly solutions in the population depending on a competitive which leads to the continuing existence of the fittest and poor solutions are eliminated from the population.

During last 50 years four main paradigms of evolutionary algorithm have been presented: genetic algorithm is an approach of computation, suggested mainly in 1975 by Holland. In 1981 strategies of Evolution developed by Rechenberg and Schwefel. While in 1966 Fogel invented Evolutionary programming, and lastly in 1992 Koza was proposed genetic programming. For solving the TSP, we will apply the GA.

Besides that, in 1992, Beckers was proposed the heuristic of ACO (Ant Colony Optimization) which based on the idea that the shortest path between the nest and the food is found by real ant. A group of artificial ants are utilized by the algorithm, which cooperate to find solution for a problem by exchanging information via deposited pheromone.

Generally, three perfect bio-inspired algorithms for TSP are (GA), (ACO) and (PSO) [18].

2.1 GENETIC ALGORITHM

Genetic Algorithms are fundamentally techniques of evolutionary optimization where we attempt to mimic to process of evolution. GA is utilized as mechanism of evolution to produce a solution for the random choosing. In GA, for the problem the chromosomes represent possible solutions to it. A group which consists of chromosomes is called population. A genetic algorithm represents an iterative method which generally operates on a population that has fixed size. Adequacy rate is determining the value of solution and is calculated with utilizing function of sufficiency. Genetic operators like evaluation, crossover and mutation can be used to obtain the population [3].

Genetic algorithms are consider stochastic approaches, which don’t present assurance convergence; usually reaching a maximum number of generations or finding an acceptable solution or more sophisticated termination criteria indicating premature convergence will trigger termination of the algorithm [19].

By exchange parts of a solution with another new produced, recombination is implemented and new solution will have formed that it may be better than the prior solutions. Besides, a solution could be mutated by influence a portion of it. Mutation and recombination are applied to improve the population near regions of the space which good accepted solutions may exist in it [18]. The overall simple genetic algorithm shown in Figure 2.1.

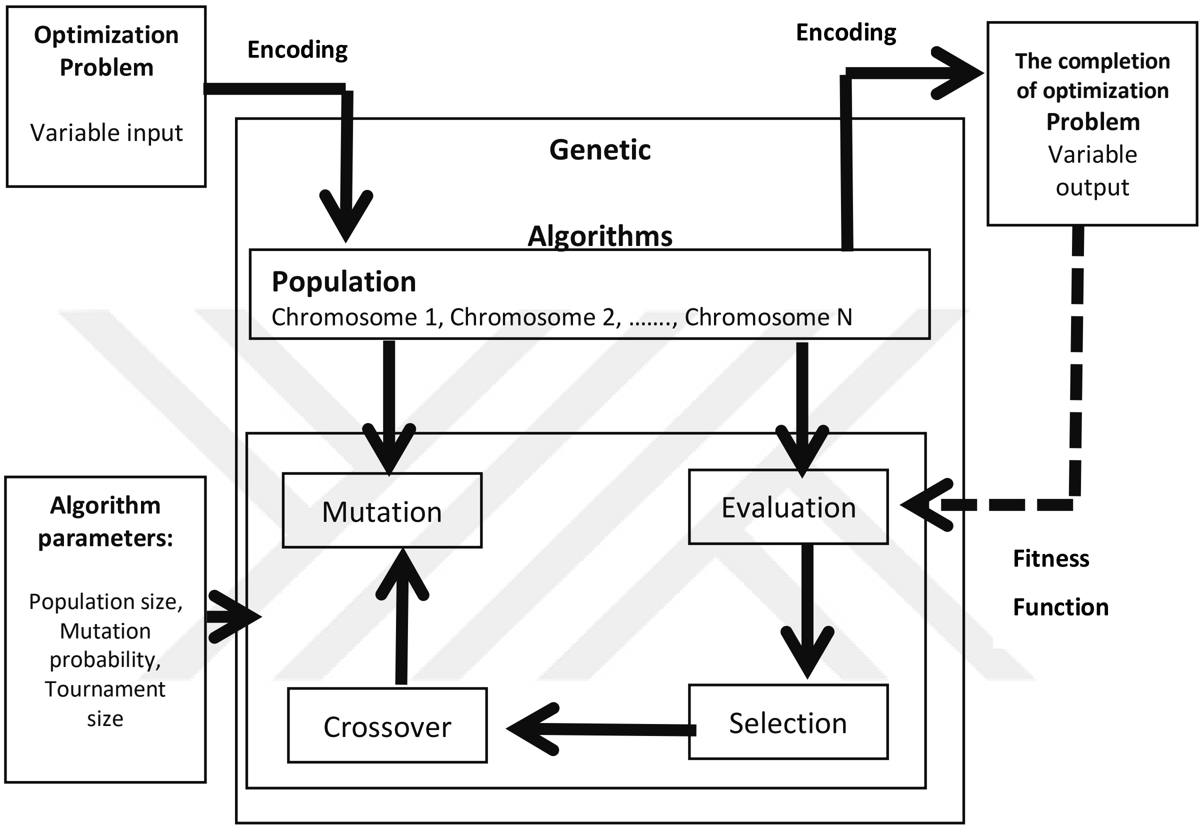


Figure 2.1: Overall structure of GA

The basic procedure of getting solution for GA is as following:

1. [Begin] Complete the graph which is containing N initial population heuristically or in random manner.
2. [Sufficiency] Evaluate the fitness for every possibility, utilizing mechanism of choice.
3. [Population] Generate population which is repeated in the next steps.
   a. [Selection] Select two populations which own highest fitnesses.
   b. [Crossover] Generate population by using operation of crossover.
   c. [Mutation] Mutate the created population according to any chance.
   d. [Addition] Add the best one in the new population.
4. [Altering] The new population will substitute the old population.
5. [Test] Find the finest solution in new population if it exists, or go back to step 2 [3].

2.1.1 Problem Representation

Researchers have proposed several alternatives to represent chromosomes in GA for TSP. Well known representations are representation as binary, representation as path, representation of adjacency, representation of ordinal, and representation of matrix [7], following will describe first three of these representation.

2.1.1.1 Binary Representation

There are different techniques how to utilize representation of binary for the problem of TS, in the most plain represent, every city will represented as a string of $\log_2 n$ bits and a string of $n(\log_2 n)$ bits will represent the solution. Then simply the crossover is implemented through using crossover of type single-point. A tour of (4, 5, 2, 7, 3, 1, 6) can be represented as figure 2.2:

$$\underbrace{100}_{4}\ \underbrace{101}_{5}\ \underbrace{010}_{2}\ \underbrace{111}_{7}\ \underbrace{011}_{3}\ \underbrace{001}_{1}\ \underbrace{110}_{6}$$

Figure 2.2. Represent of a tour of (4,5,2,7,3,1,6)

Another way for binary encoding is by using binary matrix representation, for example, a component in the matrix, in j-th column and the i-th row to be 1 if and only if city i is passed before city j in the tour; crossover of type based on two or one point could be performed on the matrices of parent, by using crossover based on one-point the tour of child is generated through just using the left column vectors of the point of crossover from one parent, and using the other parent to choose the right column vectors of the crossover point.

2.1.1.2 Representation based on Adjacency

With the representation of adjacency for the TSP, in the position i, a city j is placed if satisfied that the tour leads to city j after city i. According to the representation of adjacency, the crossover based on changing edges has been suggested as example, which principally achieved as follows:

First from one parent, an edge is selected and in the other parent it continues with the cross of this edge representing the following edge, etc. Thus the partial tour is created by selecting edges alternatingly from the two parents. If a cycle is generated through using this approach, the edge is omitted and not added and randomly the operator will choose an edge from the available edges which do not generate a cycle with continuing in the mode explained above.

2.1.1.3 Representation based on Path

The best TSP tour natural representation is known by the representation of path. In representation of path, the n cities of a tour are placed in sequence depending on a list has length of N, after that the sequence of cities to be passed is specified by the list entries with an imaginary edge to the last after start from the first list entry. Many operators of mutation and crossover have been enhanced depending on this representation, and currently many of the applied approaches for solution of TSP utilizing GAs are achieved using this representation. In spite of clear drawbacks of this representation like the equivocality (one tour can be represented in 2n dissimilar methods for a TSP that is symmetric and in n dissimilar methods for an asymmetrical TSP). This representation has permitted the performing of operators that are quite powerful like the edge recombination crossover (ERX) (OX) or the order crossover which will use subtours of parent to the solutions of child [19].

Our Simulated algorithm: uses path representation. There exist also other representations; such as adjacency, ordinal and matrix representations. But, these representations are not used anymore for GA approaches.

2.1.2 Genetic Operators

The following sections will explain the key genetic algorithm operators, which is parent mutation, replacement, crossover, and selection.

2.1.2.1 Models for Selection of Parent

Every individual in a population is given a value by applying function of fitness; the goodness of the solution is specified using the given or calculated fitness value that an individual represent. The function of fitness is often included in the description of the problem or according to the function of objective; or may also include the applying of knowledge of an expert, techniques of heuristic, or the simulation.

A population is once will be generated and fitness of it will be calculated, the group of solutions, that are chosen to be “mated” in a presented generation is created. In the genetic algorithm that is standard (SGA) for reproduction propose, a chromosome of the recent population is chosen depending on its probability that will be proportional to its fitness.

Several methods are exist for performing this selection. The most common three are:

- **Proportional selection (Roulette Wheel Selection RWS):**

RWS algorithm is dependent on the idea that, the individual's chosen likelihood is proportional with its fitness score like in the natural chosen. In RWS, all individuals are sorted according to their fitness scores (Lines 1-2). We calculate an array of sum of fitnesses for the population (Line 3). *sumOfFitnesses*[i] stores the sum of fitnesses of all individuals indexes through 0 to i. Then, we generate a random value between 0 and last element of the array which is sum of all fitnesses of the population (Line 4). To select an index, we iterate through the array and look for the first index where the random value is larger than the sum of fitnesses until that index (Lines 5-9).

Algorithm 1: Roulette Wheel Selection

```
• population.evaluateFitnesses()
• population.sortIndividuals()
• sumOfFitnesses=population.generateSumOfFitnesses()
• value= random(0,sumOfFitnesses[lastElement])
• for i=0 to SumOfFitnesses.length do
      if value < sumOfFitnesses[i] then
          return i
      end if
• endfor
```

- **Tournament selection (TS)**:

In TS algorithm, we first evaluate and sort all the chromosomes in a population (Lines 1-2). Then, we simply generate a random number to pick that amount of individuals from the population (Lines 3-4). Then we select the fittest among them (Lines 5-11). Setting aNumber as 1 will result in 100% random selection [7].

Algorithm 2: Tournament Selection

```
• population.evaluateFitnesses()
• population.sortIndividuals()
• shortest = ∞
• aNumber = random(0,N)
• for i=0 to aNumber do
• index= random(0,N)
```

```
        if length(tour[index]) < shortest then
            shortest = length(tour[index])
            selectdIndex = index
        endif
    • endfor
```

- **Rank Selection**

A value of 1 is given to chromosome which has bad fitness, and as the fitnesses of the chromosome get better the value increases to 2 or 3. The objective of in this chosen is given a probability to enable the use of chromosomes that has bad fitness in other places. Drawbacks of this method is only that it may consume a long execution time [3].

2.1.2.2 Recombination (Crossover)

In its simplest formulation, crossover cuts the strings of chromosome of two individuals after choosing it at some position which chosen randomly. The generated substrings after that exchanged to create two new chromosomes with full length.

The most important operator of GA is crossover. Search space is explored globally with this operator. Without crossover, GA wouldn't be much of a global search algorithm.

Numerous crossover approaches have been enhanced for the problem of traveling salesman. The next section will describe some of them.

- **Crossover of type Single point:**

A cut of one place randomly is made, generating two tail sections with the two head sections. After that the two tail sections are exchanged together to create two new chromosomes (individual).

For example, for the chromosomes.

c1 = 1100

c2 = 0111

The crossover point could be selected randomly after the second gene

c1 = 11 | 00

c2 = 01 | 11

Swapping the genes in positions after the point of crossover would result

c’1 = 1111 = 15

c’2 = 0100 = 4

Now two new chromosomes could be generated which could be used in the following population that referred as the next generation.

The function of evaluation assigns to each chromosome a ’value’ which based on it the chromosome’s chance of crossover is decided. Randomly the chromosomes are selected to crossover and the chromosomes that has the highest values has the better chance to be selected.

- **Crossover of type Multiple point:**

There are m points of crossover points in an m-point crossover and sub strings are exchanged with each other among the n points. Crossover of type Multiple-point is further appropriate to combine good features existing in strings since it sampled in uniform manner along the full length of a chromosome. Likewise, we can have crossover with two points. In this situation we select two crossover points randomly and swap the genes among the two points. In previous problem it could be choose the points before the fourth gene and after the first gene.

$$c1 = 1 \mid 10 \mid 0$$

$$c2 = 0 \mid 11 \mid 1$$

to obtain

$$c''1 = 1110 = 14$$

$$c''2 = 0101 = 5$$

- **Cycle crossover (CX)**

Cycle crossover (CX) works in a quite another manner. Initially, this kind of crossover can only be utilized for the first representation that was defined, that is, the chromosome c = 1234 means that salesman begin from city 1 then go city 2 after that visit city 3 and city 4 will be visited lastly. After that we do not select a point of crossover at all. The first gene is selected from one of the parents

$$c1 = 96324758$$

$$c2 = 23456789$$

If we select 2 from c2

$$c'2 = 2 - - - - - - -$$

We must choose each element from any of the parents then put it in the place it was previously in. Because the first place is occupied by 1, the number 8 from c1 cannot go there as 9 from c1 must now picked.

$$c'2 = 2 - - - - - - 9$$

This obligate us to place the 8 in location 8 and the 5 in location 4, as in c2.

c'2 = 2 − −5 − −89

Because the same group of positions is occupied by 2, 5, 8, 9 in c1 and c2, we finish by filling in the blank positions with the elements of those positions in c1. Thus c'2 = 26354789 and we get c'1 from the complement of c'2

c'1 = 93426758

This process guarantees that every chromosome is legal. It could be noticed that ending up with the offspring being the same as the parents it is possible. This is not a trouble because it usually will only happen if the parents have high fitnesses, in that situation, it could be considered as a good option yet.

- **Partially Mapped Crossover (PMX)**

Partially Mapped Crossover (PMX) exchanges information between two parents with swapping mechanism. We select two cutting points from parents. We get sub tours between cut points of the parents.

When we use the two-point crossover with the integer representation for the TSP. If a two-point crossover is implemented on the chromosomes

c1 = 9632 | 475 | 8

c2 = 2345 | 678 | 9

we would obtain

c'1 = 9632 | 678 | 8

c'2 = 2345 | 475 | 9

Which are clearly illegal since c'1 does not go to cities 4 or 5 and go to cities 6 and 8 twice. Similarly, c'2 does not go to cities 6 or 8 and goes to cities 4 and 5 twice.

PMX overcomes this trouble through noticing that it performs the exchanges 6 - 4, 7 - 7 and 8 – 5 and after that repeating these exchanges on the genes outside the crossover points, producing

c"1 = 94326785

c"2 = 23684759

It performs the exchanges, 6 - 4, 7 - 7, 8 - 5 and the rest of elements not changed. c"1 and c"2 yet be composed of portions from both the parents c1 and c2 and are now both valid.

This crossover will be better when utilized with the cycle representation, because more of the structure from the parents would be preserved in this situation. If, as in our example, we applied the first integer representation, the order in which the cities were visited would have altered significantly from the parents to the children – merely a few of the same edges would have been saved. With cycle notation a lot more of the edges would have been moved. Conversely, if we apply routine of this crossover with cycle representation we do not necessarily obtain a legal tour as a result. a repair routine to create a legal tour from the solution that the crossover produce is required to be devised , by altering as little as possible in so as to maintain a similar structure [2].

2.1.2.3 Mutation

Another important GA operator is the mutation operator. It helps the algorithm to jump out of the local optima. Various mutation operators have been developed for TSP, each of which states a local modification of an individual. The operator is completely blind unless there is a special implementation applied for it. Multiple mutations and improving mutations are examples of special implementations [19].

Therefore, diversity is provided by this operator through generations. On the other hand executing this operator with a small random probability protects most of the individuals.

In a traditional GA, if we take mutation out of the approach, most probably lots of applications can no longer produce different individuals after a certain amount of generations. Here, we present the well-known mutation operators [7].

- **Reciprocal Exchange Mutation (REM)**

REM is the classical swap mutation of the traditional GA design. We simply select two cities for the function REM and swap them.

before: (1, 6, 4, 5, 2, 3)

after: (1, 2, 4, 5, 6, 3)

To apply REM, we swap the cities 6 and 2.

- **Insertion Mutation (IM)**

IM is similar to EM rather than a city is eliminated from the tour and inserted into another randomly chosen place consequently. A random city's current and new index are given to the function IM as inputs and removed from the route. At the second stage, the removed city is inserted into a random place.

before: (1, 6, 4, 5, 2, 3)

after: (1, 6, 5, 2, 4, 3)

We remove the city 4 from the individual which leads to the subtour [1, 6, 5, 2,3]. Then, we insert city 4 to a random position of the subtour to form a complete path.

- **Displacement Mutation (DM)**

An extended version of IM is DM where a subroute is exchanged rather than a single city. For the DM function, we have 3 inputs for displacing a random subtour, which is derived from a starting index and length, is removed from the route. At the second stage, the removed subtour is inserted to the same route into a random place.

before: (1, [4, 6, 5], 2, 3)

after: (1, 2, [4, 6, 5], 3)

We remove the subtour [4, 6, 5] from the tour. We have [1, 2, 3] remained. Then, we insert subtour [4, 6, 5] into a random position of the individual to form a complete path.

- **Simple Inversion Mutation (SIM)**

SIM is the reversed version of DM. We call SIM with inputs index and length. A random subroute which is obtained from the inputs, selected for reversal and that random subroute is reversed.

before: (1, [4, 6, 5], 2, 3)

after: (1, [5, 6, 4], 2, 3)

We reverse the subtour [4, 6, 5].

- **Inversion Mutation (IVM)**

IVM is a variation of SIM where reversed subroute is inserted to the route just like the insertion pattern followed in IM and DM.

before: (1, [4, 6, 5], 2, 3)

after: (1, 2, [5, 6, 4], 3)

We remove the subtour [4, 6, 5]. Then, place its reversed version [5, 6, 4] into a random place in the individual.

- **Scramble Mutation (SM)**

Scramble Mutation has the maximum number of enhancements.

before: (1, [4, 6, 5], 2, 3)

after: (1, [6, 5, 4], 2, 3)

We scramble the subtour [4, 6, 5] leading [6, 5, 4].

- **Ends Exchange Mutation (ESEM)**

ESEM is similar to DM but we use it twice for the ends of the individual.

before: ([1, 4], 6, 5, [2, 3])

after: ([2, 3], 6, 5, [1, 4])

We simply swap two subtours [1, 4] and [2, 3] from the ends of the individual

- **Reverse Ends Mutation (RESM)**

RESM is similar to SIM but we use it twice for the ends of the chromosome.

before: ([1, 4], 6, 5, [2, 3])

after: ([4, 1], 6, 5, [3, 2])

We reverse the subtours [1, 4] and [2, 3] from the ends of the chromosome.

- **Reverse Ends Exchange Mutation (RESEM)**

RESEM is similar to IVM but we use it twice for the ends of the individual. The subtour length should be <= N/2.

before: ([1, 4], 6, 5, [2, 3])

after: ([3, 2], 6, 5, [4, 1])

We both apply reversing and swapping to the subtours [1, 4] and [2, 3].

2.1.2.4 Replacement

In the stage of replacement, child and their parents are assessed depending on their solution accuracy. Several and various kinds of replacement approaches are now exist. An example of that is adding new generated solution to the population without estimation of how much it fits to the aim. Another example is to choose best two of solutions and eliminat worst one while they are added to the new population [8].

- **Elitism:**

The individual that is the finest (or respectively the n individuals that are the best) of the preceding generation are reserved for the following generation. The special and commonly used method of just detained is the best one individual of the latest generation that also referred to it as the “golden cage model,” which is a specific situation of n-elitism with n = 1. If mutation is performed on the elite so as to avert convergence of premature, the replacement method is named as “weak elitism.”

- **Delete - last- n:**

The m individual that are the weakest are substituted by m descendants.

- **Delete - n:**

It different from the method of delete-n-last, in this approach not the n weakest but rather n arbitrarily selected individuals of the previous generation are substituted, which in one side it decreases the speed of convergence of the algorithm but on the second side likewise assists to prevent premature convergence (weak elitism versus compare elitism).

- **Replacement of Tournament:**

Tournaments are run among groups of individuals composed from the current and the last created generation, the winners will take part on the creating of the new produced population [19].

2.2 ANT COLONY OPTIMIZATION

Algorithms of Ant colony optimization (ACO) are swarm intelligence methods, and they are inspired by the conducts of real behavior of ants searching for good sources of food (figure 2.3). First introduced in the early 1990s, they were initially used to solve computational problems including traversal of graphs. A colony is a population of not dependent, simple, and asynchronous agents that collaborate to find a better solution to the problem at hand. The ants in ACO are processes that construct stochastic solution, it probabilistically and iteratively produces complete solutions from smaller components. It is this collaboration on a colony level that assigns to the members of the algorithm of ACO family their distinctive flavor.

There are several ACO algorithms. The domain of ACO algorithms is enormous, and they have been used in the dozens of various NP-hard computational problems, often performing a state-of-the art efficiency. Originally, the Ant System was first ACO algorithm that used to solve Problem of Traveling Salesman. Since then, they have been used to, for instance, problems of routing, problems of scheduling, problems of assignment, problems of subset, problems of machine learning, protein folding, and routing packets in telecommunications networks. ACO approaches may be a particularly viable method for solving “ill-structured” or highly dynamic problems [20].

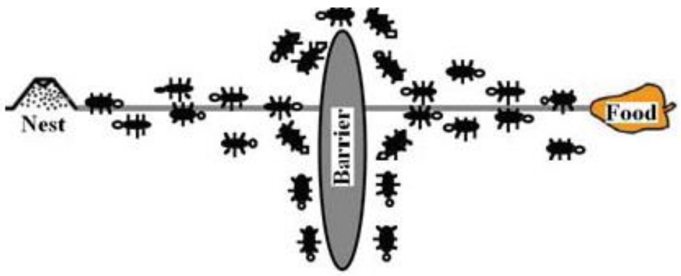


Figure 2.3: Actual behavior of the ant in finding the path that is shortest between the food and the nest

2.2.1 Basic algorithm of ant colony optimization

The essence of ant colony system method is based on idea that pheromone which is a private chemical trail is left on the ground by ant during travel, which directs the other ants towards the objective solution. In case when more ants go through the tour more pheromone is left, which enhanced the likelihood of other's ants selecting this tour. Moreover, the pheromone has a lessening action since trail will evaporated over time. Besides that, the count of ants using this trail will specify the amount of pheromone left by ants.

Figure 3 demonstrates a process of decision-making of ants selecting their trips. When ants meet at point of their decision-making A, randomly some select one side and some select other side.

Assume these ants are crawling at the same speed, those selecting long side arrive at decision-making point B less quickly than those choosing short side. The ants' selection based by chance the long side are the last to arrive to the nest. Therefore, pheromone will received on the long side later than the short one and this fact raises the chance that further ants will not chose long side but rather the short one is chosen. As a result of this, the amount of pheromone is left with higher speed in short side than long side since more ants' select short side than long side. The number of broken line in figure 2.4 is direct ratio to the count of ant approximately. The system of artificial ant colony is depending on the ides of ant colony system for solving types of optimization problems. Pheromone represent the main element of the decision-making of ants [21].

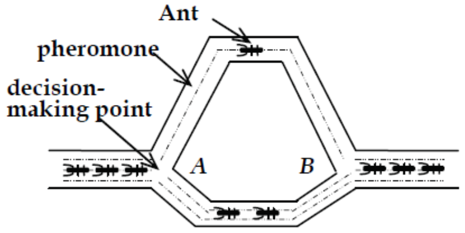
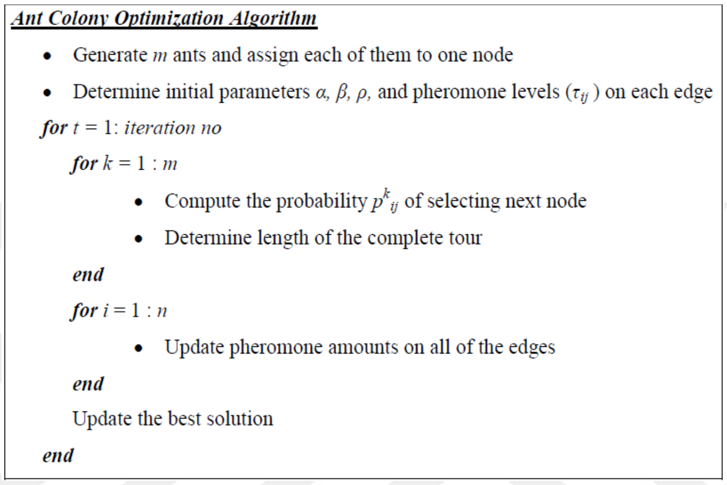


Figure 2.4: Process of decision-making by ants selecting their trips based on level of pheromone

Figure 2.5 demonstrates the structure of ACO algorithm. Algorithm begins with generating paths for all artificial ants and continues with calculating the distances of these paths, updating the quantity of pheromone, keeping the path with shortest distance in memory and lastly finishes when the capability criteria is reached. To perform improved efficiency of the ant systems not merely building the algorithm but it is also proposed to hybridize it with a component of local search [8].



```
Ant Colony Optimization Algorithm
    • Generate $m$ ants and assign each of them to one node
    • Determine initial parameters $\alpha$, $\beta$, $\rho$, and pheromone levels ($\tau_{ij}$) on each edge
    for $t = 1$: iteration no
        for $k = 1 : m$
                • Compute the probability $p^{k}_{ij}$ of selecting next node
                • Determine length of the complete tour
        end
        for $i = 1 : n$
                • Update pheromone amounts on all of the edges
        end
        Update the best solution
    end
```

Figure 2.5: The structure of the ACO meta-heuristic

2.2.2 Applications of Ant Colony Optimization

- **Problems of Routing**: Routing problems include one or more agents visiting a predefined group of locations, and the constraints and objective function based on the order in which the locations are visited. Possibly the best-known example is the problem of traveling salesman (TSP). Algorithms of ACO have been successful in tackling several variants of the problem of vehicle routing (VRP). Also method of ACO for the VRP with time window constraints (VRPTW) was introduced.
- **Problems of Scheduling**: Scheduling problems related to the assignment of jobs to one or several machines over time. Input data for these problems are processing times but likewise often additional setup times, dates of release and due dates of tasks, measures for the tasks significance and constraints of precedence among tasks. Problems of Scheduling have been a main application area of ACO approaches, and the presently available ACO applications in scheduling deal with several different job and machine characteristics. For example, the problem of single-machine total weighted tardiness (SMTWTP) has been solved by ACO.
- **Subset problems**: The objective in subset problems is, generally, to find a subset of the available items that lessens a defined cost function over the items and that fulfill a count of

constraints. This is a wide definition that can contain other sets of problems. However, there are two specific properties of the solutions to subset problems: The order of components in the solution is irrelevant, and the count of components of a solution may vary from solution to solution. A main subset problem solved by ACO is the problem of set covering (SCP).

- **Problems of assignment and layout**: In problems of assignment, a group of items has to be assigned to a given number of resources subject to a number of constraints. Probably, the most usually studied example is the problem of quadratic assignment (QAP), which was among the first problems solved by ACO algorithms.
- **Problems of machine learning**: Various problems in the Machine Learning field have been solved by means of ACO algorithms. Distinguished example is applying ACO to the problem of classification rules of learning.
- **Bioinformatics problems**: Computer applications to molecular biology (bioinformatics) have invented several problems of NP-hard combinatorial optimization. The situation of the problem of shortest common supersequence (SCSP), which is a famous NP-hard problem with applications in DNA analysis has been solved by ACO algorithm, attaining results of state-of-the-art, actually, for structured cases that are usually found in applications of real-world [22].

2.2.3 The method of ant colony optimization solving TSP

In TSP, there is a group of cities and a known distance between each of them. The aim is to find a tour of Hamiltonian that has minimum length on a fully connected graph. This aim is exact like the ants that want to find the shortest route between the nest and the food source. The problem is solved by simulating a count of artificial ants moving on a graph that encodes the problem. A variant named pheromone is associated with every edge and can be read and altered by ants. The artificial ants discover the pheromone to find the shortest Hamiltonian Tour in TSP which is the most favorable route.

In all existing algorithms of ACO for the TSP, the trails of pheromone are associated with arcs and therefore τ_{ij} denotes to the desirability of visiting city j directly after city i. The heuristic information is selected as $\eta_{ij} = 1/d_{ij}$, that is, the heuristic desired of moving from city i directly to city j is inversely related to the distance between the two cities. In case $d_{ij} = 0$ for some arc (i, j), the corresponding τ_{ij} is group to a very small value. For evaluation purposes trails of pheromone are collected into a pheromone matrix whose elements are the τ_{ij} 's. This could be performed analogously for the heuristic information.

Tours are created by performing the following plain constructive procedure to each ant:

1. According to some criterion, select a begin city at which the ant is located;
2. based on Probability, create a tour by repeatedly adding cities that the ant has not gone yet, until all cities have been visited, by using heuristic and pheromone values for this purpose;
3. Go back to the start city. After all ants have finished their tour, they may deposit pheromone on the tours they have tracked.

With noticing that in some situations and before adding pheromone, the tours formed by the ants may be enhanced by the performing of a procedure of local search. This high level explanation applies to maximum of the issued algorithms of ACO for the TSP.

This algorithm's scheme is

Procedure of ACO Metaheuristic Static

- Assign values to parameters, initialize trails of pheromone
- do while (condition of termination not satisfied)
- Construct Solutions of Ants
- Perform Local Search % optional
- Perform Pheromones update
- end
- end

After initializing the pheromone trails and other parameters, these ACO algorithms repeated through a major loop, in which at start all of the ants' tours are created, then an optional stage takes place in which the tours of ants' are improved through the application of some local search algorithm, and lastly the pheromone trails are updated. This final phase includes the update of the pheromone trails and pheromone evaporation by the ants to reflect their knowledge of search.

ACO algorithms vary from each other with some characteristics. In comparison, each of them has the main ACO structure but some different features. Under the above framework, several dissimilar versions of the algorithm are suggested. The MAX−MIN Ant System (MMAS), Ant Colony System (ACS) and Ant System (AS) are three of most common ant algorithms.

2.2.3.1 Ant System (AS)

Ant System is the first ACO algorithm. Initially, three different kinds of AS are ant-quantity, ant-cycle, and ant-density. Whereas in ant-density and ant quantity the pheromone is updated by ants immediately after a move from a city to an adjacent one, in ant-cycle the update of pheromone is only performed after all the ants had created their tours and the quantity of pheromone placed by every ant was set to be a function depend on the tour amount. Since ant-cycle accomplished much better than the rest two kinds, we will only explain the algorithm of ant-cycle, denoting to it as Ant System in the next paragraphs. In AS a solution (tour) of the TSP is constructed by every of m (artificial) ants, no local search is exist in AS. (Clearly, adding local search to AS is straightforward process.).

- **Construction of tour**. At the start, every ant is placed on a city that randomly selected. At each stage of construction, ant k performs a probabilistic action selection rule. In particular, the likelihood with which ant k, presently at city i, selects to go to city j at the τ th iteration of the algorithm is calculated by:

$$P_{ij}^{k}(t) = \begin{cases} \dfrac{[\tau_{ij}(t)]^{\alpha}[n_{ij}]^{\beta}}{\sum_{l\in N_i^k}[\tau_{il}(t)]^{\alpha}[n_{il}]^{\beta}} & if\ \ j \in N_j^k \\ 0 & otherwise \end{cases} \tag{2.1}$$

The Control of the relative significance of the pheromone versus the information of heuristic n_{ij}, is calculated by $n_{ij} = \frac{1}{d_{ij}}$, it is a heuristic value that prior exist, where d_{ij} is the remoteness between cities j and i. β and α are two parameters which specify the relative effect of the heuristic information and the pheromone trail, and N_j^k is the favorable neighborhood of ant k, that is the group of cities that ant k not visited yet. The significance of the parameters β and α is the following. If α = 0, the nearest cities are more possible to be chosen,

This means that (multiple beginning cities since ants are initially randomly placed on the cities). If β = 0, means that only pheromone strength is at work: this approach will lead to the fast evolution of a stagnation state with the corresponding generation of tours which, generally, are strongly suboptimal. Stagnation of search is defined as the case where all the ants move on the same route and construct the same solution.

- **Update of Pheromone**. The next after that all ants have created their paths, is the process to update the pheromone trails. This is performed by first lowering the strength of pheromone on all arcs using a constant factor and then permitting each ant to increase pheromone value on the arcs it has visited as in Eq. (2.2):

$$\tau_{ij}(t+1) \leftarrow (1-\rho).\tau_{ij} + \sum_{d=1}^{m} \Delta\tau_{ij}^{b}(t) \tag{2.2}$$

where 0 < p ≤ 1 is the evaporation rate of pheromone trail. The parameter p is used to evade limitless accumulation of the pheromone trails and it allows the algorithm to “forget” formerly selected decisions which it bad. If an arc is not selected by the ants, there will be exponentially decreases on its associated pheromone strength. m refers to the count of ants. $\Delta\tau_{ij}^{b}(t)$ is the quantity of pheromone ant b places on the arcs or edge (i,j) it has visited; it is calculated as following Eq. (2.3) :

$$\Delta\tau_{ij}^{b} = \begin{cases} \frac{Q}{L^{b}(t)} & if\ ant\ b\ used\ edge\ or\ acr(i,j)\ in\ its\ tour, \\ 0 & otherwise \end{cases} \tag{2.3}$$

where Q is a constant , and $L^{b}(t)$ is the distance of the tour created by ant b. the better the ant's tour is, the further pheromone is received by arcs contained in the tour. Generally, arcs which are which are included in shorter tours and visited by several ants will receive more pheromone and hence are have more chance to be selected in future iterations of the algorithm.

In the creation of a solution, ants choose the next city to be visited through a stochastic approach.

2.2.3.2 Ant Colony System (ACS)

System of Ant Colony (ACS) has been presented to increase the efficiency of AS. ACS varies in three major features from ant system. First, the pheromone is added only to arcs contained in the global-best solution. Second, each time an ant passes through an arc or edge (i;j) to move from city j after city i it eliminates some pheromone from the arc. Third, ACS utilizes a more aggressive action selection rule than AS [23].

The ACS was seen as the most effective algorithm on the problem of TSP. The major advantage of ACS is presenting a novel update for local pheromone besides to the global pheromone.

Construction of Tour. In ACS ants select the following city using the choice rule of pseudorandom- proportional action: When placed at city i, ant k moves, with likelihood q_0, to city l for which $\tau_{il}(t).[n_{il}]^{\beta}$ is maximal, that is, with likelihood q_0 the best possible move as specified by the obtained pheromone trails and the heuristic information is done (exploitation of learned knowledge). With likelihood (1 - q_0) an ant accomplishes a biased exploration of the arcs that calculated by using Equation (2.1).

Update of Global pheromone trail. Only the global best ant is permitted to add pheromone afterward every iteration in ACS. Thus, the update is processed by applying Eq. (2.4):

$$\tau_{ij}(t+1) = (1-\rho).\tau_{ij} + \rho.\Delta\tau_{ij}^{gb}(t) \tag{2.4}$$

where $.\Delta\tau_{ij}^{gb}(t) = 1/L^{gb}$. It is significant to notice that the trail update only performed to the arcs of the global-best tour, not to all the arcs similar in AS. The parameter ρ again denotes to the evaporation of pheromone. In ACS, feedback is received only for global best solution. Initially, utilizing the iteration best solution is measured for the pheromone update. Although the alteration in solution quality between using the global-best solution or the iteration-best solution is minimal for smaller TSP instances, but in case of larger instances the use of the global-best tour presents far better results [24].

Update of Local pheromone trail. Besides to the rule of global updating, in ACS the ants employee a rule for local update that they perform directly after having crossed an arc during the creation of tour:

$$\tau_{ij} = (1-\varphi).\tau_{ij} + \varphi.\tau_0 \tag{2.5}$$

where φ, $0 < \varphi < 1$, is the coefficient of pheromone decay, and τ_0 is the initial value of the pheromone. The result of the local updating rule is to make a previously selected arc less favorable for a next ant. In this manner there will be increase in the exploration of not yet visited arcs.

The update of local pheromone is accomplished by all the ants after each creation phase. Each ant performs it only to the last edge traversed.

By using the strategy of local update, there will be decrease in pheromone concentration on the traversed edges. Hence, the subsequent ants are encouraged to select other edges and to generate different solutions. As a result this means that the chance of several ants create similar solutions during one iteration will be less.

2.2.3.3 MAX −MIN Ant System (MMAS)

The MAX −MIN Ant System is an enhancement over the original Ant System. MMAS constructs the solutions exactly in the same manner as in AS, that is, the chosen likelihoods are obtained by using the Eq. (2.1). In the MMAS, pheromone trails are updated only by the best ant and the pheromone’s value is bound.

The key variations presented by MMAS with respect to AS are the following. First, to not allowed of search stagnation, the permitted range of the pheromone trail strengths is restricted to the interval $[\tau_{min}; \tau_{max}]$, that is, $\forall\tau_{ij}\ \tau_{min} \leq \tau_{ij} \leq\ \tau_{max}$. Second, the pheromone trails are assigned by value of the upper trail limit, which means that at the beginning of the algorithm there will be higher exploration. Third, to invest the best found solutions, after each iteration only one ant (like in ACS) is permitted to add pheromone.

Update of pheromone trails. The update of pheromone is performed as follows:

$$\tau_{ij} = \begin{cases} \tau_{max} & if\ (1-p).\tau_{ij} + \Delta\tau_{ij}^{best} >\ \tau_{max} \\ \tau_{min} & if\ (1-p).\tau_{ij} + \Delta\tau_{ij}^{best} <\ \tau_{min} \\ (1-p).\tau_{ij} + \Delta\tau_{ij}^{best} & otherwise \end{cases} \tag{2.6}$$

Where τ_{min} and τ_{max} are respectively the lower and upper bounds imposed on the pheromone and $\Delta\tau_{ij}^{best}$ is:

$$\Delta\tau_{ij}^{best} = \begin{cases} 1/L_{best} & if\ (i,j)\ contained\ in\ the\ best\ tour \\ 0 & otherwise \end{cases} \tag{2.7}$$

where L_{best} is the length of the tour of the best ant.

For the τ_{max} and τ_{min} , they are typically found empirically and tuned on the particular problem considered.

The ant which is permitted to add pheromone may be the global-best solution T^{gb}, or the iteration-best solution T^{ib}. Therefore, if particular arcs are frequently utilized in the solutions that are the best, they will receive a higher quantity of pheromone. Results based on previous experiments have proved that the best efficiency is attained by gradually increasing the frequency of selecting T^{gb} for the update of trail.

Trail limits. In order to not allow for search stagnation to occur, upper and lower limits on the possible pheromone are strengthen on any arc are imposed. In particular, the limits of imposed trail have the impact of indirectly limiting the probability p_{ij} of choosing a city j when an ant is in city i to an interval $[p_{min}; p_{max}]$, with $0 < p_{min} \leq p_{ij} \leq p_{max} \leq 1$. Only if an ant has one single potential option for the next city, then $p_{min} = p_{max} = 1$. Results obtained from experiments proved that the lower trail limits used in MMAS are the more significant ones, because the maximum possible trail strength on arcs is restricted in the long run due to evaporation of pheromone.

Trail initialization. In MMAS the trails of pheromone are initialized to their upper pheromone trail limits. Doing so the exploration of tours at the beginning of the algorithms is increased, because the relative dissimilarities among the strengths of pheromone trail are less pronounced [25].

3 USED METHODS

In previous chapters we explained the TSP and methods for solving this problem, in this chapter we will explain the two algorithms that used in this thesis, Genetic Algorithm and algorithms of Ant Colony Optimization.

The two algorithms were programmed using Java 9 programming language and executed several times under Operating System with 64-bit Windows 10 installed.

The tests are performed on a commodity computer; with processor CPU with type Intel® Core™ i7, 2.80 GHz and 4 GB RAM.

3.1 GENETIC ALGORITHM

This algorithm is programmed using the following five classes:

Driver class: this class contains the main method, initializes the initial route, specifies the number of generations, calls the population and genetic algorithm classes and print the structure of the result after getting the optimal route with the shortest path.

City class: contains the information of the cities that are, name, longitude and latitude with their required methods for setting and getting that information. Also contains a method for calculating the distance between cities using haversine formula then returning this distance.

Population class: has the populations of the routes and their required methods for adding routes to population, sorting the routes according to Fitness value

Route class: contains the route of cities and their required methods, a formula to obtain the fitness value when changed and also an approach for calculating the total distance of the route.

Genetic Algorithm class: this class is the core of the program, contains the attributes, fields and performs all functions and methods of the genetic algorithm for evolvement such as (selection, mutation, crossover, elite, etc.).

The following figure 3.1 illustrate the interaction among the classes. In the following we will explain each of that classes, their attributes and methods.

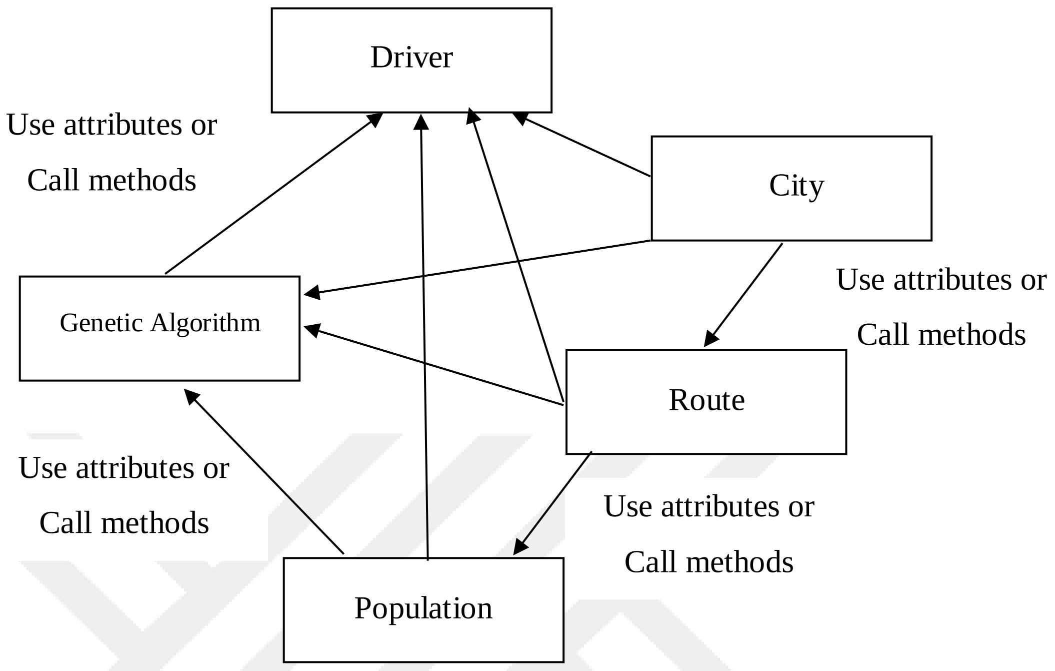


Figure 3.1: The interaction among the GA classes

3.1.1 City Class

This class contains the following attributes:

EARTH_EQUATORIAL_RADIUS: is the radius of the earth and its equal to 6378.13 kilometers.

CONVERT_DEGREES_TO_RADIANS: represents the value of PI/180, its used when converting the degrees to radians.

Longitude: the longitude of the city.

Latitude: the latitude of the city.

Name: the name of the city.

And also contains the following methods:

City (String name, double latitude, double longitude) Constructor: to set the attributes name, latitude and longitude. Then latitude and longitude can be transformed from degrees to radians by multiplying it by $\pi/180$.

getName() : to get the name of the city.

getLongitude():to get the Longitude of the city.

getLatitude ():to get the Latitude of the city.

measureDistance(   ): to measure the distance between the past end City and current City using the formula of haversine.

The haversin formula is an essential equation that used in navigation, it is applied when calculating great-circle distances between two points on a sphere from their latitudes and longitudes.

In spherical trigonometry the measurement of distances is obtained by solving spherical triangles in which sides form arcs of great circles are calculated. As in figure 3.2, arcs of great-circle form the sides of a spherical triangle, and where two arcs intersect which made the angle of spherical. On the Earth, the circles of longitude and equator are considering as natural great circles. Trigonometry of Sphere includes relations between the spherical angles and the lengths of arc (sides) between the arcs. On a spherical surface, a great circle route, often known as a geodesic, is often the shortest path between two points [26].

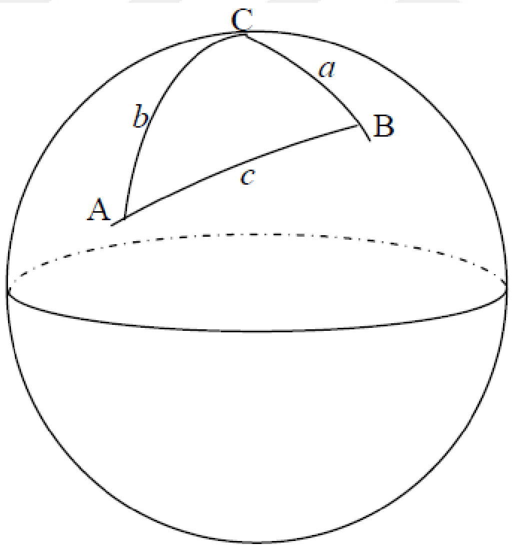


Figure 3.2: Spherical Triangle

Such shortest distance among geographical points and spherical triangles are solved by utilizing the formula of haversine as shown in Equations (3.1), (3.2) and (3.3). Thus in spherical triangle BCA in figure 7 above, given CA or b, CB or a and angle C, the formula of haversine to calculate arc length BA or c is as following:

$$hav(dist) = hav(dlat) + \cos(latA) * \cos(latB) * hav(dlong) \tag{3.1}$$

$$hav(AB) = hav(a - b) + \sin(a) * \sin(b) * hav(C) \tag{3.2}$$

Where:

haversine function can also given by

$$\text{haversin}(C) = \sin^2(C/2) = \frac{1-\cos(C)}{2}$$

Hav(dist): Haversine of distance among locations B and A.

Hav(dlong): Haversine of modification between the longitudes through locations A and B respectively.

Cos(latA) : Cosine of the latitude through location A.

Cos(latB) : Cosine of the latitude through location B.

Hav(dlat): Haversine of difference between the latitudes through locations A and B respectively.

Instead, distance of great circle can be obtained by finding the angle of interior spherical between the two locations and then multiplying that angle by the radius of the earth. Thus the distance S which represent the distance of the side of the spherical triangle in figure 3.3 is calculated by:

$$S = \alpha * r \tag{3.3}$$

Where:

S: Length of Arc (distance of great circle on the sphere)

r: Radius of the sphere.

α : Central angle measure.

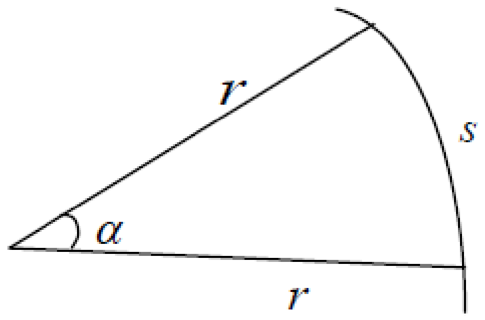


Figure 3.3: Central angle and Arc length (S)

Based on the formula of haversine the central angle in radians is stated as:

$$\alpha = 2\, arcsin\left(\sqrt{sin^2(\frac{\Delta\theta}{2}) + cos(\theta_1) * \cos(\theta_2) * sin^2(\frac{\Delta\emptyset}{2})}\right) \tag{3.4}$$

Where:

α: Interior Spherical angle

$\Delta\theta: \theta_1 - \theta_2$: Difference in Latitude (*dlat*)

θ_1 : Latitude of location 1

θ_2 : Latitude of location 2

$\Delta\emptyset : \emptyset_1 - \emptyset_2$: Difference in Longitude (*dlong*)

For our problem any two cities on earth, the haversine of the central angle between them is expressed by:

$$\text{haversin}(\frac{d}{r}) = \text{haversin}(\phi 2\text{-}\phi 1) + \cos(\phi 1)\ \cos(\phi 2)\ \text{haversin}(\lambda 2\text{-}\ \lambda 1)$$

where:

- *d* is the distance between the two cities (along a great circle of the earth).
- *r* is the radius of the earth. In this situation the radius of the earth is 6.378.137 km.
- $\phi 1$; $\phi 2$: latitude of city 1 and latitude of city 2.
- *λ1; λ2: longitude* of city 1 and longitude of city 2.

On the left side of the equation, d/r is the central angle, assuming angles are measured using radians (*λ* and *φ* can be transformed from degrees to radians by multiplying by π/180).

Solution of *d* by performing the inverse haversine (if available) or by employing the arcsine (inverse sine) function:

$$d = r * haversin^{-1}(h) = 2 * r * \text{arcsin}(\sqrt{h}) \tag{3.5}$$

where *h* is haversin(d/r), or further obviously:

$$d = 2 * r * \text{arcsin}(\sqrt{haversin(\emptyset_2 - \emptyset_1)\cos(\emptyset_1) * \cos(\emptyset_2)\, haversine(\lambda_2 - \lambda_1)}) \tag{3.6}$$

$$d = 2 * r * \text{arcsin}(\sqrt{sin^2\left(\frac{\emptyset_2 - \emptyset_1}{2}\right)\cos(\emptyset_1) * \cos(\emptyset_2)\, sin^2(\frac{\lambda_2 - \lambda_1}{2})}) \tag{3.7}$$

The following is the code for this method:

```
        public double measureDistance(     )
         {   double deltaLongitude = City.getLongitude() - this.getLongitude();
            double deltaLatitude =  City.getLatitude() - this.getLatitude();
            double a = Math.pow(Math.sin(deltaLatitude / 2), 2) + Math.cos(this.getLatitude()) *
                        Math.cos(City.getLatitude()) * Math.pow(Math.sin(deltaLongitude / 2), 2);
            return EARTH_EQUATORIAL_RADIUS * 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1
        - a));   }
```

3.1.2 Route Class:

This class contains the following attributes:

isFitnessChanged: flag to check if Fitness is changed or not.

Fitness: fitness value.

Cities: Array List of cities.

And also contains the following methods:

Route(GeneticAlgorithm geneticAlgorithm) : to fill the initial route with null values.

Route(ArrayList< > cities): to add all cities to array list then shuffle (permute) them randomly.

getCities() : to get list of cities.

getFitness() : to get the fitness value if it is changed according to the following formula:

fitness = (1/calculateTotalDistance())*10000;

calculateTotalDistance() : to calculate the total distance of the route from the origin   to the last
.

3.1.3 Population Class

This class contains the following attribute:

Routes: Array List of routes according to population size.

Also contains the following methods:

Population constructor: to initialize and add routes.

getRoutes() : to get routes.

sortRoutesByFitness() : sort the routes according to fitness value.

3.1.4 Genetic Algorithm Class

It performs all genetic algorithm operations, we used the cities names to encode the cities, and contains the following attributes:

MUTATION_RATE: for mutation.

TOURNAMENT_SELECTION_SIZE: the size of selection.

POPULATION_SIZE: number of routes in the population.

NUMB_OF_ELITE_ROUTES: number of elite routes.

NUMB_OF_GENERATIONS: number of generations, we generate it randomly in every execution.

initialRoute: the initial route.

And it has the following methods:

GeneticAlgorithm(ArrayList< > initialRoute): to set initial Route.

getInitialRoute(): get initial Route.

Population evolve(Population population): to perform the evolve (Crossover and Mutation).

crossoverPopulation(Population population): to perform the crossover operation.

Route crossoverRoute(Route route1, Route route2) : we select the partially mapped crossover, as an example of the crossover route1 and route2:

Route1: [Ankara, Bozuyuk , Antalya, Bursa, Konya , Zonguldak, Bolu, Sakarya, Inegol, Istanbul]

Route2: [Istanbul, Bolu, Zonguldak, Konya, Sakarya, Ankara, Antalya, Inegol, Bozuyuk, Bursa]

Intermediate Route: [Ankara, Bozuyuk, Antalya, Bursa, Konya, null, null, null, null, null]

Final Route: [Ankara, Bozuyuk, Antalya, Bursa, Konya , Istanbul , Bolu, Zonguldak, Sakarya, Inegol]

Route mutateRoute(Route route): to perform the mutation of the route according to random number and MUTATION_RATE. As an example route mutation is:

Original route: [Ankara, Bozuyuk, Antalya, Bursa, Konya, Zonguldak, Bolu, Sakarya, Inegol, Istanbul]

Mutated route: [Ankara, Bozuyuk, Konya, Bursa, Antalya, Zonguldak, Sakarya, Bolu, Inegol, Istanbul].

selectTournamentPopulation(Population population): to perform the tournament selection according to TOURNAMENT_SELECTION_SIZE, population size and a random number, and then sort the selected routes according to Fitness value.

As a termination condition we select the number of generations to finish the evolve process. The following figure 3.4 represents the Flowchart of GA.

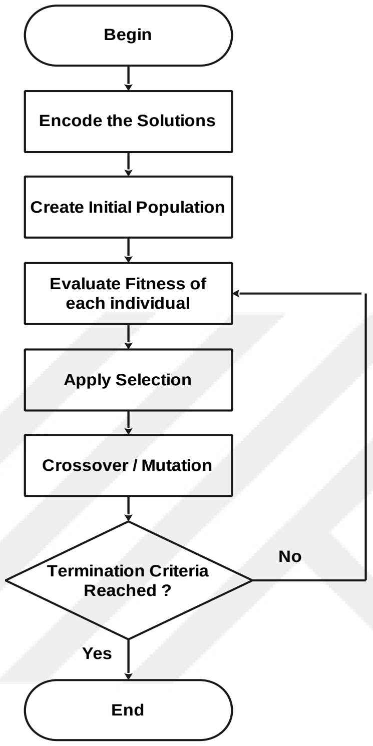

Begin
Encode the Solutions
Create Initial Population
Evaluate Fitness of each individual
Apply Selection
Crossover / Mutation
Termination Criteria Reached ?
No
Yes
End

Figure 3.4: Flowcharts of GA

3.1.5 Driver Class

This class contains the main method, and the following attributes:

Initial Route = Array List of cities of the initial Route, we select 100 Turkish cities as examples, then initialize the NUMB_OF_GENERATIONS according to random number between (1-50).

Population: initialize the population according POPULATION_SIZE and initial Route.

population.sortRoutesByFitness(): sort routes according to fitness.

GeneticAlgorithm(): to perform the GA for initial route.

printHeading(): print Title of columns of the heading for the structure of the output results.

driver.printPopulation(): print the routes in each population of each generation.

3.2 ANT COLONY OPTIMIZATION

This algorithm is programmed using the following five classes:

Driver class: this class contains the main method, initializes the initial route, specifies the number of ants, calls the ANT and ACO Optimization classes and print the structure of the result after getting the optimal route with the shortest path.

City class: contains the information of the cities that are, name, longitude and latitude with their required methods for setting and getting that information. Also contains a method for calculating the distance between cities using haversine formula then returning this distance.

ACO OPTIMIZATION class: has the pheromone Levels, Distances, with the methods for initialize pheromone Levels and Distances among cities.

Route class: contains the route of cities and their required methods.

ANT class: this class is the core of the program, contains the attributes, fields that controls the execution of the algorithm and performs all functions and methods of the ACO such as adjust pheromone level and determining the probability of transition from one city to another.

The following figure 3.5 illustrate the interaction among the classes. In the following we will explain each of that classes, their attributes and methods.

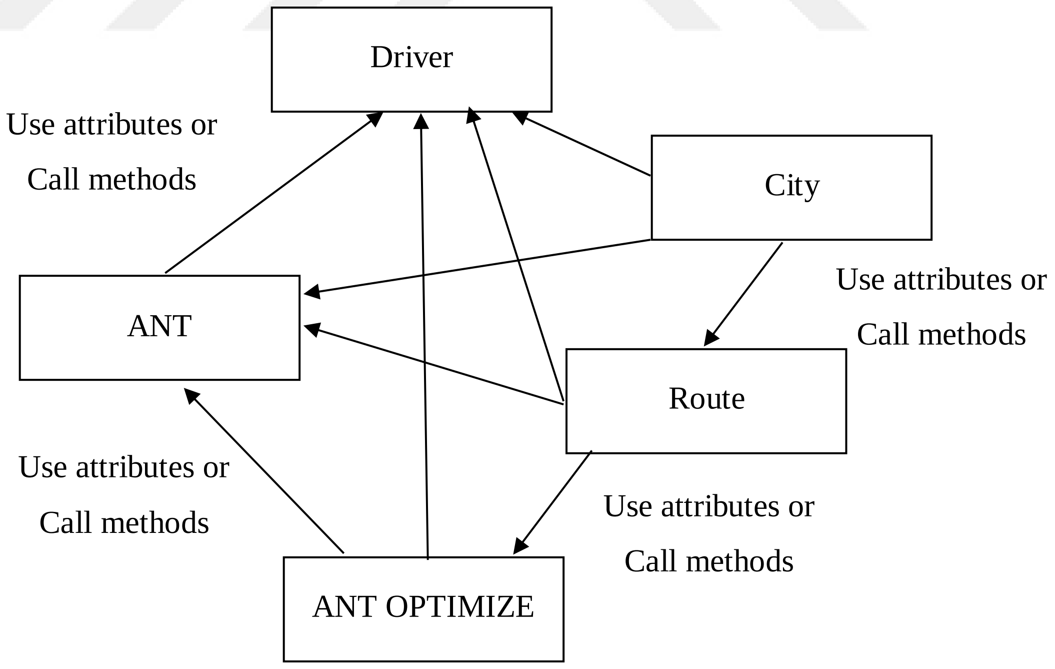


Figure 3.5: The interaction among the of ACO classes

3.2.1 City Class

This class is same to the City class of GA as illustrated previous.

3.2.2 Route Class

This class contains the following attributes:

Distance: the distance among cities in the route.

Cities: Array List of cities.

And also contains the following methods:

getCities() : to get list of cities.

getDistance() : to get the distance.

3.2.3 ACO OPTIMIZATION Class

This class contains the following attribute:

phermoneLevelsMatrix : matrix of pheromone level among the cities.

distancesMatrix: matrix of distances among the cities;

cities: cities of the initial route.

And contains the following methods

getPhermoneLevelsMatrix(): to get Pheromone Levels.

getDistancesMatrix():to get Distances

initializeDistances():to initialize Distances

initializePhermoneLevels(): to initialize Pheromone Levels.

3.2.4 ANT class

It performs all ACO operations and contains the following attributes:

Q: parameter used for adjusting the amount of pheromone deposited.

RHO: parameter used for varying the level of pheromone evaporation.

ALPHA: parameter used for controlling the importance of the pheromone trail.

BETA: parameter used for controlling the importance of the distance between source and destination.

antNumb: number of Ants.

numbOfCities: number of cities in each route.

originating Index : organize the indexes of cities.

routeCities : route of cities.

visitedCities: current visited cities.

And contains following methods:

getRoute(): get route.

Ant(AntColonyOptimization aco, int antNumb): set aco and antNumb.

Ant call(): call the ant.

adjustPhermoneLevel(int x, int y, double distance) : to adjust phermone level according to the following code:

double currentPhermoneLevel = aco.getPhermoneLevelsMatrix()[x][y].doubleValue();

double updatedPhermoneLevel = (1-RHO)*currentPhermoneLevel + Q/distance;

getTPNumerator(int x, int y) : to get the Transition Probability Numerator according to the following code:

```
    {double numerator = 0.0;
    double phermoneLevel = aco.getPhermoneLevelsMatrix()[y][x].doubleValue();
    if (phermoneLevel != 0.0)  numerator=
    Math.pow(phermoneLevel,ALPHA)*Math.pow(1/aco.getDistancesMatrix()[x][y],BETA);
    return numerator;}
```

3.2.5 Driver Class

This class contains the main method, and the following attributes:

NUMBER_OF_ANTS: set number of ants according to random number generator between (1,500).

PROCESSING_CYCLE_PROBABILITY: the probability of processing ants;

initialRoute = Array List of cities of the initial Route, we select 100 Turkish cities as examples.

Shortest Route : the shortest route found.

Active Ants : the active ants at every moment.

processAnts(): to process all active ants print the routes with shortest distance.

executorService.shutdownNow(): shut down the algorithm.

printHeading(): print Title of columns of the heading for the structure of the output results.

4 DATA AND RESULTS

After programming the code, we executed the GA and ACO in two different computers, the first commodity computer is Dell; with processor Intel® Core™ i7 CPU, 2.80 GHz and 4 GB RAM. Under Windows 10 version environment.

The second commodity computer is Toshiba; with processor Intel® Core™ i5 CPU, 2.50 GHz and 8 GB RAM. Under Windows 7 version environment.

In both cases the NetBeans IDE 8.2 software platform was used to execute the algorithms. And we used 100 of Turkish cities (name, latitude,longitude) as data that are as in table 4.1.

Table 4.1: Cities and their latitudes and Longitudes

Name	latitude	longitude	Name	latitude	longitude
Istanbul	41.00396	28.4516782	Cankir	40.60734	33.544547
Kirklareli	41.73627	27.1828603	Corum	40.5352	34.871863
Edirn	41.66887	26.5383047	Bafra	41.56421	35.870587
Tekirdag	40.95979	27.4879895	Samsun	41.29145	36.243658
Gebze	40.80449	29.3523956	Ordu	40.99345	37.766088
Yalova	40.64358	29.196147	Trabzon	40.99292	39.661219
Bursa	40.22159	28.8922011	Bayburt	40.25911	40.191333
Canakkale	40.13054	26.3570519	Rize	41.03144	40.476569
Balikesir	39.6478	27.8197115	Kars	40.59669	43.066768
Ayvalik	39.3336	26.6737007	Erzurum	39.91188	41.193476
Akhisar	38.92383	27.750777	Erzincan	39.74705	39.433867
Manisa	38.62374	27.3236154	Van	38.50282	43.218466
Izmir	38.41734	26.7995137	Batman	37.8955	41.061696
Urla	38.32663	26.7469164	Mardin	37.47719	39.998102
Cesme	38.32068	26.2268383	Elazig	38.66464	39.128111
Kusadasi	37.8557	27.2257933	Bingol	38.88318	40.483338
Selcuk	37.94868	27.3555784	Diarbakir	37.92277	40.092637
Soke	37.75261	27.3783798	Siirt	37.93037	41.898367
Aydin	37.83576	27.7773754	Patnos	39.23281	42.82282
Didim	37.3723	27.2338948	Ercis	39.02911	43.329135
Milas	37.30901	27.7450818	Hakkari	37.57427	43.699772
Bodrum	37.03579	27.3777541	Yuksekova	37.56806	44.244879
Cine	37.61254	28.0445988	Siverek	37.75368	39.291625
Mugla	37.21014	28.3467873	Sanliurfa	37.16706	38.685736
Marmaris	36.84808	28.2264257	Kilis	36.71955	37.09086
Nazilli	37.91028	28.2915433	Osmaniye	37.33439	35.739747
Denizli	37.78289	29.011605	Nigde	37.88717	34.070876

Fethiye	36.6518	29.0518288	Mus	38.75084	41.436425
Kas	36.20085	29.6322359	Tatvan	38.50373	42.221294
Antalya	37.82368	29.1268648	Sirnak	37.52206	42.438876
Isparta	37.79012	30.479748	Cizre	37.33158	39.944585
Alanya	36.54378	31.9426108	Ahlat	38.74925	42.45206
Beysehir	37.68119	31.6932248	Sakarya	40.74894	30.242206
Konya	37.87817	32.2262788	Duzce	40.8531	31.082558
KKaraman	37.17927	33.1550208	Amasra	41.74388	32.369842
Mersin	36.74263	34.3888169	Bartin	41.62977	32.303097
Tarsus	36.9229	34.8619617	Sinop	42.01411	35.059865
Adana	36.99733	35.147978	Gerze	41.80196	35.169635
Gaziantep	37.05875	37.3100958	Boyabat	41.46586	34.760633
Kahramanaras	37.55536	36.8415508	Tosya	41.01563	34.002652
Kayseri	38.72332	35.3300985	Unye	41.11399	37.167597
Aksaray	38.36072	33.8605277	Fatsa	41.04933	37.401598
Ankara	39.90304	32.4825626	Sungurlu	40.16556	34.353053
Eskisehir	39.76526	30.4047261	Yozgat	39.81521	34.777138
Beypazari	40.16237	31.8909748	Sorgun	39.8125	35.147374
Bolu	40.73572	31.5449295	Polatli	39.57855	32.10304
Zonguldak	41.459	31.7301196	Golbasi	39.78536	32.769855
Karabuk	41.2062	32.5806211	Giresun	40.90424	38.309193
Kastamonu	41.39762	33.7181286	Hopa	41.40239	41.397786
Ardahan	41.11288	42.6866254	Artvin	41.17966	41.80057

4.1 FIRST COMPUTER IMPLEMENTATION

4.1.1 Genetic Algorithm

a- With the parameters:

Population Size= 8.

MUTATION RATE = 0.25.

TOURNAMENT SELECTION SIZE = 3.

Table 4.2: Results of First Computer using GA(a)

Generation number	**Fitness value**	**Distance of Shortest Route In kilometers**	**Execution Time In seconds**
48	0.2048	48821.00	3
7	0.1900	52638.00	1
28	0.1936	51647.00	2
29	0.1939	51584.00	1

46	0.1890	52919.00	<1

b- With the parameters:

Population Size= 10.

MUTATION RATE = 0.5.

TOURNAMENT SELECTION SIZE = 3.

Table 4.3: Results of First Computer using GA(b)

Generation number	Fitness value	Distance of Shortest Route In kilometers	Execution Time In seconds
23	0.1925	51956.00	1
35	0.1888	52957.00	1
5	0.1800	55546.00	< 1
43	0.1907	52446.00	1
11	0.1859	53799.00	< 1

4.1.2 Ant Colony Optimization

a. With the parameters:

Q = 0.0005; the value is between 0 and 1

RHO = 0.2; the value is between 0 and 1

ALPHA = 0.01; the value is >=0

BETA = 9.5;

Table 4.4: Results of First Computer using ACO(a)

Count_OF_ANTS	Ant id	Distance of Shortest Route In kilometers	Execution Time In seconds
176	159	8266.34	1
429	209	8091.02	1
160	154	8301.32	1
37	3	8723.53	<1
313	247	7807.34	1

b. With the parameters:

Q = 0.0005; the value is between 0 and 1

RHO = 0.5; the value is between 0 and 1

ALPHA = 0.05; the value is >=0

BETA = 7.5;

Table 4.5: Results of First Computer using ACO(b)

Count_OF_ANTS	Ant id	Distance of Shortest Route In kilometers	Execution Time In seconds
496	125	8310.87	2
119	60	8713.85	1
348	322	8472.38	2
35	8	8418.67	<1
298	233	8287.01	1

4.2 SECOND COMPUTER IMPLEMENTATION

4.2.1 Genetic Algorithm

With the parameters:

Population Size= 10.

MUTATION RATE = 0.5.

TOURNAMENT SELECTION SIZE = 3.

Table 4.6: Results of Second Computer using GA

Generation number	Fitness value	Distance of Shortest Route In kilometers	Execution Time In seconds
31	0.2076	48159.00	< 1
3	0.1807	55330.00	< 1
45	0.2028	49303.00	1
28	0.2003	49923.00	< 1
45	0.1911	52330.00	1

4.2.2 Ant Colony Optimization

With the parameters:

Q = 0.0005; the value is between 0 and 1

RHO = 0.5; the value is between 0 and 1

ALPHA = 0.05; the value is >=0

BETA = 7.5;

Table 4.7: Results of Second Computer using ACO

Count_OF_ANTS	Ant id	Distance of Shortest Route In kilometers	Execution Time In seconds
228	13	8441.91	< 1
290	12	8393.07	< 1
236	127	8247.83	< 1
264	109	8258.60	< 1
282	213	7975.19	< 1

5 DISCUSSION AND CONCLUSION

In this thesis we examine two population and evolutionary solution approaches of TSP. We concentrated on Genetic algorithm and Ant Colony Optimization.

This comparison between two approaches depend on the results after experiments on two types of computers with their features and that experiments are performed using 100 Turkish cities.

The results of the experiments show that ACO is the better to find the optimal solution with less time for execution. At the number of ants in these experiments are between 1 and 500 randomly ants.

ACO depends on a number of parameters and very much on the count of ants used on the problem, number of cities, parameter used for adjusting the amount of pheromone deposited, parameter used for varying the level of pheromone evaporation, parameter used for controlling the significance of the pheromone trail and parameter used for controlling the significance of the distance between source and destination

While using Genetic Algorithm, the number of generations are between 1 and 50 randomly generated generations it takes longer time to produce the output.

GA depends also on a several parameters very much on the count of cities, population size, the method where the problem is encoded and on which approaches of crossover and mutation are their rates that used.

The methods that use information of heuristic give good indications for future work. GA have not found a best solution to the problem of traveling salesman than the other ideal well-known. Futures works are requiring studies towards find the relation between number of ants and cities, selection method, mutation method and mutation probability with considering other parameters.

As a future work, the ACO can be used in several computer engineering applications such as image processing, Enhance Network Routing, internet security like Spam Filtering based on ACO, solving the classification problems, Finding Shortest Path in Routing Problem in networks, also Design of distributed system with Practical Application, data mining, the TSP could also be solved using distributed systems.

All previous applications could be solved using ACO, GA and other algorithms then compare the obtained results to investigate which algorithm is the best.

REFERENCES

[1] R. Stanec, "Solving of Travelling Salesman Problem for large number of cities in environment with constraints," 2011.

[2] K. Bryant, "Genetic Algorithms and the Traveling Salesman Problem," 2000.

[3] H. Demez, "Combinatorial Optimization: Solution Methods of Traveling Salesman Problem," Gazimağusa, 2013.

[4] S. BURILLE, "Traveling Salesman Problem," 2016.

[5] T. Narwadi and Subiyanto, "An Application of Traveling Salesman Problem Using the Improved Genetic Algorithm on Android Google Maps," American Institute of Physics Publishing., 2017.

[6] B. M. Vasagam, "Ant Colony Algorithm And Genetic Algorithm For Multiple Traveling Salesman Problem," Manchester, 2012.

[7] C. Asmazoglu, "An Hybrid Approach To Solve Traveling Salesman Problem Using Genetic Algorithm," Istanbul, 2013.

[8] Ç. Cergibozan, "Meta-Heuristic Solution Approaches For Traveling Salesman And Traveling Repairman Problems," İzmir, 2013.

[9] 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*, D. Davendra, Ed., InTech, 2010.

[10] D. Johnson and L. McGeoch, "The Traveling Salesman Problem: A Case Study in Local Optimization," 1995.

[11] F. Greco, Travelling Salesman Problem, I-Tech Education and Publishing, 2008.

[12] L. Kang and W. Cao, "An Improved Genetic & Ant Colony Optimization Algorithm for Travelling Salesman Problem," *IEEE,* 2010 .

[13] F. Gharehchopogh, I. Maleki and M. Farahmandian, "New Approach for Solving Dynamic Travelling Salesman Problem with Hybrid Genetic Algorithms and Ant Colony Optimization," *International Journal of Computer Applications,* vol. 53, no. 1, September 2012.

[14] S. Sadiq, "The Traveling Salesman Problem: Optimizing Delivery Routes Using Genetic Algorithms," 2012.

[15] V. Dwivedi, T. Chauhan, S. Saxena and P. Agrawal, "Travelling Salesman Problem using Genetic Algorithm," *National Conference on Development of Reliable Information Systems,* 2012.

[16] A. Kumari and S. Singh, ”Analysis Of Travelling Salesman Problem Using Ant Colony Optimization, Genetic Algorithm And Hybrid Of Ant Colony Optimization And Genetic

Algorithm," *International Journal of Engineering Research & Technology (IJERT),* vol. 2, no. 7, July 2013.

[17] W. Hui, "Comparison of several intelligent algorithms for solving TSP problem in industrial engineering," in *The 2nd International Conference on Complexity Science & Information Engineering*, 2012.

[18] M. R. Bonyadi, M. R. Azghadi and H. Shah-Hosseini, "Population-Based Optimization Algorithms for Solving the Travelling Salesman Problem," *Travelling Salesman Problem*, I-Tech Education and Publishing, 2008, pp. 1-34.

[19] M. Affenzeller, S. Winkler, S. Wagner and A. Beham, "Genetic Algorithms and Genetic Programming Modern Concepts and Practical Applications," Taylor & Francis Group, LLC, 2009.

[20] T. Pellonpera, "Ant colony optimization and the vehicle routing problem," 2014.

[21] E. Chen and X. Liu, "Multi-Colony Ant Algorithm," *Ant Colony Optimization , Methods And Applications*, A. Ostfeld, Ed., InTech Publisher, 2011, pp. 3-12.

[22] T. Stutzle, M. Lopez-Ibanez and M. Dorigo, "A Concise Overview of Applications of Ant Colony Optimization," Brussels, 2011.

[23] T. Stautlez and M. Dorigo, "ACO algorithms for the traveling salesman problem," Evolutionary Algorithms in Engineering and Computer Science,Wiley, 1999.

[24] M. Goyal, "Modified Ant Colony Optimization Algorithm for Traveling Salesman Problem," 2012.

[25] Z. Hao, H. Huang and R. Cai, "Bio-inspired Algorithms for TSP and Generalized TSP," in *Travelling Salesman Problem*, F. Greco, Ed., I-Tech Education and Publishing, 2008, pp. 35-62.

[26] J. J. Mwemezi and Y. Huang, "Optimal Facility Location on Spherical Surfaces: Algorithm and Application," *New York Science Journal,* vol. 4, no. 7, 2011.