KWAME NKRUMAH UNIVERSITY OF SCIENCE AND TECHNOLOGY, KUMASI

COLLEGE OF SCIENCE

FACULTY OF PHYSICAL SCIENCES

DEPARTMENT OF MATHEMATICS

USING MAX-MIN ANT SYSTEM (MMAS) , TO MODEL THE INSPECTIONAL TOUR OF MAIN SALES POINTS OF GHACEM, GHANA

A CASE STUDY OF GHACEM, GHANA

BY

GYEBIL JULIUS FRANCIS

SEPTEMBER, 2012

USING MAX-MIN ANT SYSTEM (MMAS) ,

TO MODEL THE INSPECTIONAL TOUR OF MAIN SALES POINTS OF GHACEM, GHANA

A CASE STUDY OF GHACEM, GHANA

A THESIS SUBMITTED TO THE FACULTY OF DEPARTMENT OF MATHEMATICS

KWAME NKRUMAH UNIVERSITY OF SCIENCE AND TECHNOLOGY

KUMASI, IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE A WARD

OF

MASTER OF PHILOSOPHY DEGREE IN MATHEMATICS

COLLEGE OF PHYSICS SCIENCES

BY

GYEBIL JULIUS FRANCIS

SEPTEMBER, 2012.

DECLARATION

I hereby declare that this submission is my own work is towards the award of the MPIL (Mathematics) .degree and that to the best of my knowledge, it contains no material previously published by another person nor material which had been accepted for the award of any other degree of the university, except where due acknowledgement had been made in the text

Francis Julius Gyebil (3941909) … …….. ……...……..

(Student/ID) (Signature) (Date)

Certified by:

Mr. F. K. Darkwa …… ……

(Supervisor) (Signature) (Date)

Certified by:

Mr. F.K.Darkwa ……... ……...

(Head, Department of Mathematics) (Signature) (Date)

ABSTRACT

This research presents Max-Min Ants System (MMAS) under an Ant Colony Optimization (ACO) to solve a company's problem of checking the main sales points of Ghacem, Ghana starting from Tema (initial city).

This problem is formulated as a travelling salesman problem (TSP).TSP involves finding an optimal route for visiting cities and returning to point of origin. The problem formulation of the TSP in this work is based on symmetric TSP.

This work presents the solution based on Max-Min Ants System (MMAS) approach.

The MMAS algorithm proposed by Stuutzle and Hoos (2000) was coded in the matlab language in solving the problem of Ghacem, Ghana inspectional team tour of the main sales points of the company, in the country.

The result that came out the work showed that the optimal route that can be considered by the company in order to maximize profit is

Ho – Accra – Tema – Koforidua – Takoradi – Cape – Coast –Obuasi – Kumasi – Sunyani – Temale – Bolga – Wa

The total cost distance of their usual tour is $2319km$.

DEDICATION

This work is dedicated to the Almighty Lord for wonderful things He has done in my life.

ACKNOWLEDGEMENT

My most sincere gratitude goes to Mr Francis Kwaku Darkwa (Head of Department, Maths) and Dr. S.K. Amponsah for their guidance and assistance in the preparation of this work. I would like to register my sincerest gratitude to the of entire teaching staff mathematics department of KNUST.

I would like to commend the Head and staff of marketing department of Ghacem, Ghana, Tema metropolis for their willingness in releasing the required data for this work.

This work will also be uncompleted without my colleagues and family especially my wife Miss Hannah Herzuah,my mum Madam Abena Boah and my lovely Kids for their care, understanding and patience.

Thank you all.

TABLE OF CONTENT

LIST OF TABLE

LIST OF FIGURES

CHAPTER ONE

INTRODUCTION

1.1 History of *Cement*

Throughout history, cementing materials have played a vital role. They were used widely in the ancient world. The Egyptians used calcined gypsum as a cement. The Greeks and Romans used lime made by heating limestone and added sand to make mortar, with coarser stones for concrete. (Vitruvius, "The Ten Books of Architecture," Dover Publications, 1960.)

The Romans found that cement could be made which set under water and this was used for the construction of harbours. The cement was made by adding crushed volcanic ash to lime and was later called a "pozzolanic" cement, named after the village of Pozzuoli near Vesuvius. (Vitruvius, "The Ten Books of Architecture," Dover Publications, 1960.)

In places such as Britain, where volcanic ash was scarce, crushed brick or tile was used instead. The Romans were therefore probably the first to manipulate the properties of cementations materials for specific applications and situations. (Vitruvius, "The Ten Books of Architecture," Dover Publications, 1960.)

In 300BC, the Egyptians began to use mud mixed with straw to bind dried bricks. They also used gypsum mortars and mortars of lime in the building of the pyramids. The Chinese used cementitious materials in the construction of the Great Wall.

The Greeks in 800BC, used lime mortars that were much harder than later Roman mortars. This material was also in evidence in Crete and Cyprus at this time. The Babylonians and Assyrians in 300BC, used bitumen to bind stones and bricks together.

The Ancient Romans frequently used broken brick aggregate embedded in a mixture of lime putty with brick dust or volcanic ash. They built a wide variety of structures that incorporated stone and concrete, including roads, aqueducts, temples and palaces.

Between 1200BC to 1500BC, the quality of cementing materials deteriorated and even the use of concrete died out during The Middle Ages as the art of using burning lime and pozzolan (admixture) was lost, but it was later reintroduced in the 1300s. After the Romans, there was a general loss in building skills in Europe, particularly with regard to cement. Mortars hardened mainly by carbonation of lime, a slow process. The use of pozzolana was rediscovered in the late Middle Ages.

The great mediaeval cathedrals, such as Durham, Lincoln and Rochester in England and Chartres and Rheims in France, were clearly built by highly skilled masons. Despite this, it would probably be fair to say they did not have the technology to manipulate the properties of cementitious materials in the way the Romans had done a thousand years earlier.

The Renaissance and Age of Enlightenment brought new ways of thinking, which for led to the industrial revolution. The interests of industry and empire coincided, with the need to build lighthouses on exposed rocks to prevent shipping losses.

Smeaton, building the third Eddystone lighthouse (1759) off the coast of Cornwall in Southwestern England, found that a mix of lime, clay and crushed slag from iron-making produced a mortar which hardened under water. Joseph Aspdin took out a patent in 1824 for "Portland Cement," a material he produced by firing finely-ground clay and limestone until the limestone was calcined. He called it Portland Cement because the concrete made from it looked like Portland stone, a widely-used building stone in England.

A few years later, in 1845, Isaac Johnson made the first modern Portland Cement by firing a mixture of chalk and clay at much higher temperatures, similar to those used today. At these temperatures (1400C-1500C), clinkering occurs and minerals form which are very reactive and more strongly cementitious.

While Johnson used the same materials to make Portland cement as we use now, three important developments in the manufacturing process lead to modern Portland cement.

Rotary kilns heat the clinker mainly by radiative heat transfer and this is more efficient at higher temperatures, enabling higher burning temperatures to be achieved. Also, because the clinker is constantly moving within the kiln, a fairly uniform clinkering temperature is achieved in the hottest part of the kiln, the burning zone.

In 1414, the manuscripts of the Roman Pollio Vitruvius are discovered in a Swiss monastery reviving general interest in concrete.

John Smeaton (1774) found that combining quicklime with other materials created an extremely hard material that could be used to bind together other materials. He used this knowledge to build the first concrete structure since the Ancient Romans.

The Panama Canal (1914) was opened after decades of construction. It features three pairs of concrete locks with floors as thick as 20 feet, and walls as much as 60 feet thick at the bottom.

1.2 Background of the Study

Ghacem was founded by the Government of Ghana in collaboration with Norcem AS of Norway on August 30, 1967. In 1993, the Ghana Government sold 35 % of its shares to Scancem (formerly Norcem). Scancem as a result had 59.5 %, leaving Government with 40 % and 0.5 % going to a local investor. In 1997, the Ghana Government sold 5 % of its

40 % shareholding to the workers of the company. The remaining 35 % shares of the Ghana Government was sold to Scancem in 1999 and at present Scancem has 93.1 % shares in the company, workers have 5 % shares with 1.9 % owned by a local investor. In 1999, Heidelberg Cement took over Scancem, thus making it a subsidiary. Ghacem Ghana is located at both Takoradi and Tema which are coastal cities in the country. Ghacem cement has been used for construction of big and small projects, such as:

Tema Harbour, Takoradi Harbour, Akosombo Dam, Adomi Bridge, Tema Motorway, Kotoka International Airport, Aboadze Thermal Plant and West African Gas concrete piping

Construction of new stadia at Takoradi and Tamale, and the rehabilitation of Accra, Kumasi and Tema Stadia .Construction of Presidential Palace Construction of Government affordable houses for workers

The factories in Tema and Takoradi, have produced over 30 million tonnes of cement since inception in 1967. Several millions of dollars have lately been invested in expansion at both factories. These expansion works have improved the quality of Ghacem cement, reduced energy consumption at the plants, ensured efficient production and reduced environmental impact of the plants operations. Currently, Ghacem has a number of accredited distributors through-out the country. Periodic meetings are held with these distributors to reinforce partnership. An awards ceremony is held at the end of the year to honour distributors who have sold the most number of bags.

Goods from the company are transported to the regional capitals. They are then kept in the warehouses owned by key distributors in the various regional capitals. The key distributors then transport these goods to a specified area that has been allocated to them by the

company to sell. The storeowners purchase the goods from the distributors and the goods are later sold to consumers.

1.3 Statement of the Problem

The Directors of the company are tasked in every two months period to embark on tour in order to check the sales of the company's goods. They usually use the following route**;**

$$Tema \rightarrow Accra \rightarrow Cape-Coast \rightarrow Takoradi \rightarrow Obuasi \rightarrow$$
$$Kumasi \rightarrow Sunyani \rightarrow Wa \rightarrow Bolga \rightarrow Temale \rightarrow Ho \rightarrow Tema$$

Their choice of the routes for the visit was done without considering any Mathematical model. This research aims at using Min-Max Ant System (MMAS) algorithm with respect to a Symmetric Traveling Salesman Problem(STSP) model to check whether the tour is optimal.

1.4 Objective of the Study

The objectives of this research are;

1. To use a Max-Min Ant System (MMAS), which belongs to Ants Colony Optimization (ACO) to model the tour distance of the inspectional team of Ghacem, as a travelling salesman problem.

2. To provide an optimal tour distance of the inspectional team of Ghacem, as they go on to check on the sales performance of the twelve main sales points in Ghana.

1.5 Methodology

The tour of Directors of Ghacem to the major sales points to inspect sales will be modeled as Symmetric Travelling Salesman Problem. The Min-Max Ant System (MMAS) algorithm, which belongs Ants Colony Optimization (ACO) family, will be used as a method of solving the Symmetric TSP model.

In this work, a biologically inspired heuristic (ant colony) is used to solve such problem. The ant colony implemented is closely rooted at the biological and behavioral model of the real social insects. It is a non-deterministic heuristic and could be used as both constructive and iterative. The solution uses many ants of simple nature and limited memory requirements. The intelligence of this heuristic is not portrayed by individual ants, but rather is expressed by the colony as a whole. Careful presentation of the problem to the ant colony model facilitates the close biological solution derivation.

Ghana High Ways Authority will be consulted for information on the distance of the network routes from one major sales point to the other.

A matlab and Genta programs that uses the ACO algorithm will be employed to solve the TSP model. The internet, KNUST Library and Mathematics journals will be used to obtain the related literature

1.6 Justification of the Study

Ghacem Ghana is a state own company which contribute to the government revenue. It therefore prudent to minimize its operational cost so as to maximize its profit. The profit margin the company will go along way to increase the revenue of the government. In this light the fitted model will help the company to minimize its operational cost in order maximize its profit.

1.7 Structure of the thesis

Chapter one deals with the historical background of the study, the statement of the problem, the objective of the study, significant of the study and the limitation of study.

- Chapter two deals with Review of relevant literature on the topic of study
- Chapter three covers the Mathematical tools that will be used in analyze the data in order to establish the appropriate model
- Chapter four talks about data collection as well as its analyses.
- Chapter five discuss findings, conclusion, summary and recommendation of the study.

CHAPTER TWO

LITERATURE REVIEW

2.0 INTRODUCTION

This chapter will review the relevant literature and applications on Ants Colony Optimization (ACO) and Travel Salesman Problem (TSP) .In computer science and operations research, the ant colony optimization algorithm (ACO) is a probabilistic technique for solving computational problems which can be reduced to finding good paths through graphs

This algorithm is a member of ant colony algorithms family, in swarm intelligence methods, and it constitutes some metaheuristic optimizations. Initially proposed by Dorigo (1992) in his PhD thesis, the first algorithm was aiming to search for an optimal path in a graph, based on the behavior of ants seeking a path between their colony and a source of food. The original idea has since diversified to solve a wider class of numerical problems, and as a result, several problems have emerged, drawing on various aspects of the behavior of ants.

2.1 Applications on ACO

Ant colony optimization (ACO) has widely been applied to solve combinatorial optimization problems in recent years. There are few studies, however, on its convergence time, which reflects how many iteration times ACO algorithms spend in converging to the optimal solution. Based on the absorbing Markov chain model, they analyzed the ACO convergence time.

Huanq et al, (2008), presented a general result for the estimation of convergence time to reveal the relationship between convergence time and pheromone rate. This general result was then extended to a two-step analysis of the convergence time, which included the following:

(1) the iteration time that the pheromone rate spends on reaching the objective value and

(2) the convergence time that was calculated with the objective pheromone rate in expectation. Furthermore, four brief ACO algorithms were investigated by using the proposed theoretical results as case studies. Finally, the conclusions of the case studies that the pheromone rate and its deviation determine the expected convergence time were numerically verified with the experiment results of four one-ant ACO algorithms and four ten-ant ACO algorithms.

ACO has been proved to be one of the best performing algorithms for NP-hard problems as TSP. Many strategies for ACO have been studied, but little theoretical work has been done on ACO's parameters α and β, which control the relative weight of pheromone trail and heuristic value. Yang et al, (2011), described the importance and functioning of α and β, and drawn a conclusion that a fixed β may not enable ACO to use both heuristic and pheromone information for solution when α= 1. Later, following the analysis, an adaptive β strategy was designed for improvement. Finally, a new ACO called adaptive weight ant colony system (AWACS) with the adaptive β and α= 1 was introduced, and proved to be more effective and steady than traditional ACS through the experiment based on TSPLIB test.

Khader et al,(2008), proposed an ant colony optimization (ACO) algorithm together with traveling salesman problem (TSP) approach to investigate the clustering problem in

protein interaction networks (PIN). They named this combination as ACOPIN. The purpose of that work was two-fold. First, to test the efficacy of ACO in clustering PIN and second, to propose the simple generalization of the ACO algorithm that might allow its application in clustering proteins in PIN. They split that paper to three main sections. First, they described the PIN and clustering proteins in PIN. Second, They discussed the steps involved in each phase of ACO algorithm. Finally, presented some results of the investigation with the clustering patterns.

Stitizle et al, (1999), gave an overview on the available ACO algorithms for the TSP. they first introduced the TSP. they outlined how ACO algorithms can be applied to that problem and present the available ACO algorithms for the TSP. They also discussed local search for the TSP, while presented experimental results which have been obtained with MAX --MIN Ant System, one of the improved versions of Ant System. Since the first application of ACO algorithms to the TSP, which had been applied to several other combinatorial optimization problems? On many important problems ACO algorithms have proved to be among the best available algorithms. They gave a concise overview of these other applications of ACO algorithms. On The Traveling Salesman Problem

Beam-ACO algorithms are hybrid methods that combine the metaheuristic ant colony optimization with beam search. Christian Blum et al..,(2005) heavily relied on accurate and computationally inexpensive bounding information for choosing between different partial solutions during the solution construction process. In this work they presented the use of stochastic sampling as a useful alternative to bounding information in cases were computing accurate bounding information was too expensive. As a case study they chose the well-known travelling salesman problem with time windows. their results clearly

demonstrated that Beam-ACO, even when bounding information was replaced by stochastic sampling, may have important advantages over standard ACO algorithm

Dorigo et al,(2005), researched on a new metaheuristic that focused on proof-of-concept applications. It was only after experimental work had shown the practical interest of the method that researchers tried to deepen their understanding of the method's functioning not only through more and more sophisticated equations such as "how and why the method works' 'is important, because finding an answer may help in improving its applicability. Ant colony optimization, which was introduced in theearly1990s as a novel technique for solving hard combinatorial optimization problems, finds itself currently at this point of its life cycle. With this article they provided a survey on theoretical results on ant colony optimization. First, were view some convergence results. Then they discussed relations between ant colony optimization algorithms and other approximate methods for optimization. Finally, they focused on some research efforts directed at gaining a deeper understanding of the behavior of ant colony optimization algorithms. Throughout the paper they identified some open questions with a certain interest of being solved in the near future

Shan et al, (2010), addressed an integrated model that schedules multi-item replenishment with uncertain demand to determine delivery routes and truck loads, where the actual replenishment quantity only becomes known upon arrival at a demand location. The paper departed from the conventional ant colony optimization (ACO) algorithm, which minimizes total travel length, and incorporates the attraction of pheromone values that indicate the stock out costs on nodes. The contributions of the paper to the literature were made both in terms of modeling this combined multi-item inventory management with the

vehicle-routing problem and in introducing a modified ACO for the inventory routing problem.

Ant colony optimization (ACO) is a metaheuristic for solving combinatorial optimization problems that is based on the foraging behavior of biological ant colonies. Starting with the 1996 seminal paper by Dorigo, Maniezzo and Colorni, ACO techniques have been used to solve the traveling salesperson problem (TSP). Maniezzo et al, (1996), focused on a particular type of the ACO algorithm, namely, the rank-based ACO algorithm for the TSP. In particular, that paper identifies an optimal set of key parameters by statistical analysis applied to results of the rank-based ACO for the TSP. Specifically, for six frequently used TSPs available on the World Wide Web.

Ant Colony Optimization (ACO) is a metaheuristic inspired by the foraging behavior of ant colonies that had been successful in the resolution of hard combinatorial optimization problems like the Traveling Salesman Problem (TSP). Osvaldo et al, (2000), proposed the Omicron ACO (OA), a novel population-based ACO alternative originally designed as an analytical tool. To experimentally prove OA advantages, that work compared the behavior between the OA and the MMAS as a function of time in two well-known TSP problems. A simple study of the behavior of OA as a function of its parameters showed its robustness.

Yuren et al, (2006), presented the first rigorous analysis of a simple ACO algorithm called (1 + 1) MMAA (Max-Min ant algorithm) on the TSP. The expected runtime bounds for (1 + 1) MMAA on two TSP instances of complete and non-complete graphs are obtained. The influence of the parameters controlling the relative importance of pheromone trail versus visibility was also analyzed, and their choice was shown to have an impact on the expected runtime.

Amirahmad et al, (2005), estimation of sediment concentration in rivers was very important for water resource projects planning and managements. The sediment concentration was generally determined from the direct measurement of sediment concentration of river or from sediment transport equations. Direct measurement was very expensive and cannot be conducted for all river gauge stations. However, sediment transport equations do not agree with each other and require many detailed data on the flow and sediment characteristics. Various models have been developed so far to identify the relation between discharge and sediment load. Most of the models based on regression method have some restrictive assumptions. Ant colony optimization (ACO) is now being used more frequently to solve optimization problems other than those for which they were originally developed. The main purpose of that paper was literature review of Ant Colony Optimization for suspended sediment estimation.

Ant colony optimization (ACO) has been proved to be one of the best performing algorithms for NP-hard problems as TSP. The volatility rate of pheromone trail is one of the main parameters in ACO algorithms. It is usually set experimentally in the literatures for the application of ACO. Yong et al, (2006), presented a paper that proposed an adaptive strategy for the volatility rate of pheromone trail according to the quality of the solutions found by artificial ants. The strategy was combined with the setting of other parameters to form a new ACO algorithm. Finally, the experimental results of computing traveling salesman problems indicated that the proposed algorithm was more effective than other ant methods.

White et al, (2008), proposed the addition of Genetic Algorithms to Ant Colony System (ACS) applied to improve performance. Two modifications were proposed and tested. The first algorithm was a hybrid between ACS-TSP and a Genetic Algorithm that encodes experimental variables in ants. The algorithm does not yield improved results but offered

concepts that could be used to improve the ACO algorithm. The second algorithm used a Genetic Algorithm to evolve experimental variable values used in ACSTSP. They found that the performance of ACS-TSP could be improved by using the suggested values.

ACO is a metaheuristic inspired in the behavior of natural ant colonies to solve combinatorial optimization problems, based on simple agents that work cooperatively communicating by artificial pheromone trails.

Eduardo et al, (2009), used a model to solve the municipal waste collection problem by containers was presented, which applies a concept of partial collection sequences that must be joined to minimize the total collection distance. The problem to join the partial collection sequences was represented as a TSP, which is solved by an ACO algorithm. Based on the literature, algorithm parameters are experimentally calibrated and range of variations that represents good average solutions are recommended. The model was applied to a waste collection sector of the San Pedro de la Paz commune in Chile, obtaining recollection routes with less total distance with respect to the actual route utilized and to the solution obtained by a previously developed approach.

Beam-ACO algorithms are hybrid methods that combine the metaheuristic ant colony optimization with beam search.

Lopez et al, (2005), heavily relied on accurate and computationally inexpensive bounding information for choosing between different partial solutions during the solution construction process. In that work they presented the use of stochastic sampling as a useful alternative to bounding information in cases were computing accurate bounding information is too expensive. As a case study they choose the well-known travelling salesman problem with time windows. Their results clearly demonstrated that Beam-ACO,

even when bounding information was replaced by stochastic sampling, may have important advantages over standard ACO algorithms.

Yunmming et al, (2010), combined with the idea of the Bean Optimization algorithm (BOA), the ant colony optimization (ACO) algorithm was presented to solve the well known traveling salesman problem (TSP). The core of that algorithm was using BOA to optimize the control parameters of ACO which consist of heuristic factor, pheromone evaporation factor and random selection threshold, and applying ant colony system to solve two typical TSP. The new algorithm effectively overcame the influence of control parameters of ACO and decreased the numbers of experiments. The novel hybrid algorithm ACOBOA found the balance between exploiting the optimal solution and enlarging the search space. The results of the experiments showed that ACOBOA had better optimization performance and efficiency than the general ant colony optimization algorithm and genetic algorithm. The new algorithm could also be generalized to solve other NP problems.

Recently, researchers have been dealing with the relation of ACO algorithms to the other methods for learning and optimization. One example is the work presented in

Birattari, et al, (2002), presented work that relates ACO to the fields of optimal control and reinforcement learning. A more prominent example is the work that aim at finding similarities between ACO algorithms and other probalistic learning algorithms such as stochastic gradient ascent (SGA), and the cross-entropy (CE) method.

Meuleau et al, (2002), shown that the pheromone update as outline in the proof-of-concept application to the TSP (Dorigo *et al.* 1991, 1996) is very similar to a stochastic gradient ascent in the space of pheromone values.

Blum, (2004), proposed the first implementation of SGA-based ACO algorithms where it was shown that SGA-based pheromone updates avoid certain types of search bias.

Zlochin et al, (2004), proposed a unifying framework from so-called model-based search (MBS) algorithms. An MBS algorithm is characterized by the used of a (parameterized) probabilistic. The class of MBS algorithm can be divided into two subclasses with respect to the way the probabilistic model is used. The algorithm in the first subclass use a given probabilistic model without changing the model structure at run-time, whereas the algorithms of the second subclass use and change the probabilistic model in alternating phases.

ACO algorithms are examples of algorithms from the first subclass. While convergence proofs can provide insight into the working of an algorithm, they usually not very useful to the practitioner that wants to implement efficient algorithms. This is because, generally, either infinite time or infinite spaces are required for a stochastic optimization algorithm to converge to an optimal solution (or to the optimal solution value). The existing convergence proofs for particular ACO algorithms are no exception.

Blum et al, (2005, 2004), adopted the term deception for the field of ant colony optimization , similarly to what had previously been done in evolutionary computation. It was shown that ant colony optimization algorithms in general suffer from first order deception in the same way as Gas suffer from deception. they further introduce the concept of second order deception, which is caused by a bias that leads to decreasing algorithm performance over time.

Recently Montgomery et al,(2004), recently made an attempt to extend the work by Blum and Sampels, (2002), to assignment problems, and to attribute search bias to different algorithmic components.

Merkle et al, (2002), were the first to study the behavior of a simple ACO algorithm by analyzing the dynamics of its model, which is obtained by applying the expected pheromone update. Their work deals with the application of ACO to idealized permutation problems. When applied to constrained problems such as permutation problems, the solution construction process of ACO algorithms consist of a sequence of random decisions in which later decisions depend on earlier ones. Therefore, the later decisions of the construction process are inherently biased by the earlier ones. The work of Merkle and Middendorf shows that this leads to a bias which they call selection bias. Furthermore, the competition between the ants was identified as the main driving force of the algorithm.

ACO is a metaheuristic inspired in the behavior of natural ant colonies to solve combinatorial optimization problems, based on simple agents that work cooperatively communicating by artificial pheromone trails.

Nelson et al, (2009), generated a model to solve the municipal waste collection problem by containers was presented, which applied a concept of partial collection sequences that must be joined to minimize the total collection distance. The problem to join the partial collection sequences is represented as a TSP, which was solved by an ACO algorithm. Based on the literature, algorithm parameters were experimentally calibrated and range of variations that represents good average solutions are recommended. The model was applied to a waste collection sector of the San Pedro de la Paz commune in Chile, obtaining recollection routes with less total distance with respect to the actual route utilized and to the solution obtained by a previously developed approach.

Ant colony optimization (ACO) has been proved to be one of the best performing algorithms for NP-hard problems as TSP. The volatility rate of pheromone trail is one of the main parameters in ACO algorithms. It is usually set experimentally in the literatures

for the application of ACO. Shanker et al, (2008), presented a paper that proposed an adaptive strategy for the volatility rate of pheromone trail according to the quality of the solutions found by artificial ants. The strategy was combined with the setting of other parameters to form a new ACO algorithm. Finally, the experimental results of computing traveling salesman problems indicated that the proposed algorithm was more effective than other ant methods

The behavior of a (1+1)-ES process on Rudolph's binary long k paths was investigated extensively in the asymptotic framework with respect to string length l. First, the case of $k=l^{\alpha}$ was addressed.

Kallel et al, (2002), proved that the long k path was a long path for the (1+1)-ES in the sense that the process follows the entire path with no shortcuts, resulting in an exponential expected convergence time. For $\alpha<1/2$, the expected convergence time is also exponential, but some shortcuts occur in the meantime that speed up the process. Next, in the case of constant k, the statistical distribution of convergence time was calculated, and the influence of population size was investigated for different $(\mu+\lambda)$-ES. The histogram of the first hitting time of the solution shows an anomalous peak closed to zero, which corresponds to an exceptional set of events that speed up the expected convergence time with a factor of l^2. A direct consequence of this exceptional set is that performing independent (1+1)-ES processes proves to be more advantageous than any population-based $(\mu+\lambda)$-ES

Chun et al, (2004), put forward a brief runtime analysis of an evolutionary programming (EP) which is one of the most important continuous optimization evolutionary algorithms. A theoretical framework of runtime analysis was proposed by modeling EP as an absorbing Markov process. The framework was used to study the runtime of a classical EP

algorithm named as EP with Cauchy mutation (FEP). It was proved that the runtime of FEP could be less than a polynomial of n if the Lebesgue measure of optimal solution set was more than an exponential form of 2. Moreover, the runtime analysis result could be used to explain the performance of EP based on Cauchy mutation.

Ant colony optimization (ACO) is one of the most famous bio-inspired algorithms. Its theoretical research contains convergence proof and runtime analysis. The convergence of ACO has been proved since several years ago, but there are less results of runtime analysis of ACO algorithm except for some special and simple cases. Yang et al, (2010), presented a paper that proposed a theoretical framework of a class of ACO algorithms. The ACO algorithm was modeled as an absorbing Markov chain. Afterward its convergence could be analyzed based on the model, and the runtime of ACO algorithm was evaluated with the convergence time which reflects how many iteration times ACO algorithms spend in converging to the optimal solution. Moreover, the runtime analysis result was advanced as an estimation method, which was used to study a binary ACO algorithm as a case study.

2.2 Travelling Salesman Problem

The Travelling Salesman problem is one of the most popular problems from the NP set; it is also one of the hardest too.

The solution to this problem enjoys wide applicability in a variety of practical fields. Thus, it highly raises the need for an efficient solution for this NP Hard problem.

According to Schrijver, (2005), the general form of the TSP appears to have been first studied during the 1930s in Vienna and at Harvard, notably by Karl Menger, who defined the problem, considered the obvious brute-force algorithm, and observed the non-optimality of the nearest neighbour heuristic; i.e. the rule that one should first go from the starting point to the closest point, then to the point closest to this, etc., in general does not

yield the shortest route.

Hassler, (2006), introduced the name travelling salesman problem soon after (Schrijver, 2005). In the 1950s and 1960s, the travelling Salesman problem became increasingly popular in scientific circles in Europe and the USA.

Dantzig et al, (1954), at the RAND Corporation in Santa Monica, made notable contribution who expressed the problem as an integer linear program and developed the cutting plane method for its solution. A problem instance with 49 cities was then solved to optimality with these new methods by constructing a tour and proving that no other tour could be shorter.

In the years that followed, the problem was studied by many researchers from mathematics, computer science, chemistry, physics, and other sciences.

Karp et al, (1970), explored new approaches to the TSP in which 1-trees, which are a variant of spanning trees, play essential role. They observed that a tour is a 1-tree in which each vertex has degree 2.

Karp et al, (1971), described Dynamic programming algorithm for solving small instances and for finding approximate solutions to larger instances. The exact algorithm was used to solve 13-city instance on an IBM 7090 computer. The approximation algorithm (also programmed for the IBM 7090) found the optimal solution to the 42-city Dantzig-Fulkerson-Johnson example on two out of five trials, and was also tested on a new 48-city instance.

Karp, (1972), showed that the Hamiltonian cycle problem was NP-complete, which implies the NP-hardness of TSP. This supplied a mathematical explanation for the apparent computational difficulty of finding optimal tours.

The Christofides, (1976), algorithm, one of the first approximation algorithms, combines the minimum spanning tree with a solution of another problem, minimum-weight perfect matching. This gives a TSP tour which is at most 1.5 times the optimal. The Christofides algorithm was, in part, responsible for drawing attention to approximation algorithms as a practical approach to intractable problems. The term "algorithm" was not commonly extended to approximation algorithms until later; the Christofides algorithm was initially referred to as the Christofides heuristic (Schrijver, 2005).

Land, (1979), described a cutting-plane algorithm for the TSP. She solved linear-programming relaxations in integer arithmetic, thus avoiding rounding errors in the computations. The separation algorithms included a shrinking heuristic for identifying subtour inequalities and a heuristic for identifying blossom inequalities. If no subtours or blossoms were found, a Gomory-cut was added to the relaxation. She used column generation to handle the great number of edges present in larger instances.

Padberg et al, (1980), described a cutting-plane algorithm which makes use of new separation routines for comb inequalities. Like Land (1979), the linear programming computations were carried out using integer arithmetic to "avoid any problems connected with round-off errors." In their computational study, the authors solved 54 out of total of 74 instances by linear programming relaxation. For the 318-city example of Lin and Kernighan (1973), the bound obtained via the relaxation was within a factor of 0.96 of the best tour that was found.

Padberg et al, (1980), solved the 318-city instance, described in Lin and Kernighan (1973), which remained until 1987 as the largest TSP solved. In their work, the authors made further rounds of cutting planes and IBM MPSX-MIP/370 interger-programming solver

was used to carry out a branch and bound search on the final linear programming relaxation.

Grötschel et al, (1984), managed to exactly solve instances with up to 2392 cities, using cutting planes and branch-and-bound.

Hopfield, (1986), explored an innovative method to solve combinatorial optimization problems and implemented a neural network of the Hopfield Model (HM) into an electric circuit that produces approximate solutions to the TSP quite efficiently.

Walshaw, (2002), derived, and implemented a multilevel approach to the travelling salesman problem. The resulting algorithm progressively coarsens the problem, initialises a tour, and then employs either the Lin-Kernighan (LK) or the Chained Lin-Kernighan (CLK) algorithm to refine the solution on each of the coarsened problems in reverse order. In experiments on a well-established test suite of 80 problem instances.

Walshaw, (2002), found multilevel configurations that either improved the tour quality by over 25% as compared to the standard CLK algorithm using the same amount of execution time, or that achieved approximately the same tour quality over seven times more rapidly.

According to Walshaw, (2002), the multilevel variants seemed to optimise far better the more clustered instances with which the LK and CLK algorithms have the most difficulties.

Applegate et al, (2003, 2006), developed the program *Concorde*, which has been used in many recent record solutions.

Reinel, (1991), published the travelling Salesman Problem Library (TSPLIB), a collection of benchmark instances of varying difficulty, which has been used by many research groups for comparing results.

Applegate et al, (1994), solved a travelling salesman problem which models the production of printed circuit boards having 7397 holes (cities)..

Applegate, et al, (2008), described a computer code and data that together certify the optimality of a solution to the 85,900-city travelling salesman problem, pla85900, the largest instance in the TSPLIB collection of challenge problems, currently the largest solved TSPLIB instance.

Cook et al, (2005), computed an optimal tour through a 33,810-city instance given by a microchip layout problem

Haist et al, (2007), introduced an optical method based on white light interferometry in order to solve the travelling salesman problem. To the authors' knowledge, it was the first time that a method for the reduction of non–polynomial time to quadratic time had been proposed. The authors showed that this achievement was limited by the number of available photons for solving the problem. It turned out that the number of photons was proportional to N^N for a travelling salesman problem with N cities and that for large numbers of cities the method in practice therefore was limited by the signal–to–noise ratio.

Kohonen self organizing map is an important artificial neural network technique that uses competitive, unsupervised learning to produce a low-dimensional discretized representation of the input space of the training samples which preserves the topological properties of the input space. The fuzzy set theory introduces the concept of membership

function to the learning process of Self Organizing Map which helps to handle the inherent vagueness involved in most of the real life problems.

Chaudhuri et al, (2009), used fuzzy self organizing map with one dimensional neighbourhood to find an optimal solution for the symmetrical Traveling Salesman Problem. The solution generated by the Fuzzy Self Organizing Map algorithm was improved by the 2-opt algorithm. Finally, the Fuzzy Self Organizing Map algorithm was compared with Lin-Kerninghan Algorithm and Evolutionary Algorithm with Enhanced Edge Recombination operator and self- proposed by Yang et al, (2008), adapting mutation rate.

Yang et al, (2008), proposed, Shuffled frog-leaping algorithm (SFLA) is a new mimetic meta-heuristic algorithm with efficient mathematical function and global search capability, When applying the shuffled frog-leaping algorithm in TSP, the authors built memeplex and submemeplex and the evolution of the algorithm, especially the local exploration in submemeplex was carefully adapted based on the prototype SFLA.

According to Yang et al, (2008), experimental results show that the shuffled frog leaping algorithm is efficient for small-scale TSP. Particularly, for TSP with 51 cities; the algorithm manages to find six tours which are shorter than the optimal tour provided by TSPLIB.

Amponsah et al, (2010), used an algorithm due to Dharwalker, (2008), to find Hamilton circuits in solving a ten-city TSP in Ghana.

Ameyaw, (2010), also used simulated annealing to solve an eleven-city TSP of eleven sales points of Unilever in Ghana. In both instances, the problems were modelled as symmetric TSP.

2.2.1 Variants of Travelling Salesman Problem

Many forms of the TSP have been proposed by different authors in the literature. In the next seven sections some of the various forms of the TSP will be reviewed.

The Selective Travelling Salesman Problem

The Selective Travelling Salesman Problem is defined on a graph in which profits are associated with vertices and costs are associated with edges. Some vertices are compulsory. The aim is to construct a tour of maximal profit including all compulsory vertices and whose cost does not exceed a preset constant.

Gendreau et al, (1998), developed several classes of valid inequalities for the symmetric Selective Travelling Salesman Problem and used them in a branch-and-cut algorithm. Depending on problem parameters, the proposed algorithm can solve instances involving up to 300 vertices.

Non-Euclidean Visual Travelling Salesman Problem

In the task of finding the shortest tour of *n* cities given intercity costs, usually, the intercity costs are 2-Dimentional Euclidean distances. In the presence of obstacles or in the case of 3-Dimentional surfaces, the intercity distances are in general not Euclidean. The TSP with obstacles and on 3-Dimentional surfaces approximates our everyday visual navigation and this leads to the Non-Euclidean visual travelling salesman problem

Catrambone, et al, (2008),.proposed three questions that are related to the mechanisms involved in solving TSP:

i. How do subjects find the intercity distances?
ii. How do they determine clusters of cities?

iii. How do they produce the TSP tour?

In their model, on Non-Euclidean visual travelling salesman problem, they found the non-Euclidean distances (geodesics); the geodesic distances were then used as intercity costs in a graph pyramid. The original TSP problem was represented by a sequence of problems involving clusters of cities. The hierarchical clustering was performed by using a Boruvka's minimum spanning tree. Close to the top of the pyramid, the original TSP problem was represented at a very coarse level and involved very small number of "cities". This coarse representation was solved optimally. Expanding this coarse tour in a top-down manner led to a solution of the original TSP. The new model had an adaptive spatial structure and it simulated visual acuity and visual attention. The model solved the TSP problem sequentially, by moving its attention from city to city.

The Generalized Travelling Salesman Problem (GTSP)

The generalized travelling salesman problem (GTSP) is an extension of the TSP. In GTSP, a partition of cities into groups is given and a minimum length tour that includes exactly one city from each group is to be found.

The Probabilistic Travelling Salesman Problem

Campbell et al, (2007), defined the Probabilistic Travelling Salesman Problem (PTSP) as a generalization of the well known Travelling Salesman Problem (TSP). In contrast to the TSP, each city in the PTSP has to be visited only with a certain probability, thus allowing more realistic models and scenarios. The goal here is to find a so called a-priori tour that visits all cities exactly once, minimizing the expected cost over all possible a-posteriori tours, where cities which do not require a visit are just skipped without changing the order of the a-priori tour.

As a generalization of the TSP, the PTSP is NP-hard and therefore algorithms computing near optimal solutions in a reasonable amount of time are of great interest.

Formally, the PTSP can be defined over a complete undirected edge- and node-weighted graph $G = \{V, c, p\}$. $V = \{1, 2, \ldots, n\}$ is the set of nodes which represent the customers,

$p: V \rightarrow Q[0,1]$ is the probability function that assigns to each node the probability that the node requires a visit and $c: V \times V \rightarrow \mathrm{IR}^{+}$ is the symmetric cost function that represents the non-negative travel costs between any two nodes

The Probabilistic Travelling Salesman Problem with Deadlines (PTSPD)

Campbell et al, (2007), found out that time-constrained deliveries were one of the fastest growing segments of the delivery business, and yet there was surprisingly little literature that addressed time constraints in the context of stochastic customer presence.

The authors began to fill that void by introducing the probabilistic travelling salesman problem with deadlines (PTSPD) which is an extension of the well-known probabilistic travelling salesman problem (PTSP) in which, in addition to stochastic presence, customers must also be visited before a known deadline.

Campbell et al, (2007), presented Two recourse models and a chance constrained model for the PTSPD.In their work, special cases were discussed for each model, and computational experiments were used to illustrate under what conditions stochastic and deterministic models lead to different solutions.

The General Routing Problem (GRP)

The general routing problem (GRP) is the problem of finding a minimum length tour, visiting a number of specified vertices and edges in an undirected graph.

Muyldermans et al, (2005), described how the well-known 2-opt and 3-opt local search procedures for node routing problems could be adapted to solve arc and general routing problems successfully. Two forms of the 2-opt and 3-opt approaches were applied to the GRP. The first version was similar to the conventional approach for the travelling salesman problem; the second version included a dynamic programming procedure and explored a larger neighbourhood at the expense of higher running times.

The Asymmetric Travelling Salesman Problem with Time Windows

In the asymmetric travelling salesman problem (ATSP) the cost or distance from city i to city j is not the same as the cost or distance from city j to city i. The asymmetric travelling salesman problem with time windows (ATSP-TW), an extension of ATSP, is a basic model for scheduling and routing applications.

Ascheuer et al, (2000), presented a formulation of the problem involving only 0/1 variables associated with the arcs of the underlying digraph. This had the advantage of avoiding additional variables as well as the associated (typically very ineffective) linking constraints. In the formulation, time-window restrictions were modelled using "infeasible path elimination" constraints. The authors presented the basic form of these constraints along with some possible strengthening. Several other classes of valid inequalities derived from related asymmetric travelling salesman problems were also described.

2.2.2 Applications of Travelling Salesman Problem

There are several practical applications of the TSP. Discussion that covers some possible

applications, not complete though, is given; we start with applications that can be modelled directly as one of the variants given in the previous section.

Drilling of printed circuit boards

A direct application of the TSP is the drilling problem whose solution plays an important role in economical manufacturing of printed circuit boards (PCBs).

Grötschel et al, (1991), gave a computational study in an industry application of a large electronics company. To connect a conductor on one layer with a conductor on another layer, or to position (in a later stage of the PCB production) the pins of integrated circuits, holes have to be drilled through the board. The holes may be of different diameters. To drill two holes of different diameters consecutively, the head of the machine has to move to a tool box and change the drilling equipment. This is quite time consuming. Thus it is clear at the outset that one has to choose some diameter, drill all holes of the same diameter, change the drill, drill the holes of the next diameter, etc where the "cities" are the initial position and the set of all holes that can be drilled with one and the same drill. The "distance" between two cities is given by the time it takes to move the drilling head from one position to the other. The aim here is to minimize the travel time for the head of the machine.

X-Ray crystallography

An important application of the TSP occurs in the analysis of the structure of crystals (Bland and Shallcross, 1987; Dreissig and Uebach, 1990). Here an X-ray diffractometer is used to obtain information about the structure of crystalline material. To this end a detector measures the intensity of X-ray reflections of the crystal from various positions. Whereas the measurement itself can be accomplished quite fast, there is a considerable overhead in

positioning time since up to hundreds of thousands positions have to be realized for some experiments and the positioning involves moving four motors (Dreissig and Uebach, 1990). The time needed to move from one position to the other can be computed very accurately. According to the authors, the result of the experiment does not depend on the sequence in which the measurements at the various positions are taken. However, the total time needed for the experiment depends on the sequence. Therefore, the problem consists of finding a sequence that minimizes the total positioning time. This leads to a travelling salesman problem.

Overhauling gas turbine engines

This application was reported by Plante, (1987), and occurs when gas turbine engines of aircraft have to be overhauled. To guarantee a uniform gas flow through the turbines there are so-called nozzle-guide vane assemblies located at each turbine stage. Such an assembly basically consists of a number of nozzle guide vanes affixed about its circumference. All these vanes have individual characteristics and the correct placement of the vanes can result in substantial benefits (reducing vibration, increasing uniformity of flow, reducing fuel consumption). The problem of placing the vanes in the best possible way can be modelled as a TSP with a special objective function.

The order-picking problem in warehouses

This problem is associated with material handling in a warehouse (Ratliff and Rosethal, 1981)). Assume that at a warehouse an order arrives for a certain subset of the items stored in the warehouse. Some vehicle has to collect all items of this order to ship them to the customer. The relation to the TSP is immediately seen. The storage locations of the items correspond to the nodes of the graph. The distance between two nodes is given by the time

needed to move the vehicle from one location to the other. The problem of finding a shortest route for the vehicle with minimum pickup time can now be solved as a TSP.

Computer wiring

A special case of connecting components on a computer board is reported in Lenstra and Kan, (1974), Modules are located on a computer board and a given subset of pins has to be connected. In contrast to the usual case where a Steiner tree connection is desired, here the requirement is that no more than two wires are attached to each pin. This leads to the problem of finding a shortest Hamiltonian path with unspecified starting and terminating points. A similar situation occurs for the so-called test bus wiring. To test the manufactured

Board, one has to realize a connection which enters the board at some specified point, runs through all the modules, and terminates at some specified point. For each module we also have a specified entering and leaving point for this test wiring. This problem also amounts to solving a Hamiltonian path problem with the difference that the distances are not symmetric and that starting and terminating point are specified.

Vehicle routing

Suppose that in a city *n* mail boxes have to be emptied every day within a certain period of time, say, 1 hour. The problem is to find the minimum number of trucks to do this and the shortest time to do the collections using this number of trucks. As another example, suppose that *n* customers require certain amounts of some commodities and a supplier has to satisfy all demands with a fleet of trucks. The problem is to find an assignment of customers to the trucks and a delivery schedule for each truck so that the capacity of each truck is not exceeded and the total travel distance is minimized. Several variations of these

two problems, where time and capacity constraints are combined, are common in many real-world applications.

Lenstra et al, (1974), Applied the method for the TSP to find good feasible solutions.

Control of robot motions

In order to manufacture some workpiece a robot has to perform a sequence of operations on it (drilling of holes of different diameters, cutting of slots, etc.). The task is to determine a sequence of the necessary operations that leads to the shortest overall processing time. A difficulty in this application arises because there are precedence constraints that have to be observed. This can be modelled as a problem of finding the shortest Hamiltonian path (where distances correspond to times needed for positioning and possible tool changes) that satisfies certain precedence relations between the operations.

CHAPTER THREE

METHODOLOGY

3.1 Introduction

Several exact and heuristic algorithms exist in the literature that can solve instances of the TSP will be discussed in this chapter However, in this study, we used a heuristic method namely Man-Min Ant System (MMAS) algorithm.

This algorithm is a member of ant colony algorithms family, in swarm intelligence methods, and it constitutes some metaheuristic optimizations. Initially proposed by Dorigo, (1992), in his PhD thesis, the first algorithm aimed to search for an optimal path in a graph, based on the behavior of ants seeking a path between their colony and a source of food. The original idea has since diversified to solve a wider class of numerical problems, and as a result, several problems have emerged, drawing on various aspects of the behavior of ants.

3.2 Formulation of TSP Model

The first step to solving instances of large TSPs must be to find a good mathematical formulation of the problem. The mathematical structure is represented by a graph where each city is denoted by a point (or node) and lines [(called arcs or edges)] are drawn connecting every two nodes. Associated with every edge is a distance (or cost). When the salesman can get from every city to every other city directly, then the graph is said to be *complete*. A round-trip of the cities corresponds to some subset of the edges, and is called a Hamilton tour or a Hamiltonian cycle in graph theory. The length of a tour is the sum of the lengths of the lines in the round-trip.

3.2.1 Formulation of the Asymmetric TSP

The problem can be defined as follows: Let *G = (V,E)* be a complete directed graph with vertices *W*, $|W|=n$, where n is the number of cities, and edges *E* with edge length d_{ij} for *(i,j)*. We focus on the asymmetric TSP case in which $d_{ij} \neq d_{ji}$, for all *(i,j)*. *where* $c_{ij} = d_{ij}$

The asymmetric TSP can be formulated as an integer linear programme in the following way. Let $n \times n$ distance matrix $C = c_{ij}$ be given. We the introduce a binary variable x_{ij} by

$$x_{ij} = \begin{cases} 1 \; if \;\; j \;\; is \;\; visited \;\; immediately \;\; after \;\; i \\ 0 \qquad\qquad otherwise \end{cases}$$

$$P1: Minimize = \sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij} \qquad (3.1)$$

Subject to

$$\sum_{i=1}^{n} x_{ij} = 1 \quad for \;\; all \;\; j = 1,2,...,n \qquad (3.2)$$

$$\sum_{j=1}^{n} x_{ij} = 1 \quad for \;\; all \;\; i = 1,2,...,n \qquad (3.3)$$

$$\sum_{i \in W} \sum_{j \in W} x_{ij} \leq |W| - 1 \qquad for \;\; all \;\; W \subset \{1,2,...,n\}, \; 1 \leq |W| \leq n-1 \qquad (3.4)$$

$$x_{ij} \in \{0,1\} \quad for \;\; all \;\; i, j = 1,2,...,n \qquad (3.5)$$

Equation (3.3) : Objective function, which minimize the total distance traveled

Equation (3.2) and (3.3): Constraints (3.2) and (3.3) define a regular assignment problem, where (3.2) ensures that each city is entered from only one other city, while (3.3) ensures that each city is only departed to on other city.

Constraint(3.4): is sub tour elimination constraint which ensures that every tour has at most |W|-1 edges with both endpoints in the set W

Constraint(3.5):is the integrality constraint that ensures that the decision variable is either 0 or 1 However, the difficulty of solving TSP is that subtour constraints will grow exponentially as the number of city grows large, so it is not possible to generate or store these constraints. Many applications in real world do not demand optimal solutions.

3.2.2 The symmetric TSP Model

The problem can be defined as follows: Let $G = (V,E)$ be a complete undirected graph with vertices W, $|W|=n$, where n is the number of cities, and edges E with edge length dij for (i,j). We focus on the symmetric TSP case in which $d_{ij} = d_{ji}$, for all (i,j).

The formulation of symmetric TSP model in the Binary Linear programme form is;

$$P2: Minimize \sum_{i=1}^{n-1} \sum_{j=i+1}^{n} c_{ij} x_{ij} \qquad (3.6)$$

Subject to

$$\sum_{j \neq i} x_{ij} = 2 \quad for\ all\ i = 1,2,...,n \qquad (3.7)$$

$$\sum_{i \in W} \sum_{j \in W} x_{ij} \leq |W| - 1 \quad for\ all\ W \subset \{1,2,...,n\},\ \ 2 \leq |W| \leq n-1 \qquad (3.8)$$

$$x_{ij} \in \{0,1\} \quad for\ all\ i, j = 1,2,...,n \qquad (3.9)$$

c_{ij} =the distance from city i to city j

x_{ij} = the decision variable that is either 0 or 1

W= arbitrary subset of {1,2,…,n}

n=the total number of cities

Equation (3.6) is the objective function, which minimizes the total distance to be traveled.

Equation (3.7): Each edge is incident with exactly two cities

Equation (3.8):The subtour elimination constraint

Equation (3.9): The integrality constraint

3.3 Methods of Solving TSP

The two most popular exact methods that are used to solving IP problems are discussed. They are cutting plane method and Branch and Bounds.

3.3.1 Cutting Plane Method

Cutting plane methods are exact algorithms for integer programming problems. They have proven to be very useful computationally in the last few years, especially when combined with a branch and bound algorithm in a branch and cut framework. These methods work by solving a sequence of linear programming relaxations of the integer programming problem.

The relaxations are gradually improved to give better approximations to the integer programming problem, at least in the neighborhood of the optimal solution. For hard

instances that cannot be solved to optimality, cutting plane algorithms can produce approximations to the optimal solution in moderate computation times, with guarantees on the distance to optimality.

Cutting plane algorithms have been used to solve many different integer programming problems, including the traveling salesman problem (Gr¨otschel and Holland ,1991, Padberg and Rinaldi, 1991, Applegate et al, 1994);the linear ordering problem (Gr¨otschel et al, 1984, Mitchell and Borchers, 1996,Mitchell and Borchers, 1997); maximum cut problems in (Barahona, et al, 1988, De Simone et al, 1995 and Mitchell, 1997) and packing problems (Gr¨otschel, and Weismantel, (1996) , Nemhauser and Sigismondi, (1992).

J¨unger et al. (1995) contains a survey of applications of cutting plane methods, as well as a guide to the successful implementation of a cutting plane algorithm. Nemhauser and Wolse (1992) provides an excellent and detailed description of cutting plane algorithms as well as other aspects of integer programming..

3.3.1.1 Using the fractional algorithm of cutting plane

In this algorithm all coefficients including the right hand side need to be integer. This condition is necessary as all variables (original, slack and artificial) are supposed to be integer. The elements of A and b need not be integer although this can be transformed into integers as shown below.

In case a constraint with fractional coefficient exist then both sides of the inequality (equality) are multiplied by the least common multiple of the denominator (LCMD).

For instance $3x_1 + \frac{1}{5}x_2 \le \frac{2}{3}$ *becomes* $45x_1 + 3x_2 \le 10$

3.3.1.2 Procedure for cutting plane algorithm

Solve the integer programming problem as a Linear Programming Problem.

If the optimal solution is integer stop else go to step c.

Introduce secondary constraints (cut) that will push the solution towards integrality (Return to a).

We show how to constraint the secondary constraints in the following sections

3.1.3 The construction of the secondary constraints:

Given the integer problem

Minimize $Z=C^{T}X$

Subject to $AX \leq b$

$X \geq 0$, integer

X=Vector of decision variable.

C^{T} =Vector coefficients

A=the given matrix

B=vector coefficient

The optimal tableau of the Linear programming Problem is given in table 3 below:

For simplicity of notation let us have $X = \left(X_B, X_{NB} \right)$

$X_B = (X_1 ... X_M)$ and $X_{NB} = (W_1 ... W_N)$

Table 3.1**: Showing the variables to be considered in the Cutting Plane Method.**

	Z	$X_1 \dots X_i \dots X_M$	$W_1 \dots W_J \dots W_N$	Solution
Z	1	0 0 0	$C_1 \dots C_j \dots C_N$	β_0
X_1	0	1 0 0	$\alpha_{11} \dots \alpha_{1j} \dots \alpha_{1N}$	β_1
X_i	0	0 1 0	$\alpha_{i1} \dots \alpha_{ij} \dots \alpha_{iN}$	β_i
X_M	0	0 0 1	$\alpha_{M1} \dots \alpha_{Mj} \dots \alpha_{MN}$	β_M

Consider the *ith* equation where X_i was required to be integer but found not integer.

$$X_i = \beta_i - \sum_{j=1}^{N} \left(\alpha_{ij}.W_j\right) \text{ and } \beta_i \text{ non integer : } i = 1,\dots,M \quad (3.10)$$

Any real number can be written as the sum of two parts , integer part and the fractional part.

Let $\beta_i = [\beta_i] + f_i$ and $\alpha_{ij} = [\alpha_{ij}] + g_{ij}$ (3.11)

then

$$x_i = [\beta_i] + f_i - \sum_{j=1}^{N} \left([\alpha_{ij}] + g_{ij}\right) w_j \text{ and}$$

$$f_i - \sum_{j=1}^{N}\left(g_{ij}W_j\right) = X_i - [\beta_i] + \sum_{j=1}^{N}\left(\left[\alpha_{ij}\right]W_j\right) \qquad (3.12)$$

Where $[a] \le a$ and ([a] is integer part of a); $0 < f_i < 1$;

$0 \le g_{ij} < 1$($[\beta] \le \beta$ *and* ($[\beta]$ is the integer part of β)

(note that $f_i > 0$ as X_i is presently not integer)

Since all $x_i\,(i = 1,...,M)$ *and all* $W_j\,(j = 1,...,N)$ must be integer, the right-hand side is consequently integer and therefore the left-hand side is also integer thus from table 3.1

$$f_i - \sum_{j=1}^{N}(g_{ij}W_j) \in \square \text{ (Integer)} \qquad (3.13)$$

$g_{ij} \ge 0$ *and* $W_{ij} \ge 0$ then from equation (3.3) with $X_i > [\beta_i]$ $f_i - \sum_{j=1}^{N}\left(g_{ij}W_j\right) \ge 0$

Therefore

$$f_i \ge f_i - \sum_{j=1}^{N}(g_{ij}W_j) \text{ for all } i = 1,...,N \qquad (3.14)$$

Since $0 < f_i < 1$ we have $f_i - \sum_{j=1}^{N}(g_{ij}W_j) < 1$ and using (3.13) we obtain

$$f_i - \sum_{j=1}^{N}(g_{ij}W_j) \le 0 \qquad (3.15)$$

Constraint (13.15) is the cut and can be expressed as a secondary constraints by adding slack variable:

This gives

$$f_i - \sum_{j=1}^{N}(g_{ij}W_j) + S_i = 0 \rightarrow S_i = \sum_{j=1}^{N}(g_{ij}W_j) - f_i \qquad (3.16)$$

for all $i = 1,...,M$, Where $S_i \ge 0$ (integer slack variable).

3.3.1.4 Choice of the cut

Suppose two rows in table 3.1 gives non-integer solutions in X_i and X_k then there will be two cuts based on X_i and X_k having the following conditions:

i. $f_i \le \sum_{j=1}^{N} g_{ij} W_j$

ii. $f_k \le \sum_{j=1}^{N} g_{kj} W_j$

Cut (i) is stronger than cut (k) if

(iii) $f_i \ge f_k$ and $g_{ij} \le g_{kj}$ for all j

With the strict inequality happening at least once.

In other words a cut is deeper in the X_i direction as f_i increases and g_{ij} decreases.

The condition (iii) is difficult to implement computationally and therefore empirical rule that take into account the above definition have been developed.

(a) $f_r \Big/ \sum_{j=1}^{N} g_{rk} = Max \left\{ f_i \Big/ \sum_{i=1}^{N} g_{ik} ; i = 1,...,M; X_i \text{ for a specified } k \right\}$

(b) $f_r \Big/ \sum_{j=1}^{N} g_{rj} = Max \left\{ f_i \Big/ \sum_{i=1}^{N} g_{ij} ; i = 1,...,M; X_i \neq \square \text{ but } X_i \text{ required to be integer} \right\}$

(c) $f_r / g_{ik} = Max \left\{ f_i / g_{ik} ; i = 1,...,M, \text{ for a specified } k \right\}$

Criterion (b) is more efficient as this represents the definition given by (iii) better.

3.3.1.5 Prototype Example

Maximize $Z = 7x_1 + 9x_2$

Subject to

$-x_1 + 3x_2 \leq 6$

$7x_1 + x_2 \leq 35$

$x_1 \geq 0$, $x_2 \geq 0$, integer

Solution:

Maximize $Z = 7x_1 + 9x_2 + 0s_1 + 0s_2$

Subject to

$-x_1 + 3x_2 + 1s_1 = 6$

$7x_1 + x_2 + 1s_2 = 35$

Table 3.2: **Final Tableau for first iteration**

	C_j	**7**	**9**	**0**	**0**	
C_B	Basic variable	x_1	x_2	s_1	s_2	**Solution**
9	x_2	**0**	1	$\frac{7}{22}$	$\frac{1}{22}$	$\frac{7}{2}$
7	x_1	**1**	**0**	$\frac{-1}{22}$	$\frac{1}{22}$	$\frac{9}{2}$
	Z_j	**7**	**9**	**0**	**0**	**63**
	$C_j - Z_j$	**0**	**0**	$\frac{-28}{11}$	$\frac{-15}{11}$	

Let $s_1 = x_3$, $s_2 = x_4$, $Z = x_5$

From the tableau the optimal solution becomes Z=63, where $x_2 = \frac{7}{2}$ and $x_1 = \frac{9}{2}$

Since x_2 and x_1 are not integers, we apply the concepts of cutting plane techniques.

$$x_2 + \frac{7}{22} x_3 + \frac{1}{22} x_4 = \frac{7}{2} \quad \text{(i)}$$

$$x_1 + 0 x_2 - \frac{1}{22} x_3 + \frac{3}{22} x_4 = \frac{9}{2} \quad \text{(ii)}$$

Choice of cut

Taking equations (i) and (ii)

$$x_2 + \left(0 + \frac{7}{22}\right) x_3 + \left(0 + \frac{1}{2}\right) x_4 = \left(3 + \frac{1}{2}\right) \quad \textbf{(i)a}$$

$$x_1 - \left(1 - \frac{21}{22}\right) x_3 + \left(0 + \frac{3}{22}\right) x_4 = \left(4 + \frac{1}{2}\right) \quad \textbf{(ii)b}$$

$$\frac{1}{2} - \frac{7}{22} x_3 - \frac{1}{2} x_4 = x_2 + 0x_3 + 0x_4 - \quad \text{(iii)}$$

$$\frac{1}{2} - \frac{21}{22} x_3 - \frac{3}{22} x_4 = x_1 - x_3 + 0x_4 \quad \text{(iv)}$$

$$f_2 = \frac{1}{2} \ , \ g_{23} = \frac{7}{22} \ , \ g_{24} = \frac{1}{2}$$

$$f_3 = \frac{1}{2} \ , \ g_{33} = \frac{21}{22} \ , \ g_{34} = \frac{3}{22}$$

Using

$$f_r \Big/ \sum_{j=1}^{N} g_{rj} = Max\left\{ f_i \Big/ \sum_{j=1}^{N} g_{ij} ; i = 1,...,M; X_i \notin \square \text{ but } X_i \text{ required to be integer} \right\}$$

when $i = 2$, $j = 3,4$

$$f_2 = \frac{1}{2} \text{ , } g_{23} = \frac{7}{22} \text{ and } g_{24} = \frac{1}{22}$$

$$\sum_{j=3}^{4} g_{ij} = \frac{7}{22} + \frac{1}{22} = \frac{8}{22}$$

when $i = 3$, $j = 3,4$

$$g_{33} = \frac{21}{22} \text{ and } g_{34} = \frac{3}{22}$$

$$\sum_{j=3}^{4} g_{ij} = \frac{21}{22} + \frac{3}{22} = \frac{24}{22}$$

$$\max\left[\left(1/2 \Big/ 8/22 \right), \left(1/2 \Big/ 24/22 \right) \right]$$

$$\max\left[\frac{22}{16} \text{ , } \frac{22}{48} \right] = \frac{22}{16}$$

Hence (ia) would be considered to be part of the new constraints.

Thus $\frac{1}{2} - \frac{7}{22} x_3 - \frac{1}{22} x_4 \leq 0$

and $\frac{1}{2} - \frac{7}{22} x_3 - \frac{1}{22} x_4 + S_3 = 0$

$$-\frac{7}{22} x_3 - \frac{1}{22} x_4 + s_3 = -\frac{1}{2}$$

The system of equations becomes;

$$Z = 7x_1 + 9x_2 + 0x_3 + 0x_4 + 0x_5$$

Subject to;

$$x_2 + \frac{7}{22} x_3 + \frac{1}{22} x_4 = \frac{7}{2}$$

$$x_1 + 0 x_2 - \frac{1}{22} x_3 + \frac{3}{22} x_4 = \frac{9}{2}$$

$$-\frac{7}{22} x_3 - \frac{1}{22} x_4 + x_5 = -\frac{1}{2}$$

$$S_3 = X_5$$

Table 3.3**: Final Tableau for the second iteration**

	c_j	**7**	**9**	**0**	**0**	**0**	
c_B	Basic variable	x_1	x_2	x_3	x_4	s_3	**Solution**
9	x_2	**0**	**1**	**0**	**0**	**1**	**3**
7	x_1	**1**	**0**	**0**	$\frac{1}{7}$	$\frac{-1}{7}$	$\frac{32}{7}$
0	x_3	**0**	**0**	**1**	$\frac{1}{7}$	$\frac{-22}{7}$	$\frac{11}{7}$
	z_j	**7**	**9**	**0**	**1**	**0**	**59**
	$c_j - z_j$	**0**	**0**	**0**	**-1**	**-8**	

$z_{\max} = 59$, $x_2 = 3$, $x_1 = \frac{32}{7}$ and $x_3 = \frac{11}{7}$

Since x_1 and x_3 are not integers we apply the cutting plane techniques.

Using the fractional algorithm;

$$x_1 + \frac{1}{7}x_4 - \frac{1}{7}x_5 = \frac{32}{7} \qquad \textbf{(i)*}$$

$$\rightarrow x_1 + \left(0 + \frac{1}{7}\right)x_4 + \left(-1 + \frac{6}{7}\right)x_5 = 4 + \frac{4}{7}$$

$$\rightarrow x_1 + 0x_4 - 1x_5 - 4 = \frac{4}{7} - \frac{1}{7}x_4 + \frac{6}{7}x_5 \qquad \text{(i)*}$$

$$x_3 + \frac{1}{7}x_4 - \frac{22}{7}x_5 = \frac{11}{7} \qquad \text{(ii)*}$$

$$\rightarrow x_3 + \left(0 + \frac{1}{7}\right)x_4 + \left(-4 + \frac{6}{7}\right)x_5 = 1 + \frac{4}{7}$$

$$\rightarrow x_3 + 0x_4 - 4x_5 - 1 = \frac{4}{7} - \left(\frac{1}{7}x_4 + \frac{6}{7}x_5\right) \qquad \text{(ii)*}$$

Choice of Cut

From (1a)* $f_2 = \frac{4}{7}$, $g_{24} = \frac{1}{7}$, $g_{25} = \frac{6}{7}$

From (2a)* $f_3 = \frac{4}{7}$, $g_{34} = \frac{1}{7}$, $g_{35} = \frac{6}{7}$

Using

$$f_r \Big/ \sum_{j=1}^{N} g_{rj} = Max\left\{ f_i \Big/ \sum_{j=1}^{N} g_{ij} ; i = 1,...,M; X_i \notin \square \text{ but } X_i \text{ required to be integer} \right\}$$

When $i = 2$, $j = 4,5$

$$f_2 = \frac{4}{7} \quad g_{24} = \frac{1}{7}, \; g_{25} = \frac{6}{7}$$

Therefore

$$\sum_{j=4}^{5} g_{ij} = \frac{1}{7} + \frac{6}{7} = 1$$

When $i = 3$, $j = 4,5$

$$f_3 = \frac{4}{7} \quad g_{34} = \frac{1}{7} \quad , \quad g_{35} = \frac{6}{7}$$

$$\sum_{j=4}^{5} g_{ij} = \frac{1}{7} + \frac{6}{7} = 1$$

$$\max\left[\frac{f_2}{\sum_{j=4}^{5} g_{ij}}, \frac{f_3}{\sum_{j=4}^{5} g_{ij}}\right] = \max\left[\frac{4}{7}, \frac{4}{7}\right] = \frac{4}{7}$$

Tie will be broken arbitrary by choosing equation (ii)* as the new constraints to be added.

Where $s_3 = x_5$.

The system of equations becomes;

$$Z = 7x_1 + 9x_2 + 0x_3 + 0x_4 + 0x_5 + 0s_4$$

Subject to

$$x_2 = 3$$

$$x_1 + \frac{1}{7}x_4 - \frac{1}{7}x_5 = \frac{32}{7}$$

$$x_3 + \frac{1}{7}x_4 - \frac{22}{7}x_5 = \frac{11}{7}$$

$$-\frac{1}{7}x_4 - \frac{6}{7}x_5 + s_4 = -\frac{4}{7}$$

Table 3.4**: Final tableau for the last iteration**

	c_j	7	9	0	0	0	0	
c_B	Basic variable	x_1	x_2	x_3	x_4	x_5	s_4	Solution
9	x_2	0	1	0	0	0	0	3
7	x_1	1	0	0	0	-1	1	4
0	x_3	0	0	1	0	-4	1	1
0	x_4	0	0	0	1	6	-7	4
	z_j	7	9	0	0	-7	7	55
	$c_j - z_j$	0	0	0	0	7	-7	

Now the $Z_{\max} = 55$, $x_2 = 3$, $x_1 = 4$, $x_3 = 1$ and $x_4 = 4$

Since all the variables are integers, we stop here.

3.4 Branch and Bound Method

Branch-and-bound algorithms was developed by Eastman, (1958), Little et al, (1963), and Shapiro, (1966),. Additionally, Hatfield and Pierce, (1966), used branch-and-bound algorithms to solve a job sequencing problem closely related to the traveling salesman problem, but further constrained because of job deadlines to be met. The work of Little, et al, (1963), is a tour-building algorithm, while the work of Eastman and Shapiro are examples of subtour elimination algorithms. The authors are not aware of a branch-and-bound algorithm based upon tour-to-tour improvement, although presumably one could be

constructed. A rather complete survey of branch-and-bound methods has been given by Lawler and Wood, (1966).

The problem can be modeled as

Maximize $Z = \sum_{j=1}^{n} C_j X_j$

Subject to $\sum_{j=1}^{n} a_i x_j \leq b_i \ , \ 1 \leq i \leq m$

and X_j is an integer , $1 \leq j \leq n$

To solve such integer programming problem the following steps should be considered.

3.4.1 Branch and Bound Algorithm .

The steps below are used in the branch and bound algorithm

STEP 1: Relaxed problem P_0 with respect to integrality condition is called the relaxed problem. This leads to the following linear programming problem which is called the relaxed problem,

P_0 : Maximize $Z = \sum_{j=1}^{n} C_j X_j$

Subject to:

$$\sum_{j=1}^{n} a_{ij} x_j \leq b_i \ , \ 1 \leq i \leq m$$

$X_j \geq 0$ X_j is integer

We solve the relaxation problem P_0 by the simplex method.

STEP 2: If in the solution of P_0 every variable that is supposed to be an integer is indeed an integer , then we are done .If this is not the case, then there exist at least one variable which is required to be an integer and whose value in our solution is not an integer. Pick any such variable and branch on it as follows

STEP 3: Suppose that at least one variable X_j where $(1 \leq j \leq n)$

has a non-integer value $X_j = K_j$ when it should been an integer.

We define $[K_j]$ to be the lower integer part of K_j so that $[K_j] < X_j < [K_j]+1$.Since X_j must be an integer, it follows that it must obey exactly one of the following constraints.

(i) $X_j \leq [K_j]$ or (ii) $X_j \geq [K_j]+1$

STEP 4: To branch on X_j means solving the following problem.

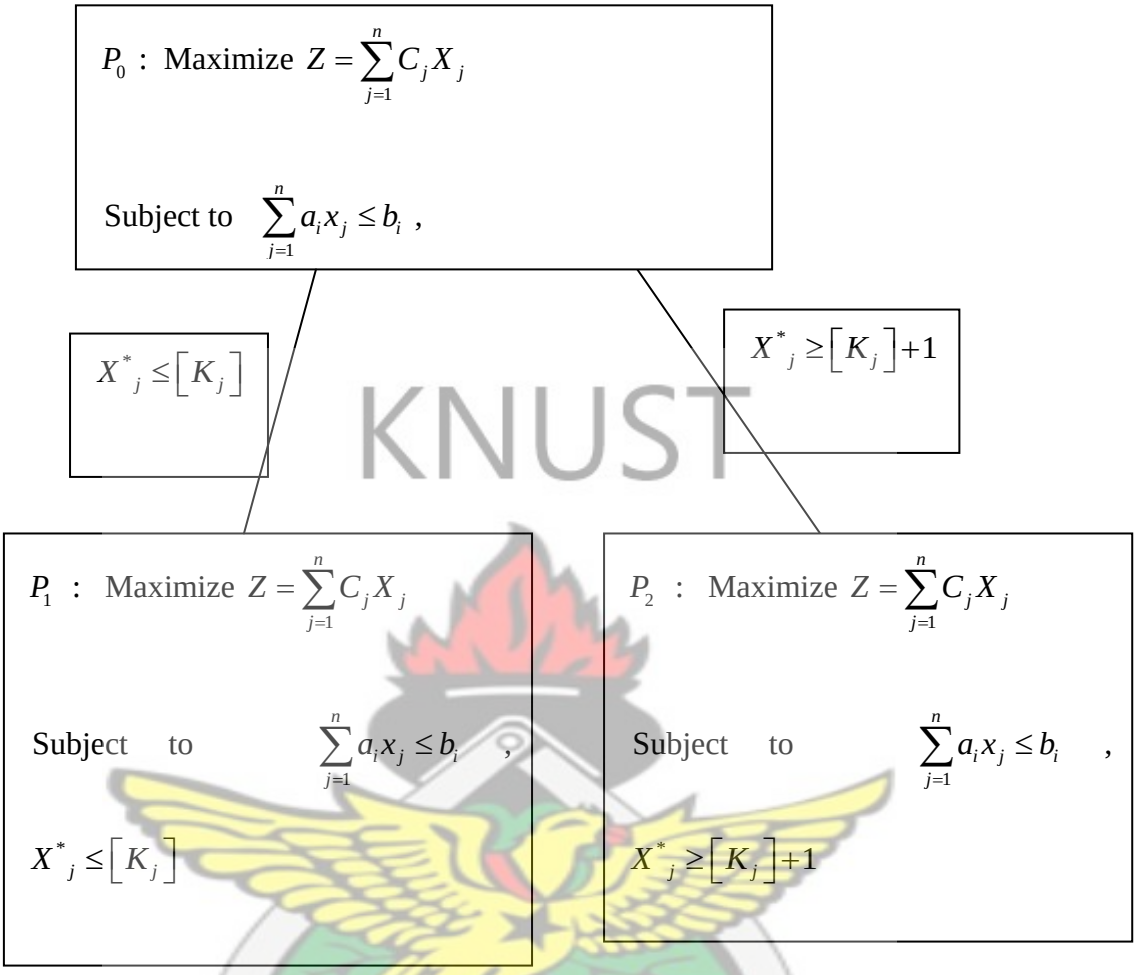
Form two sub problems P_1 *and* P_2 to replace the current problem P_0 adding a lower bound constraint to one and an upper-bound constraint to the other for the variable selected above in step 3.It then partition the current subset of solutions into two new subsets of solutions.

We now solve the LP of P_1 such that

(iv) $P_1 : P_0 + \{x_j \leq [k_j]\}$ and the problem

$P_2 : P_0 + \{x_j \geq [k_j]+1\}$

The branching is illustrated in the tree of figure 3.1 below



STEP 5: Let maximum objective function value of the two sub problems be Z= M_i in $P_i, i = 1,2$ Since the feasible region of problem P_i is a subset of the feasible region of P_0, it follows that $M_i \le M_0, i = 1,2.$ Hence , M_0 is an upper bound to the optimal solutions 0f the problems P_1 and P_2.Test the problems P_1 and P_2 , for feasibility, discard any infeasible problem and solve the feasible ones. If, in the solution of P_1 or P_2 all the variables in the original problem that satisfy integrality conditions are integers, we are done and our optimal value is either M_1 or M_2, depending on which is the larger one.

STEP 6: If , in the solution of a problem P_i, $i=1,2.$, all the variables that should be integers are indeed integers , we say that the problem P_i is fathomed. If either P_1 or P_2 is not fathomed, we branch on it, choosing the problem with the higher bound. We continue in this manner until some problems have been fathomed and all the unfathomed problems have bounds lower than those in the fathomed problems. We then select the solution of the fathomed problems with the highest objective function value as our solution.

3.4.1.1 Prototype Example

Using the steps given ,we consider the following problem:

Minimize Z= $X_{1`} + 4X_2$ (i)

Subject to ;

$2X_{1`} + X_2 \leq 8$ (ii)

$X_{1`} + X_2 \geq 6$ (iii)

$X_{1`} \geq 0$ , $X_{2`} \geq 0$ (iv)

X_1, X_2 are integers (v)

STEP 1 : The algorithm begins by solving (i) to (iv) as an LP problem. This has the following optimal solutions for P_0 , $X_1^* = \frac{10}{3}$, $X^*_2 = \frac{4}{3}$ and $Z^* = \frac{26}{3} = 8\frac{2}{3}$ is the lower bound on the set of all feasible solutions .If this first solution had satisfied (v), it would have been optimal for the integer programming problem and the algorithm would have been terminated. However, as this is not the case , we shall proceed.

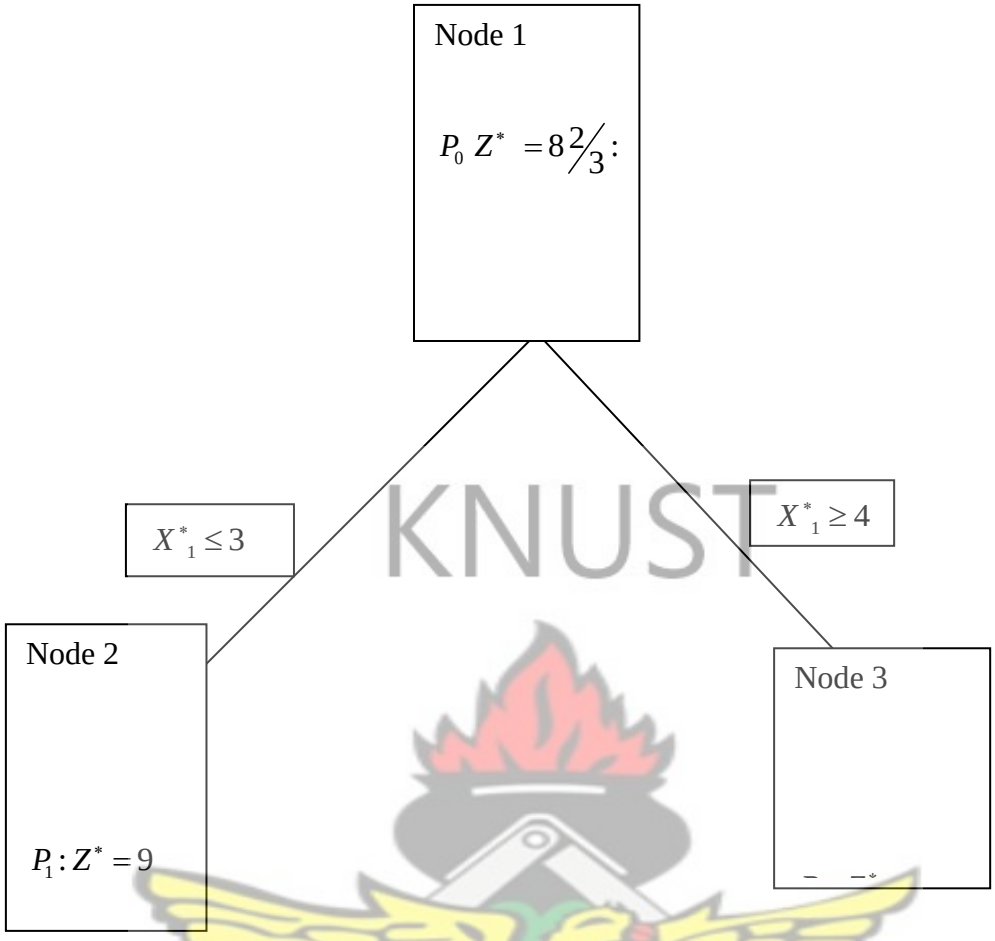


Figure 3.1: Solution tree for Dakin's algorithm.

STEP (2):Since X_1^* *and* X_2^* both have non-integer values in step 1.Arbitrarily select one to branch on .The set of feasible solutions is partitioned into two subsets. One set contains all the feasible solutions with the addition of constraint $X^*_1 \le [10/3] = 3$ and the other contains the set of feasible solutions with the addition of constraint $X^*_1 \ge [10/3] + 1 = 4$.This reduces the region of feasible solutions of the LP problem, but leaves the region of feasible solutions of the integer Linear programming problem unchanged, since there are no integer solution between $3=[10/3]$ and $4=[10/3]+1$ .Therefore, iteration 1 begins by partitioning the entire set of solutions into the two subsets below.

(1) Solution in which $X_1 \le 3$

(2) Solutions in which $X_1 \geq 4$

We now create two LP problems, P_1 and P_2:

P_1 : Minimize $Z = X_1 + 4X_2$

Subject to $2X_1 + X_2 \leq 8$

$X_1 + 2X_2 \geq 6$

$X_1 \leq 3$

$X_1 \geq 0,\ X_2 \geq 0\ \ x_1, x_2$ integers

P_2 : Minimize $Z = X_1 + 4X_2$

Subject to $2X_1 + X_2 \leq 8$

$X_1 + 2X_2 \geq 6$

$X_1 \geq 4\ \ X_1 \geq 0,\ X_2 \geq 0\ \ x_1, x_2$ integers

For problem P_1,we solve the corresponding LP problem. The solution is

$X^*_1 = 3,\ X^*_2 = 3/2$ and $Z^* = 9$.

The solution is still non-feasible for the original problem, but $Z^* = 9$ is the lower bound on the set of all feasible solutions with $X^*_1 = 3$,as shown in figure(2).Also, the problem corresponding to P_2 is solved by using the corresponding LP problem. There is no feasible solution for problem P_2.

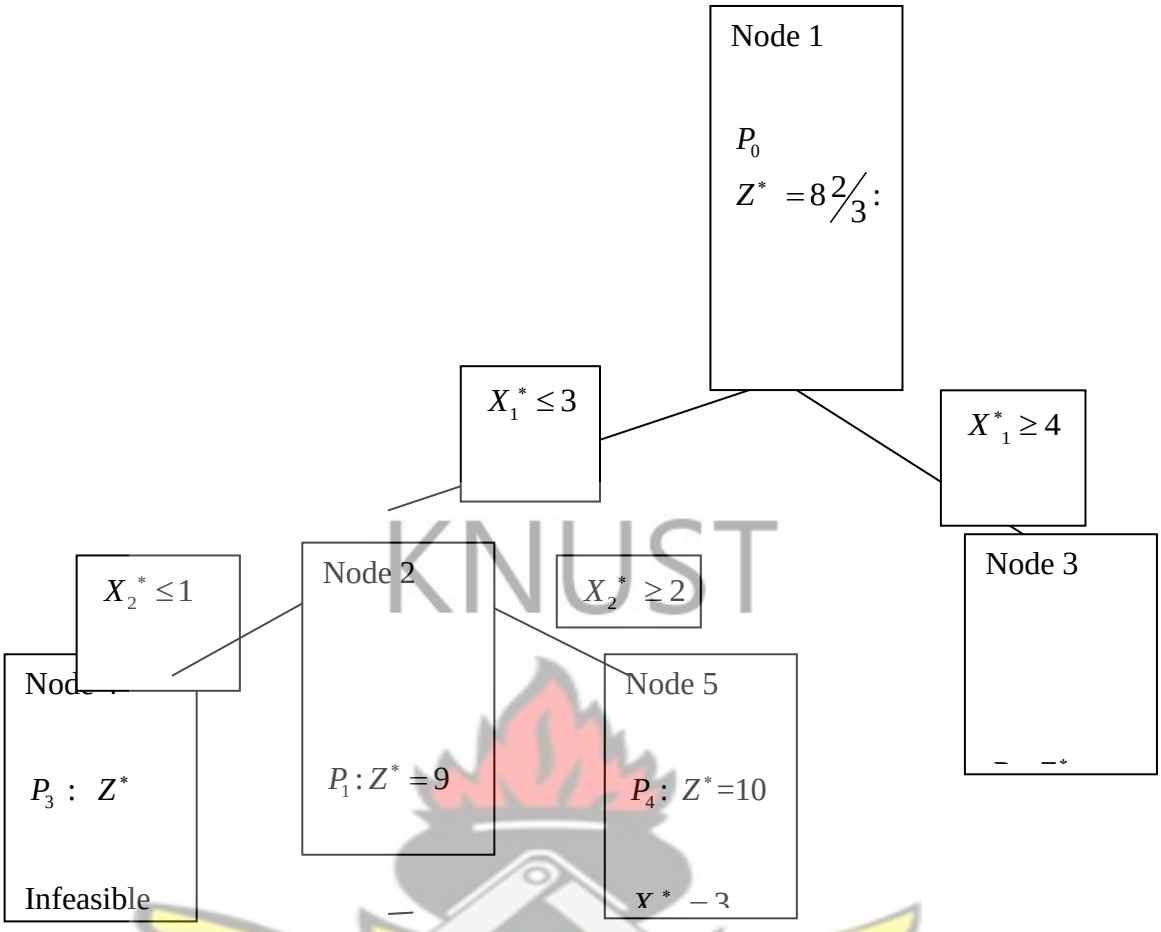


Figure 3.2 : The complete solution tree for Daskin's algorithm.

STEP 3 : Node 2 of problem P_1 is the only one for branching. The solutions with $X_2^* \ge \frac{3}{2}$ from problem P_1 is partitioned into two subset, one with $X_2^* \le [\frac{3}{2}] = 1$ and the other with $X_2^* \ge 2$.

These subsets correspond to Nodes 4 and 5 respectively of problems P_3 and P_4 as shown in figure2.4.2.Therefore the sub problem to solve at node 4 of problem P_3 is

P_3 : Minimize $Z = X_1 + 4X_2$

Subject to

$$2X_1 + X_2 \leq 8$$

$$X_1 + 2X_2 \geq 6$$

$$X_1 \leq 3$$

$$X_2 \leq 1$$

$$X_1 \geq 0,\ X_2 \geq 0$$

By solving the LP of problem P_3 at Node 4, we find the solution to be infeasible. The sub problem P_4 at Node 5 is

Minimize $Z = X_1 + 4X_2$

Subject to

$$2X_1 + X_2 \leq 8$$

$$X_1 + 2X_2 \geq 6$$

$$X_2 \geq 2$$

$X_1 \geq 0,\ X_2 \geq 0, X_1$ and X_2 are integers.

By solving the LP of problem P_4 at Node 5, the solution is

$$X_1^* = 3,\ X_2^* = 2, \text{ and } Z^* = 10.$$

$X_1^* = 3$ and $X_2^* = 2$ is the optimal solution of the original problem with an optimal value of the objective function being $Z^* = 10$.

3.6 Heuristic Approaches to Solving TSP

According to Hillier and Lieberman, (2005), heuristic method is a procedure that is likely to discover a very good feasible solution, but not necessarily an optimal solution, for the specified problem being considered. No guarantee can be given about the quality of the solution obtained, but a well-designed heuristic method usually can provide a solution that is at least nearly optimal. The procedure often is a full-fledged iterative algorithm, where each iteration involves conducting a search for a new solution that might be better than the best solution found previously. When the algorithm is terminated after a reasonable time, the solution it provides is the best one that was found during any iteration. Heuristic methods are often based on relatively simple common sense ideas on how to search for a good solution. These ideas need to be carefully tailored to fit the specific problem of interest. Thus, heuristic methods tend to be *ad hoc* in nature). TSP heuristics can be partitioned into two classes*:* construction heuristics and improvement heuristics**.**

Construction Heuristics Construction heuristics build a tour from scratch and stop when one is

produced. The simplest and most obvious construction heuristic is *nearest neighbor (NN)*: the tour starts at any vertex *x* of the complete directed or undirected graph; we repeat the following loop until all vertices have been included in the tour: add to the tour a vertex (among vertices not yet in the tour) closest to the vertex last added to the tour.

Improvement Heuristics: Improvement heuristics start from a tour normally obtained using a construction heuristic and iteratively improves it by changing some parts of it at each iteration. Improvement heuristics are typically much faster than the exact algorithms, yet often produce solutions very close to the optimal one.

It appears that currently the best improvement heuristics are based on local search, on genetic algorithm approach, or on a mixture of the two, which is often called memetic algorithms.

The most developed TSP improvement algorithms are local search algorithms that use edge exchange, in which a tour is improved by replacing *k* its edges with *k* edges not in the solution. For STSP, the *2-opt* algorithm starts from an initial tour *T* and tries to improve *T* by replacing two of its non-adjacent edges with two other edges to form another tour. Once an improvement is obtained, it becomes the new *T*. The procedure is repeated as long as an improvement is possible (or a time limit is exceeded). For $k \geq 3$, the *k-opt* algorithm is the same as 2-opt except that *k* edges are replaced at each iteration (Rego and Glover 2002).

The best local search algorithms use a variable *k*-Opt search called the Lin-Kernighan local search, where at each iteration the actual value of *k* varies depending on which value of *k* gives the best improvement (Rego and Glover ,2002).

3.6.1 Sub-Tour Reversal Algorithm

The sub-Tour Reversal Algorithm is a local improvement procedure of a TSP that adjusts the cities visited in the current trial solution by a subsequence of the cities of the current solution and simply reversing the order in which that subsequence of cities is visited.(The subsequence being reversed can consist of as few as two cities, but also can have more). It improves upon the current trial solution to obtain a *local optimum*.

The Sub-tour reversal algorithm is implemented below

1. Initialization: Start with any feasible tour as the initial trial solution

2. Iteration: For the current trial solution, consider all possible ways of performing a sub tour reversal (exclude the reversal of the entire tour). Select the one that

provides the largest decrease in the distance to the new trial solution. (Ties may be broken arbitrary)

3. Stopping Rule: Stop when no sub-tour reversal will improve the current trial solution. Accept this solution as the final solution. Otherwise go to step

The example shows the network of a traveling salesman problem with seven cities. City 1 is the salesman's city.

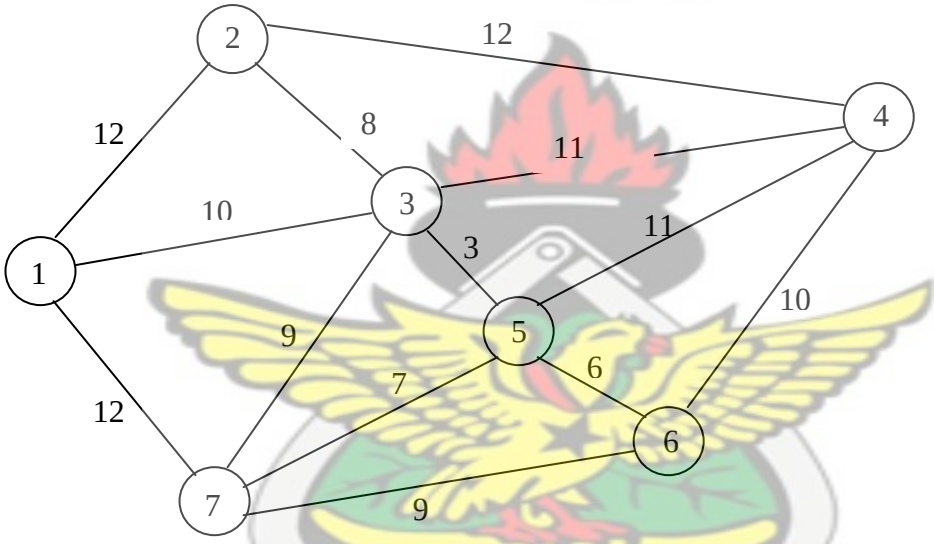


Figure 3.3: Travel Salesman Problem

Therefore, starting from city1 the salesman must choose a route to visit each of the other cities exactly once before returning to city 1. The number next to each link between each pair of cities represents the distance (or cost or time) between these cities. We assume that the distance is the same in either direction. The objective is to determine which route will minimize the total distance that the salesman must travel.

Let the initial trial solution for the network in figure 3.3 is to visit the cities in numerical order: 1-2-3-4-5-6-7-1 with d(i,j) representing the distance from city*i* to city*j*

Trial solution:1-2-3-4-5-6-7-1 gives distance

Distance=d(1,2)+d(2,3)+d(3,4)+d(4,5)+d(5,6)+d(6,7)+d(7,1)=69

If we select, say, the subsequence 3-4 and reverse it, we obtain the new trial solution:

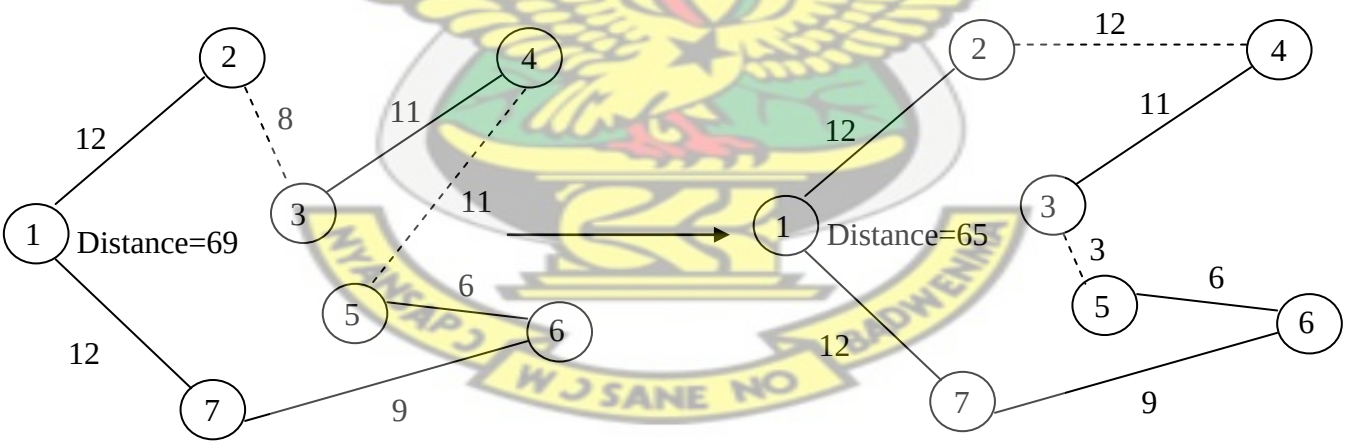
1-2-4-3-5-6-7 with distance

Distance =d(1,2)+d(2,4)+d(4,3)+d(3,5)+d(5,6)+d(6,7)+d(7,1)=65

Thus, this particular sub-tour reversal has succeeded in reducing the distance for the complete tour from 69 to 65.

Figure 3.4 below depicts this sub-tour reversal, which leads from the initial solution on the left to the new trial solution on the right.

The dashed lines indicate the links that are deleted from the tour (on the left) or added to the tour (on the right).



(a)Initial solution (b):New solution

Figure 3.4: A sub-tour reversal that replaces the tour on the left (the initial trial solution) by the tour on the right (the new trial solution)

Applying the sub-tour reversal algorithm to this example starting with 1-2-3-4-5-6-7-1 as the initial solution, there are four possible sub-tour reversals that would improve upon this

solution. These sub-tour reversals are as listed in the second, third, fourth and fifth rows below. 1-2-3-4-5-6-7-1

Distance =d(1,2)+d(2,3)+d(3,4)+d(4,5)+d(5,6)+d(6,7)+d(7,1)= 69 (i)

Reverse ii-iii: 1-3-2-4-5-6-7-1

Distance =d(1,3)+d(3,2)+d(2,4)+d(4,5)+d(5,6)+d(6,7)+d(7,1)= 68 (ii)

Reverse iii-iv: 1-2-4-3-5-6-7-1

Distance =d(1,2)+d(2,4)+d(4,3)+d(3,5)+d(5,6)+d(6,7)+d(7,1)= 65 (iii)

Reverse iv-v: 1-2-3-5-4-6-7-1

Distance =d(1,2)+d(2,3)+d(3,5)+d(5,4)+d(4,6)+d(6,7)+d(7,1)= 65 (iv)

Reverse v-vi: 1-2-3-4-6-5-7-1

Distance =d(1,2)+d(2,3)+d(3,4)+d(4,6)+d(6,5)+d(5,7)d(7,1)= 66 (v)

The two solutions with Distance = 65 tie for providing the largest decrease in the distance traveled, so suppose that the first of these, 1-2-4-3-5-6-7-1 (as shown on figure 2.5.2) is chosen arbitrarily to be the next trial solution. This completes the first iteration.

The second iteration begins with the tour 1-2-4-3-5-6-7-1 as the current trial solution. For this solution, there is only one sub-tour reversal that will provide an improvement, as listed below in equation vi:

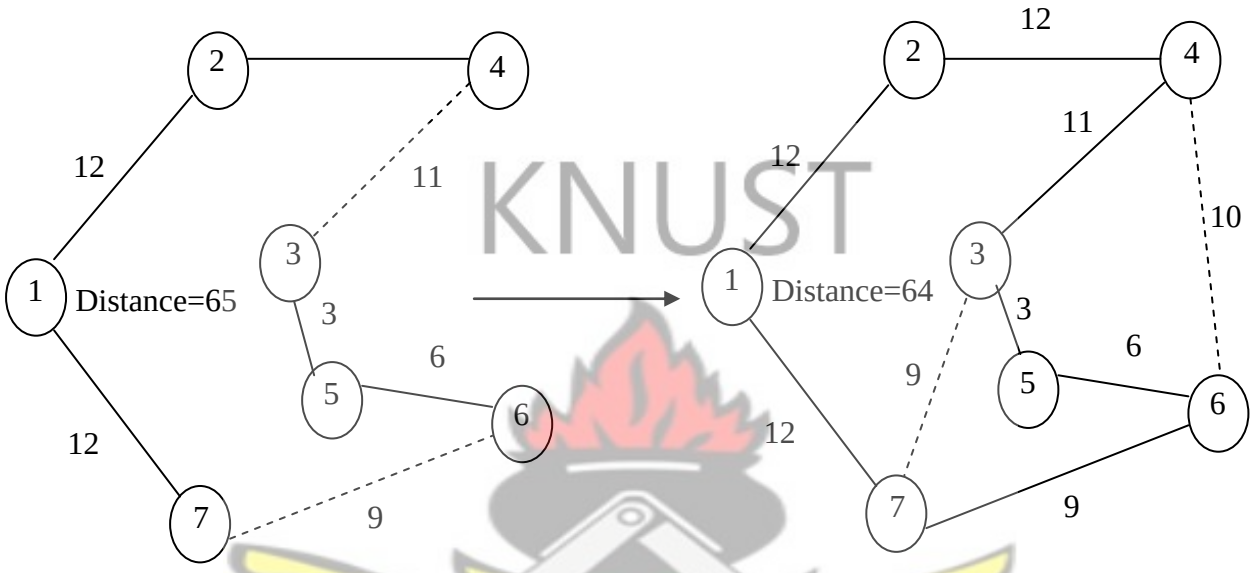
1-2-4-3-5-6-7-1

Distance =d(1,2)+d(2,4)+d(4,3)+d(3,5)+d(5,6)+(6,7)+d(7,1)= 65

Reverse iii-v-vi: 1-2-4-6-5-3-7-1

Distance =d(1,2)+d(2,4)+d(4,6)+d(6,5)+d(5,3)+d(3,7)+d(7,1)= 64 (vi)

Figure 3.5 shows this sub-tour reversal, where the entire subsequence of cities 3-5-6 on the left row is visited in reverse order (6-5-3) on the right.



(a) Initial solution (b) New solution

Figure 3.5: The sub-tour reversal of 3-5-6 that leads from the trial solution on the left to an improved trial solution on the right.

Thus, the tour of equation 6 traverses the link 4-6 instead of 4-3, as well as the link 3-7 instead of 6-7, in order to use the reverse order 6-5-3 between cities 4 and 7.

But 1-2-4-6-5-3-7-1 is not the optimal solution. The optimal solution turns out to be

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

Distance =d(1,2)+d(2,4)+d(4,6)+d(6,7)+d(7,5)+d(5,3)+d(3,1)= 63

(or 1-3-5-7-6-4-2-1 by reversing the order of this entire tour). However, this solution cannot be reached by performing a sub-tour reversal that improves 1-2-4-6-7-5-3-1. In this case, the algorithm stops.

The sub-tour, 1-2-4-6-7-5-3-1, is a local optimum solution because there is no better solution within its local neighborhood that can be reached by performing a sub-tour reversal. The solution is trapped in a local optimum.

3.7 Tabu Search

The concept of Tabu Search (TS) is derived from artificial intelligence where intelligent use of "memory" helps in exploiting useful historical information. The memory concept of TS is quite crucial. Golden et al (1998) defines two types of memory: Short term and Long term memory. The short term memory is imposed to restrict the search from revisiting solutions that have already been considered and to discourage the search from cycling between subsets of solutions. On the other, the long term memory is used to diversify the search.

3.7.1 The tabu search Algorithm

The Tabu Search (TS) based algorithms continue the search even if a locally optimal solution is found. Briefly speaking, TS is a process of subsequent moves from one local optimum to another. The best local optimum found during this process is the resulting solution of TS. Thus, TS uses extended descent local search to escape. However, it has the mechanism of trapping local optima. Consequently, it explores much larger part of the solution space when compared with local search (LS). Hence, TS offers more opportunities for discovering high quality solutions than traditional LS (http//itc.ktu.it/itc32/Misev32.pdf).

The central idea of the TS method is allowing climbing moves when no improving neighboring solution exists, i.e. a move is allowed even if a new solution s' from the neighborhood of the current solution s is worse than the current one. Naturally, the return to the locally optimal solutions previously visited is to be forbidden in order to avoid

cycling of the search. Thus, TS is based on a methodology of prohibitions: some moves are "frozen" (become "tabu") from time to time (http//itc.ktu.it/itc32/Misev32.pdf)

More formally, TS starts from an initial solution $s°$ in S. The process is then continued in an iterative way moving from a solution s to a neighbouring one s'. At each step of the procedure, a certain subset $\Theta'(s)$ of the neighbouring solutions of the current solution is considered, and the move (to the solution $s' \in \Theta'(s) \subseteq \Theta(s)$) that improves most the objective function value f is chosen. Naturally, s' must not necessary be better than s: if there are no improving moves, the TS algorithm chooses one that least degrades (increases) the objective function. In order to eliminate an immediate returning to the solution just visited, the reverse move must be forbidden. This is done by storing the corresponding solution (move) (or its "attribute") in a memory (called a tabu list (T)). The tabu list keeps information on the last |T| moves which have been done during the search process (thus, a move from s to s' is considered as tabu if s', or its "attribute", is contained in T). This way of proceeding hinders the algorithm from going back to a solution reached in the last |T| steps. However, the straightforward prohibition may sometimes lessen the efficiency of the search. Moreover, it might be worth returning after a while to a solution visited previously to search in another promising direction. Consequently, an aspiration criterion is introduced to permit the tabu status to be dropped under certain favourable circumstances. Usually, a move from s to s′ (no matter its status) is permitted if the solution f (s′) at s' is better than the solution $f(s^*)$ at s^* , where s^* is the best solution found so far. The resulting decision rule can thus be described as follows: replace the current solutions by the new solution s′ if f(s′) is better than $f(s^*)$

The search process is stopped as soon as a termination criterion is satisfied (for example, a fixed a priori number of iterations (trials) have been performed .

The framework of the Tabu Search for the problem: Optimize $f(x), x \in S$ where S is the solution space, consists of

Step 1.

Initialization: A starting solution generated by choosing a random solution $x \in S$. the evaluating function *f (x)* is used to evaluate *x*. The solution is stored in the algorithm memory called the Tabu list.

Step 2..

Neighborhood exploration: All possible neighbors *N (s)* of the solution *x* are generated and evaluated. Solutions in the Tabu list are unreachable neighbors; they are Taboo (Tabu).

Step 3

New solution: A new solution is chosen from the explored neighborhood. This solution should not be found in the Tabu list before it is discovered and has to have the best move evaluation value of *f (x)* for all reachable neighbors of *x*.

i.Do Tabu check on the new solution. If successful, replace the current solution and update the Tabu list and other Tabu attributes. Here the new solution evaluation can be worse compared with that of the current solution. This enables the procedure not to be trapped at a local optimum.

ii.If the solution is in the Tabu list, then check the aspiration level. If successful replace the current solution and update the Tabu list and other Tabu attributes.

iii.If checks (i) and (ii) are not successful, then keep the current solution, otherwise replace the current solution by the new solution.

iv.Compare the best to the current solution. If the current solution is better than the best solution, then replace the best solution.

v.Until loop condition is satisfied, go to Step 2.

vi.Until termination condition is satisfied, go to Step 1

3.6.1.1 Prototype Example

Using the matrix representing a complete graph of figure 3.6, we find the optimal value by applying the Tabu Search algorithm.

	1	2	3	4	5	6	7
1	0	12	10	21	13	21	12
2	12	0	8	12	11	17	17
3	10	8	0	11	3	9	9
4	21	12	11	0	11	10	18
5	13	11	3	11	0	6	7
6	21	17	9	10	6	0	7
7	12	17	9	18	7	7	0

Figure 3.6: Tabu Search algorithm.

First iteration

Tabu List = [9, 9, 9]

Tabu position = [0, 0, 0, 0, 0, 0, 0, 0]

Tabu state = [0, 0, 0, 0, 0, 0, 0, 0, 3]

We randomly take the initial solution $x^{(0)}$=[1,2,3,4,5,6,7,1]

Objective value ($x^{(0)}$) =d ($x^{(0)}$)

d($x^{(0)}$)=d(1,2)+d(2,3)+d(3,4)+d(4,5)+d(5,6)+d(6,7)+d(7,1)=69

We find the move values by applying the method

$$(i, j) = [d(i-1, j) + d(j, i) + d(i, j+1)] - [d(i-1, j+1) + d(i, j) + d(j, j+1)]$$

Where i *and* j are the move values.

We check the move values of (2,3),(3,4),(4,5),(5,6) by using the matrix

$(2,3) \rightarrow i = 2,\ j = 3$

$$(2,3) = [d(1,3) + d(3,2) + d(2,4)] - [d(1,2) + d(2,3) + d(3,4)] = 10+8+12-12-8-11 = -1$$

$(3,4) \rightarrow i = 3,\ j = 4$

$$(3,4) = [d(2,4) + d(4,3) + d(3,5)] - [d(2,3) + d(3,4) + d(4,5)] = 12+11+3-8-11-11 = -4$$

$(4,5) \rightarrow i = 4,\ j = 5$

$$(4,5) = [d(3,5) + d(5,4) + d(4,6)] - [d(3,4) + d(4,5) + d(5,6)] = 3+11+10-11-11-6 = -4$$

$(5,6) \rightarrow i = 5,\ j = 6$

$$(5,6) = [d(4,6) + d(6,5) + d(5,7)] - [d(4,5) + d(5,6) + d(6,7)] = 10+6+7-11-6-9 = -3$$

We break tie arbitrary by considering the least move value (3, 4)

Thus d ($x^{(0)}$) + the move value (3, 4) =69-4=65.

By swapping (3,4) we get the new solution $x^{(1)}$ =[1,2,4,3,5,6,7,1] and the objective value

d($x^{(1)}$)= $d(1,2) + d(2,4) + d(4,3) + d(3,5) + d(5,6) + d(6,7) + d(7,1)$ =65

Since d ($x^{(1)}$) <d ($x^{(0)}$), we assign $x^{(0)} \leftarrow x^{(1)}$

Second Iteration

Then new solution is $x^{(0)}$ = [1, 2, 4, 3, 5, 6, 7, 1]

Tabu List = [4, 9, 9]

Tabu position= [0, 0, 0, 0, 0, 0, 0, 1]

Tabu state= [0, 0, 0, 1, 0, 0, 0, 0, 2]

We find the move values (2, 4), (3, 5), (5, 6)

$(2,4) \rightarrow i = 2,\ j = 4$

$(2,4) = [d(1,4) + d(4,2) + d(2,5)] - [d(1,2) + d(2,4) + d(4,5)]$ = 21+12+11-12-12-12=9

$(3,5) \rightarrow i = 3,\ j = 5$

$(3,5) = [d(2,5) + d(5,3) + d(3,6)] - [d(2,3) + d(3,5) + d(5,6)]$ =11+3+9-8-3-6=6

$(5,6) \rightarrow i = 5,\ j = 6$

$(5,6) = [d(4,6) + d(6,5) + d(5,7)] - [d(4,5) + d(5,6) + d(6,7)]$ =10+6+7-11-6-9= -3

The least move value is (5, 6)

Thus d ($x^{(0)}$) + the move value (5, 6) =65-3=62

By swapping (5, 6) we obtain the solution $x^{(1)}$ = [1, 2, 4, 3, 6, 5, 7, 1].

d($x^{(1)}$)= $d(1,2) + d(2,4) + d(4,3) + d(3,6) + d(6,5) + d(5,7) + d(7,1) = 62$

Since d ($x^{(1)}$) <d ($x^{(0)}$), we assign $x^{(0)} \leftarrow x^{(1)}$

Third Iteration

The new solution is $x^{(0)}$ = [1, 2, 4, 3, 6, 5, 7, 1].

Tabu List = [6, 4, 9]

Tabu position= [0, 0, 0, 0, 0, 0, 1, 1]

Tabu state= [0, 0, 0, 1, 0, 1, 0, 0, 1]

We find the move values (2, 4), (3, 6)

$(2,4) \rightarrow i = 2,\ j = 4$

$(2,4) = [d(1,4) + d(4,2) + d(2,5)] - [d(1,2) + d(2,4) + d(4,5)]$ = 21+12+11-12-12-11=9

$(3,6) \rightarrow i = 3,\ j = 6$

$(3,6) = [d(2,6) + d(6,3) + d(3,7)] - [d(2,3) + d(3,6) + d(6,7)]$ =17+9+9-8-9-9= 9

We break tie arbitrary by taking the move value (3, 6)

Thus d ($x^{(0)}$) + the move value (3, 6) =65+9=74

$x^{(1)}$ =[1,2,4,6,3,5,7,1]

d ($x^{(1)}$) =65

The process continues until the optimal solution is obtained.

3.8 Simulated Annealing

Simulated annealing (SA) is a generic probabilistic metaheuristic for the global optimization problem of applied mathematics, namely locating a good approximation to the global minimum of a given function in a large search space. It is often used when the search space is discrete (e.g., all tours that visit a given set of cities). For certain problems,

simulated annealing may be more effective than exhaustive enumeration — provided that the goal is merely to find an acceptably good solution in a fixed amount of time, rather than the best possible solution.

The name and inspiration come from annealing in metallurgy, a technique involving heating and controlled cooling of a material to increase the size of its crystals and reduce their defects. The heat causes the atoms to become unstuck from their initial positions (a local minimum of the internal energy) and wander randomly through states of higher energy; the slow cooling gives them more chances of finding configurations with lower internal energy than the initial one.

By analogy with this physical process, each step of the SA algorithm replaces the current solution by a random "nearby" solution, chosen with a probability that depends on the difference between the corresponding function values and on a global parameter T (called the temperature), that is gradually decreased during the process. The dependency is such that the current solution changes almost randomly when T is large, but increasingly "downhill" as T goes to zero. The allowance for "uphill" moves saves the method from becoming stuck at local minima—which are the bane of greedier methods.

The method was independently described by Kirkpatrick et al in 1983 and by V. Černý in 1985 . The method is an adaptation of the Metropolis-Hastings algorithm, a Monte Carlo method to generate sample states of a thermodynamic system, invented by N. Metropolis et al. in 1953 .

3.8.1 Using simulated Annealing to solve TSP

The TSP was one of the first problems to which simulated annealing was applied, serving as an example for both Kirkpatrick et al, (1983), and Cerny, (1985)., Since then the TSP

has continued to be a prime test bed for the approach and its variants. Most adaptations have been based on the simple schema presented in Figure below, with implementations differing as to their methods for generating starting solutions (tours) and for handling temperatures, as well as in their definitions of equilibrium, frozen, neighbor, and random. Note that the test in Step *6* is designed so that large steps uphill are unlikely to be taken except at high temperatures *t*. The probability that an uphill move of a given cost will be accepted declines as the temperature is lowered. In the limiting case, when $T = 0$, the algorithm reduces to a randomized version of iterative improvement, where no uphill moves are allowed at all.

3.8.1.1 General schema for a simulated annealing algorithm

Step 1. Generate a starting solution S and set the initial solution $S^* = S$.

Step 2. . Determine a starting temperature T.

Step 3. While not yet at *equilibrium* for this temperature, do the following:

Step 4. Choose a *random neighbor* S' of the current solution.

Step 5 . Set $\Delta = Length(S') - Length(S)$.

Step6.. If $\Delta \leq 0$ (downhill move):

Set $S = S'$.

.If $Length(S) < Length(S^*)$, set $S^* = S$.

Else (uphill move):

Step 7 .Choose a random number r uniformly from [0, 1].

If $r < e^{-\Delta/T}$, set $S = S'$.

Step 8. End ‘‘While not yet at equilibrium’’ loop.

Step 9. Lower the temperature T.

Step 10. End the when the improved solution is obtained else return to S *.

3.8.1.2 Prototype Example

Considering Figure 3.5

Taking the initial solution to be in the tour in the order: 1-2-3-4-5-6-7-1

We use the parameters;

$T_0 = 20 \quad T_{k+1} = \alpha T_k \quad \alpha = 0.5$

Stop when $T < 0.1$

First Iteration

Assuming x^0 =1-2-3-4-5-6-7-1

d(x^0)=d(1,2)+d(2,3)+d(3,4)+d(4,5)+d(5,6)+d(6,7)+d(7,1)=69

Using the subtour reversal as local search to generate the new solution x^1 =1-3-2-4-5-6-7-1

d(x^1)=d(1,3)+d(3,2)+d(2,4)+d(4,5)+d(5,6)+d(6,7)+d(7,1)=68

$\delta = d(x^1) - d(x^0)$ =68-69=-1

Since $\delta < 0$ then we set $x^0 \leftarrow x^1$

We then update the temperature $T_1 = \alpha T_0$ =0.5(20)=10

Second Iteration

d(x^0)=68

By sing the subtour reversal as local search to generate the new solution 1-2-3-5-4-6-7-1

x^1 =1-2-3-5-4-6-7-1

d(x^1)=d(1,2)+d(2,3)+d(3,5)+d(5,4)+d(4,6)+d(6,7)+d(7,1)=65

$\delta = d(x^1) - d(x^0)$ =65-68=-3

Since $\delta < 0$ then we set $x^0 \leftarrow x^1$

Updating the temperature , T_2 =0.5(10)=5

Third Iteration

d(x^0)=65

Using the subtour reversal as local search to generate the new solution 1-2-3-4-6-5-7-1

x^1 =1-2-3-4-6-5-7-1

d(x^1)=d(1,2)+d(2,3)+d(3,4)+d(4,6)+d(6,5)+d(5,7)+d(7,1)=66

$\delta = d(x^1) - d(x^0)$ =66-65=1

Since $\delta > 0$ we the apply Boltzmann's condition $m = e^{-\delta / T_2}$ =0.81

A random number would be generated from a computer say θ

If m> θ then we set $x^0 \leftarrow x^1$ otherwise $x^1 \leftarrow x^0$

Updating the temperature, $T_3 = 0.5(5) = 2.5$

The process will continue until the final temperature is obtained.

3.9 The Ant Colony Optimization

The Ant Colony Optimization (ACO) algorithm is a nature-inspired cybernetic method in artificial intelligence. ACO can be considered to be a cognitive informatics (CI) model of social animals like ants rather than the CI model of humans .The idea of ACO comes from the ants' behavior, which is different from traditional mathematics-based cybernetic techniques. The ACO algorithm does a surprisingly successful performance in the solution of NP-hard problems, which draws more and more attention on ACO research, particularly to the study of its theoretical foundation.

3.9.1 Variations of ACO

Different ACO algorithms are discussed subsequently:

3.9.2 ANT SYSTEM (AS)

In AS, *K* artificial ants probabilistically construct tours in parallel exploiting a given pheromone model. Initially, all ants are placed on randomly chosen cities. At each iteration, each ant moves from one city to another, keeping track of the partial solution it has constructed so far. The algorithm has two fundamental components:

(i) the amount of pheromone on arc (i, j), τ_{ij}

(ii) desirability of arc (i, j), η_{ij}

where arc (i, j) denotes the connection between cities i and j.

At the start of the algorithm an initial amount of pheromone τ_0 is deposited on each arc: $\tau_{ij} = \tau_0 = \frac{K}{L_O}$, where L_0 is the length of an initial feasible tour and K is the number of ants.

In AS, the initial tour is constructed using the nearest-neighbor algorithm; however, another TSP heuristic may be utilized as well. The desirability value (also referred to as

visibility or heuristic information) between a pair of cities is the inverse of their distance $\eta_{ij} = \frac{1}{d_{ij}}$ where $d_i j$ is the distance between cities i and j. So, if the distance on the arc (i, j) is long, visiting city j after city i (or vice-versa) will be less desirable.

Each ant constructs its own tour utilizing a transition probability: an ant k positioned at a city i selects the next city j to visit with a probability given by

$$p_{ij}^k = \begin{cases} \dfrac{[\tau_{ij}]^\alpha . [\eta_{ij}]^\beta}{\sum_{l \in N_i^k} [\tau_{ik}]^\alpha . [\eta_{ik}]^\beta}, & j \in N_i^k \\ 0 & \text{, otherwise} \end{cases} \qquad (3.17)$$

where, N_i^k denotes the set of not yet visited cities; α and β are positive parameters to control the relative weight of pheromone information τ_{ij} and heuristic information η_{ij}. After each ant has completed its tour, the pheromone levels are updated. The pheromone update consists of the pheromone evaporation and pheromone reinforcement. The pheromone evaporation refers to uniformly decreasing the pheromone values on all arcs. The aim is to prevent the rapid convergence of the algorithm to a local optimal solution by reducing the probability of repeatedly selecting certain cities. The pheromone reinforcement process, on the other hand, allows each ant to deposit a certain amount of pheromone on the arcs belonging to its tour. The aim is to increase the probability of selecting the arcs frequently used by the ants that construct short tours. The pheromone update rule is the following:

$$\tau_{ij} \leftarrow (1-\rho).\tau_{ij} + \sum_{k=1}^{K} \Delta\tau_{ij}^k \quad \forall (i, j) \ . \qquad (3.18)$$

In this formulation $\rho\,(0 < \rho \leq 1)$ is the pheromone evaporation parameter and $\Delta\tau_{ij}^k$ is the amount of pheromone deposited on arc (i, j) by ant k and is computed as follows:

$$\Delta\tau_{ij}^{k} = \begin{cases} \dfrac{1}{L_k}, \text{ if k}^{\text{th}} \text{ ant uses path } (i,\ j) \text{ in its tour} \\ 0 \quad \text{, otherwise} \end{cases} \tag{3.19}$$

where L_k is the tour length constructed by the *k*-th ant.

3.9.3 Algorithm 1. Ants System Algorithm

Input: a combinatorial optimization problem (S, Ω, f)

Step 1.: Initializes the pheromone matrix $T(0), t = 0$

($T(t)$ is the pheromone matrix at time t)

Step 2.: $S_{bs}(t) \leftarrow Null$

($s_{bs}(t)$ is the best so-far solution at time t)

While the termination condition is not satisfied **do**

Step 3 $t = t + 1, S_{iter}(t) \leftarrow \varnothing$

($S_{iter}(t)$ is the set solutions obtained by ants by at t)

for $j = 1, ...K$ **do**

Step 4 The j-th artificial builds solution s

Step 5. **if** $(f(s) \le f(s_{bs}(t)))$ or $(s_{bs}(t) = NULL)$ **then**

$s_{bs} \leftarrow s$

Step 6 $S_{iter}(t) \leftarrow S_{iter}(t) \bigcup \{s\}$

end **while**

Output: the best-so-far solution $s_{bs}(t)$

3.9,4 IMPROVEMENT OF ANT SYSTEM

The success of ant heuristic lie sorely on the door steps of the pheromone trial. A substantial research on ACO has focused on how to improve AS all in the aim of improving the tour length. Some of these AS improvement algorithms are

(i) Elitist Ant System (EAS);

(ii) Rank Based Ant System (A S Rank);

(iii) Ant Colony System (ACS) and

(iv) Max-Min Ant System (MMAS)

3.9.5 ELITIST ANT SYSTEM (EAS)

In the EAS an elitist strategy is implemented by further increasing the pheromone levels on the arcs belonging to the best tour achieved since the initiation of the algorithm. That best-so-far tour is referred to as the "global-best" tour. The pheromone update rule is as follows:

$$\tau_{ij} \leftarrow (1-\rho)\tau_{ij} + \sum_{k=1}^{K} \Delta\tau_{ij}^{k} + w\Delta\tau_{ij}^{gb} \qquad \forall(i,\ j) \tag{3.20}$$

Here, w denotes the weight associated with the global-best tour and τ_{ij}^{gb} is the amount of pheromone deposited on arc (i, j) by the global-best ant and calculated by the following formula:

$$\tau_{ij}^{gb} = \begin{cases} \dfrac{1}{L^{gb}}, & \text{if the global best ant uses arc } (i,\ j) \text{ in its tour} \\ 0, & \text{otherwise} \end{cases}$$

here L^{gb} is the length of global-best tour.

3.9.6 RANK BASED ANT SYSTEM (Rank AS)

In the *AS*rank a rank-based elitist strategy is adopted in an attempt to prevent the algorithm from being trapped in a local minimum. In this strategy, *w* best ranked ants are used to update the pheromone levels and the amount of pheromone deposited by each ant decreases with its rank. Furthermore, at each iteration, the global-best ant is allowed to deposit the largest amount of pheromone. The pheromone update rule is given by:

$$\tau_{ij} \leftarrow (1-\rho)\tau_{ij} + \sum_{k=1}^{w-1}(w-r)\Delta\tau_{ij}^{r} + w\Delta\tau_{ij}^{gb} \qquad \forall(i,\ j) \tag{3.21}$$

3.9.7 ANT COLONY SYSTEM (ACS)

The ACS attempts to improve AS by increasing the importance of exploitation versus exploration of the search space. This is achieved by employing a strong elitist strategy to update pheromone levels and a pseudo-random proportional rule in selecting the next node to visit. The strong elitist strategy is applied by using the global-best ant only to increase the pheromone levels on the arcs that belong to the global-best tour:

$$\tau_{ij} \leftarrow (1-\rho)\tau_{ij} + \rho\Delta\tau_{ij}^{gb} \qquad \forall(i,\ j)$$

The mechanism of the pseudo-random proportional rule is as follows: an ant *k* located at customer *i* may either visit its most favorable city or randomly select a city. The selection rule is the following:

$$j^{k} = \begin{cases} \underset{j\in N_i^k}{\operatorname{argmax}}\ \tau_{ij}^{\alpha}\eta_{ij}^{\beta}\tau_{ij} & z \le z_0 \\ J^{k}, & \text{otherwise} \end{cases}$$

where *z* is a random variable drawn from a uniform distribution *U*[0,1] and z_0 $(0 \le z_0 \le 1)$ is a parameter to control exploitation versus exploration. j^k is selected according to the

probability distribution p_{ij}^{k}. ACS also uses local pheromone updating while building solutions: as soon as an ant moves from city *i* to city *j* the pheromone level on arc (*i*, *j*) is reduced in an attempt to promote the exploration of other arcs by other ants. The local pheromone update is performed as follows:

$$\tau_{ij} \leftarrow (1-\xi)\tau_{ij} + \xi\tau_0$$

where ξ is a positive parameter less than 1.

Similar to ACS, uses either the global-best ant or the iteration-best ant alone to reinforce the pheromone.

3.9.8 Max-Min Ant System (MMAS*):*

Stuuzle and Hoos, (2000), proposed the MMAS algorithm to have more control on the pheromone trail, so as to avoid the stagnation situation in which all ants are stuck within a local optimum.

According to Stutzle et al, (200), local search is used to improve the algorithm, the importance of local heuristic information is replace by local search. Therefore, local heuristic information is ignored in this version of state transition rule.

The state transition rule used is either the random-proportional rule or the pseudo-random-proportional rule. The pheromone trail is updated when all ants complete their solution construction by $\tau_{ij}^{new} = \rho\tau_{ij}^{old} + \Delta\tau_{ij}^{e}$,

where either the best solution in this iteration and the best solution found so far is used for $\Delta\tau_{ij}^{e}$ All τ_{ij} are initialized as $\tau_{\max}$ and $\tau_{\min} \le \tau_{ij} \le \tau_{\max}$.

StUtzle et al, (1999), also proposed a variation of the state transition rule as $P_{ij} = \dfrac{\tau_{ij}}{\sum_{l \Xi u} \tau_{il}}$

Algorithm 2: The algorithm MMAS*.

Step 1. **function** *MMAS* on G=(V,E)* is

Step 2. $\tau(e) \leftarrow {}^{1}\!/\!_{|V|}, \forall e \in E;$ where E is the of set edges and V is the set of vertices

Step 3. $x^* \leftarrow construct(\tau);$

Step 4 $update(\tau, x^*);$ This is done by using the model $\tau_{ij} \leftarrow (1-\rho)\tau_{ij} + \sum_{r=1}^{k} \Delta\tau_{ij}^{r} + \frac{Q}{L^k}$

Step 5 **while true do**

Step 6 $x \leftarrow construct(\tau);$

Step 7 **if** $f(x) > f(x^*)$ **then**

Step 8 $x^* \leftarrow x;$

Step 9 $\tau \leftarrow update(\tau, x^*);$

3.9.9 Mathematical Formulation Of STSP Model

The problem can be defined as follows: Let *G = (V,E)* be a complete undirected graph with vertices *V*, $|V|=n$, where n is the number of cities, and edges *E* with edge length *dij* for *(i,j)*. We focus on the symmetric TSP case in which $C_{ij} = C_{ji}$, for all *(i,j)*.

The problem PI is;

Minimize $Z = \sum_{i \in v} \sum_{j \in v} c_{ij} x_{ij}$ (3.22)

$$\sum_{\substack{j \in v \\ j \neq i}} x_{ij} = 1 \qquad i \in v \qquad (3.23)$$

$$\sum_{\substack{i \in v \\ i \neq j}} x_{ij} = 1 \qquad j \in v \qquad (3.24)$$

$$\sum_{i \in s} \sum_{j \in s} x_{ij} \leq |s| - 1 \qquad \forall s \subset v, s \neq \varnothing \qquad (3.25)$$

$$x_{ij} 0 or 1 \qquad i, j \in v \qquad (3.26)$$

The problem is an assignment problem with additional restrictions that guarantee the exclusion **o**f subtours in the optimal solution. Recall that a subtour in *V* is a cycle that does not include all vertices (or cities). Equation (3.23) is the objective function, which minimizes the total distance to be traveled.

Constraints (3.24) and (3.25) define a regular assignment problem, where (3.23) ensures that each city is entered from only one other city, while (3.24) ensures that each city is only departed to on other city. Constraint (3.25) eliminates subtours. Constraint (3.26) is a binary constraint, where x_{ij} = 1 if edge *(i,j)* in the solution and x_{ij} = 0, otherwise.

3.10 ACO ALGORITHM FOR OUR PROPOSED WORK

The construction graph *G* = (N, A), where the set *A* fully connects the components *N*, is identical to the problem graph, that is the set of states of the problem corresponds to the set of all possible partial tours.

An initial solution is first obtained using the nearest-neighbor heuristic: start at the depot and then select the not yet visited closest feasible customer as the next customer to be visited.

Each artificial ant has a memory called tabu list. The tabu list forces the ant to make legal tours. It saves the cities already visited and forbids the ant to move already visited cities until a tour is completed.

After all cities are visited, the tabu list of each ant will be full. The shortest path found is computed and saved. Then, tabu lists are emptied. This process is iterated for a user-defined number of cycles.

Suppose there are N nodes and b_i is the number of ants at city i. Consider the following notation:

$K = \sum_{i=1}^{n} b_i$: Total number of ants

Step 1. Set the of cities to be visited by the Ants to be N .

Step 2. Add the city already visited by the Ant to the ;Tabu list of the k-th ant

$tabu_k(s)$: s-th city visited by the k-th ant in the tour

Step 3. $\tau_{ij}(t)$: Intensity of trail on edge between city i and city j at time t This is done by the use of the model $\tau_{ij} \leftarrow (1-\rho)\tau_{ij} + \sum_{r=1}^{k} \Delta\tau_{ij}^{r} + \frac{Q}{L^k}$.

Step 4. Calculate the Visibility of edge between city i and city j to be $\eta_{ij} = \frac{1}{d_{ij}}$

η_{ij} is usually assumed as the inverse of the distance between city *i* and city *j* (d_{ij}) Thus,

$$\eta_{ij} = \frac{1}{d_{ij}}$$

After *K* artificial ants are randomly placed on cities, the first element of each ant's tabu list is set to be equal to its starting city. Then, they move to unvisited cities. The probability of moving from city *i* to cityr *j* for the *k*-th ant is defined as: (p_{ij}^k)

$$p_{ij}^k = \begin{cases} \dfrac{[\tau_{ij}]^{\alpha}.[\eta_{ij}]^{\beta}}{\sum_{j \in N}[\tau_{ik}]^{\alpha}.[\eta_{ik}]^{\beta}}, & j \in N_i^k \\ 0, & \text{otherwise} \end{cases} \quad (3.27)$$

where *N* is the set of neighboring cities , α and β are parameters that control the relative importance of pheromone trail versus visibility.

3.10.1. HEURISTIC INFORMATION

Generally in solving TSP, the visibility value between a pair of cities of the Ant is the inverse of their distance, thus $\eta_{ij} = \frac{1}{d_{ij}}$.

3.10.2. INITIAL PHEROMONE TRIALS

In most of the ant colony based algorithms to TSP, initial pheromone trails τ_0 is set equal to the inverse of the best known route distances found for the particular problem. Thus. $\tau_0 = \frac{1}{d_{i,j}}$. When the initial route is constructed, it is started at the initial point and the city

with the highest $\tau_{i,j}$ value is selected as the first city to be visited. Then, the tour is constructed by selecting the not yet visited feasible city with the highest $\tau_{i,j}$ at each time.

3.10.3 ROUTE CONSTRUCTION PROCESS

It is assumed that the number of ants is equal to the number of directors, who are suppose to embark on the inspectional tour of the twelve main sales point of Ghacem ,Ghana. Thus serving as artificial Ants. Then, each ant constructs its own tour by successively selecting a not yet visited feasible customer. The choice of the next director to visit is based on proportional fitness (Roulette Wheel ie the basic part of the selection process is to stochastically select from one generation to create the basis of the next generation, the requirement is that the fittest individuals have a greater chance of survival than weaker ones. This replicates nature in that fitter individuals will tend to have a better probability of survival and will go forward to form the mating pool for the next generation.) in conjunction with the information of both the pheromone trails and the visibility of that choice given in equation $\varphi_{ij} = \tau_{ij}[\eta_{ij}]^{\beta}$, τ_{ij} denotes the amount of pheromone on arc (i, j) and β is power weighting parameter that weights the consistency of arc (i , j).

3.10.4 PHEROMONE UPDATE

Our pheromone update consists of an improved ant system strategy. In this strategy our pheromone update rule is as follows: $\tau_{ij} \leftarrow (1-\rho)\tau_{ij} + \sum_{r=1}^{k} \Delta\tau_{ij}^{r} + \frac{Q}{L^{k}}$

where Q is a constant ie the product of the longest distance between nodes $_{i,j}$ and, where N is the number of cities to be visited by an ant k . L_k is the distance ant k has visited ,

L^k is the length of tour of ant K and $\rho, 0 \leq \rho \leq 1$, is the evaporation factor, which determines the strength of an update.

3. 11.0 Illustrative Example

In order to get more insight of the algorithm, we shall consider a five (5) node TSP problem. The objective is to find a minimum tour required to visit all the five (5) nodes.

A connectivity matrix of figure 3.7 is given in Table 3.5. The values given in the table denotes the distance ''d'' between nodes and it is assumed to be a symmetric TSP problem, in which $d_{ij} = d_{ji}$

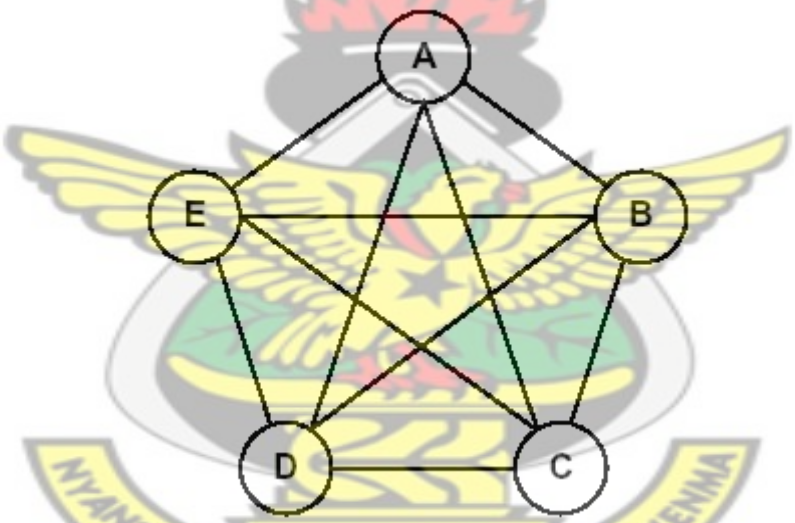


Figure 3.7: **Road Network of the Five (5) Cities**

3.11.1 Distance Matrix for the five cities in Kilometers (km)

The distance matrix was formulated from the network graph of figure 3.6 Where the cities have no direct link, the minimum distance along the edges are considered. The cells indicated zeros shows that there is no distance thus when $C_{ii} = C_{jj} = 0$ .It therefore represents a complete graph for the five cities.

3.11.2 **Table 3.5 shows the connectivity matrix of the five cities which have been Degninated with alphabets The distance between the various are shown and the Zeros indicate that** $d_{i,j} = d_{j,i} = 0$

Table 3.5: Connectivity matrix of TSP in Figure 3.6.

	A	B	C	D	E
A	0	100	125	100	75
B	100	0	50	75	125
C	125	50	0	100	125
D	100	75	100	0	50
E	75	125	125	50	0

Each edge in the graph is given an initial pheromone value (τ_0). For the simplicity of this example τ_0 is set to be 1

Where $n = 5$, thus the number of cities to be visited by an Ant.

The heuristic value $\eta_{ij} = \frac{1}{d_{ij}}$, is the inverse the distance between city i and city j .

The probability of selecting an edge is then equal to

$$p_{ij}^{k}(t) = \frac{[\tau_{ij}(t)]^{\alpha} * [\eta_{ij}]^{\beta}}{\sum_{l \in N} [\tau_{il}(t)]^{\alpha} * [\eta_{il}]^{\beta}} \qquad \text{(i)}$$

Where N is the set of neighboring cities, , $\tau_{i,j}$ is the initial pheromone and

$\eta_{i,j}$ is the heuristic value , α and β are two parameters that control the relative weight of pheromone trail and heuristic value.

In this example, for the sake of simplicity, the value of α and β are set equal to 1.

3.11.3 The table 3.6 shows the **Heuristic value (η) between nodes of the five in table 3.5 above** .These values were obtained through the use of the model $\eta_{ij} = \frac{1}{d_{ij}}$.

Table 3.6: Heuristic value (η) for each edge in Figure 3.6.

	A	B	C	D	E
A	0.000	0.010	0.008	0.010	0.013
B	0.010	0.000	0.020	0.013	0.0008
C	0.008	0.020	0.000	0.010	0.008
D	0.010	0.013	0.010	0.000	0.020
E	0.013	0.008	0.008	0.020	0.000

Since there are 5 cities, assume that the size of the colony of ant is 5. Each ant will start its tour from different city. For example, the first ant starts from city A, the second ant starts from city B, and so on.

Iteration 1: Ant 1 at node 1(A)

$$p_{ij}^{k}(t) = \frac{[\tau_{ij}(t)]^{\alpha} * [\eta_{ij}]^{\beta}}{\sum_{l \in N} [\tau_{il}(t)]^{\alpha} * [\eta_{il}]^{\beta}} \qquad \text{(ii)}$$

$$P_{1,2}^{\ 1}(1) = \frac{[\tau_{1,2}(1)]^{1} * [\eta_{1,2}]^{1}}{\sum_{j \in 5} [\tau_{1,j}(1)]^{1} * [\eta_{1,j}]^{1}}$$

$$P_{1,2}^{\ 1}(1)=\frac{[1.0]^1*[0.010]^1}{(1.0*0.01)+...+(1.0*0.013)}=\frac{0.01}{0.041}=0.2439024$$

$$P_{1,3}^{\ 1}(1)=\frac{[1.00]^1*[0.008]^1}{[1.0*0.010]^1+...+[1.0*0.013]^1}=\frac{0.008}{0.041}=0.1951219$$

$$P_{1,4}^{\ 1}(1)=\frac{[1.00]^1*[0.010]^1}{[1.0*0.010]^1+...+[1.0*0.013]^1}=\frac{0.01}{0.041}=0.24390$$

$$P_{1,5}^{\ 1}(1)=\frac{[1.00]^1*[0.013]^1}{[1.0*0.010]^1+...+[1.0*0.013]^1}=\frac{0.013}{0.041}=0.31707$$

The first ant starts the tour from city A. There are four neighboring cities to be considered by the ant.

The probability of choosing any edge leading to certain city is calculated using the

Probability decision rule, ie $p_{ij}^{k}(t)=\frac{[\tau_{ij}(t)]^{\alpha}*[\eta_{ij}]^{\beta}}{\sum_{l\in N}[\tau_{il}(t)]^{\alpha}*[\eta_{il}]^{\beta}}$

Table 3.7 Shows the neighboring cities left for the Ant 1 to select from.

B	C	D	E
0.24	0.19	0.24	0.32

Using a stochastic process, (Roulette Wheel), the ant chooses the next city. Assume that the ant takes city B as the next city to visit.

The ant will update its memory and put city B in its *Tabu List* (to add to A)

When the ant arrives at city B, there are 3 cities left to visit. The probability of choosing these cities is given in the table.3.8.

Table 3.8 Shows the neighboring cities left for the Ant 1 to select from

C	D	E
0.48	0.32	0.19

$$P_{2,3}{}^{1}(1) = \frac{[1.00]^1 * [0.02]^1}{[1.0*0.020]^1 + [1.0*0.013]^1 + [1.0*0.008]^1} = \frac{0.02}{0.041} = 0.487804$$

$$P_{2,4}{}^{1}(1) = \frac{[1.00]^1 * [0.013]^1}{[1.0*0.020]^1 + [1.0*0.013]^1 + [1.0*0.008]^1} = \frac{0.013}{0.041} = 0.317073$$

$$P_{2,5}{}^{1}(1) = \frac{[1.00]^1 * [0.008]^1}{[1.0*0.020]^1 + [1.0*0.013]^1 + [1.0*0.008]^1} = \frac{0.008}{0.041} = 0.195121$$

C	D	E
0.48	0.32	0.19

Assume that city D is taken. The ant will then update its *Tabu List* by adding city D.

There are two neighbors of city D: C and E. The following table shows the probability of choosing each of these cities.

$$P_{4,5}{}^{1}(1) = \frac{[1.00]^1 * [0.010]^1}{[1.0*0.01]^1 + [1.0*0.020]^1} = \frac{0.010}{0.03} = 0.33333$$

$$P_{5,5}{}^{1}(1) = \frac{[1.00]^1 * [0.020]^1}{[1.0*0.01]^1 + [1.0*0.020]^1} = \frac{0.02}{0.03} = 0.6666$$

Table 3.9 Shows the neighboring cities left for the Ant 1 to select from

C	E
0.33	0.66

Assume that the ant selects city E. The content of its *Tabu List* is then: A, B, D, and E. Since there is one remaining city to visit, the next process will certainly take C. The path that was built by the ant is then: $A \rightarrow B \rightarrow D \rightarrow E \rightarrow C$. The length of this path is $L = AB + BD + DE + EC = 100 + 75 + 50 + 125 = 350$.

The remaining ants will proceed according to the same procedure.

3.11.4; Table3.10 summarizes the solutions built by all ants. The last column in Table 3.10 shows the gain obtained by each ant. Since the longest distance between cities is 125, the solution built by the ant must not exceed Q=4 *125 = 500. Thus, the gain of each ant can be formulated as 500/*L*, with *L* is the length of the path of the solution.

Table 3.10 : Shows the Solutions built by all ants in the first iteration

Ant	Path	Length of the path (L)	$\Delta\tau = 500/L$
ant1	$A \rightarrow B \rightarrow D \rightarrow E \rightarrow C$	350	1.43
ant2	$B \rightarrow C \rightarrow D \rightarrow E \rightarrow A$	275	1.82
ant3	$C \rightarrow B \rightarrow D \rightarrow E \rightarrow A$	250	2.00
ant4	$D \rightarrow E \rightarrow A \rightarrow B \rightarrow C$	275	0.82
ant5	$E \rightarrow A \rightarrow B \rightarrow C \rightarrow D$	325	1.54

When all ants finish their tour, they will back track and update the pheromone along their path by putting additional pheromone ($\Delta\tau$). Note that, the amount of $\Delta\tau$ is proportional to the gain obtained by the ant.

$$\Delta\tau_{i,j} = \sum_{k=1}^{N} \Delta\tau_{ij}^{\,k} \qquad \text{(iii)}$$

Where $\Delta\tau_{ij}^{\,k}$ is the adding pheromone to the arcs in the tour ant k has visited,

$$\Delta\tau_{i,j} = \frac{Q}{L_k}, \qquad \text{(iv)}$$

where Q is constant, ie the product of the longest distance between nodes i and j where N is the number of cities to be visited by an ant k. L_k is the distance ant k has visited. For example $Q = 4*125 = 500$

The new pheromone value is given by the following model

$\tau(t+1) = \tau(t) + \Delta\tau(t)$. (v)

Consider, for example, edge *AB* was used by ant1, ant4 and ant5. The new pheromone value for edge *AB* is therefore equal to 1 + 1.43 + 1.82 + 1.54 = 5.79.

Then, pheromone will evaporate according to the formula:

$\tau = (1 - \rho) * \tau$ (vi)

Assume that ρ is equal to 0.2. Then the pheromone value on edge *AB* is equal to 0.8 * 5.79 = 4.63. The calculation of pheromone value is performed for all edges.

$AB \rightarrow \tau = (1-0,2)*5.79 = 4.632$,where $\tau + \Delta\tau = 1+1.43+1.82+1.54 = 5.79$

$AC \rightarrow \tau = \tau + \Delta\tau = 1.0+0.0 = 1.0$, Since ant k did not have direct link to C

$\Rightarrow \Delta\tau = 0$

$AC \rightarrow \tau = 0.8*1.0 = 0.8$

$AD \rightarrow \tau = 0.8*1.0 = 0.8$

$AE \rightarrow \tau = 0.8*(1+1.82+2.00+1.82+1.54) = 6.544$, where

$\tau + \Delta\tau = (1+1.82+2.00+1.82+1.54) = 8.18$

3.11.5; The Table 3.11 shows the new pheromone values on each edge at the end of iteration

Table 3.11: Pheromone values for each edge after iteration 1.

	initial pheromone value					new pheromone value				
	A	B	C	D	E	A	B	C	D	E
A	0.00	1.00	1.00	1.00	1.00	0.00	4.63	0.80	0.80	6.54
B	1.00	0.00	1.00	1.00	1.00	4.63	0.00	6.54	3.54	0.80
C	1.00	1.00	0.00	1.00	1.00	0.80	6.54	0.00	0.80	0.80
D	1.00	1.00	1.00	0.00	1.00	0.80	3.54	0.80	0.00	6.45
E	1.00	1.00	0.80	1.00	0.00	6.54	0.80	0.80	6.45	0.00

3,11,6 Figure 3.7 (a) shows the visualization of pheromone values on the edges. In this figure, the darker the edge, the higher the pheromone. The best solution found by the heuristic in the first iteration is shown in Figure 3.7 (b).

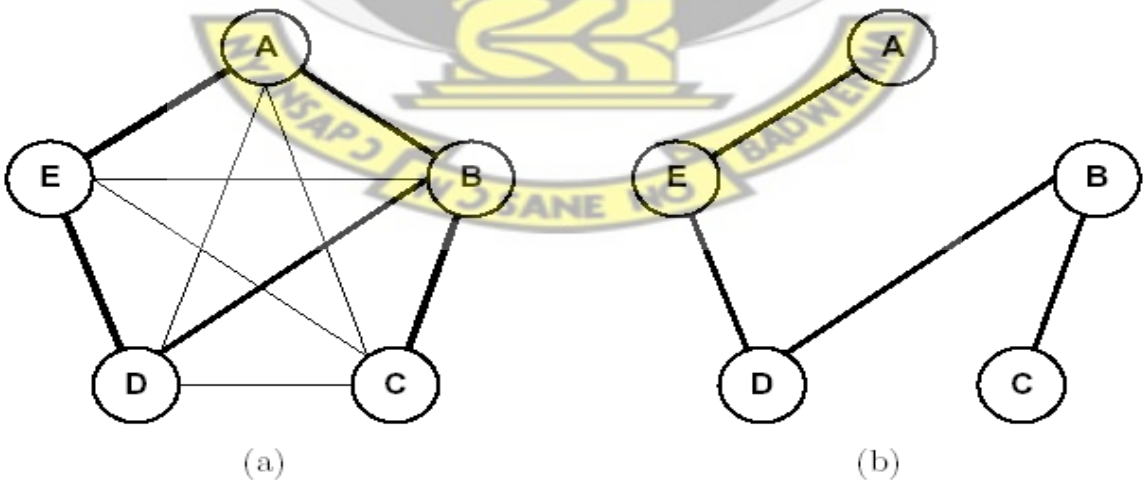


Figure 3.7. shows the visualization of pheromone values on the edges

Iteration 2

The same process that was performed in the first iteration is repeated in the second. However, the initial pheromone values on all edges have changed. Thus, the probability of selecting a certain edge will also change. The higher the pheromone on the edge, the more attractive the edge for an ant to choose.

3.11.7 Assume that all ants have finished their tour construction. The table 3.12 summarizes the solutions built by all ants.

Table 3.12: Solutions built by the ant in the second iteration.

Ant	**Path**	**Length of the path (L)**	**$\Delta\tau$ =5 00/L**
ant1	*A→E→D→B→C*	250	2.00
ant2	*B→C→D→E→A*	275	1.82
ant3	*C→B→D→E→A*	250	2.00
ant4	*D→E→A→B→C*	275	1.82
ant5	*E→A→D→B→C*	300	1.67

3.11.8 The pheromone update and pheromone evaporation procedures are then performed. This will change the value of pheromone on each edge. The new pheromone values for edge after iteration 2 are shown on table 3.13

Table 3.13 Shows the Pheromone values for each edge after iteration 2.

	initial pheromone value					new pheromone value				
	A	B	C	D	E	A	B	C	D	E
A	0.00	4.63	0.80	0.80	6.54	0.00	6.45	0.80	2.47	15.84

B	4.63	0.00	6.54	3.54	0.80	6.45	0.00	15.84	9.21	0.80
C	0.80	6.54	0.00	0.80	0.80	0.80	15.84	0.00	2.62	0.80
D	0.80	3.54	0.80	0.00	6.45	2.47	9.21	2.62	0.00	14.09
E	6.54	0.80	0.80	6.45	0.00	15.84	0.80	0.80	14.09	0.00

3.11.9 Figure 3.7 (a) shows the visualization of pheromone values on the edges. As we can see, the lines representing edge AE, ED and BC are very thick. These lines are thicker than the corresponding ones in the previous iteration (see Figure 3.6).

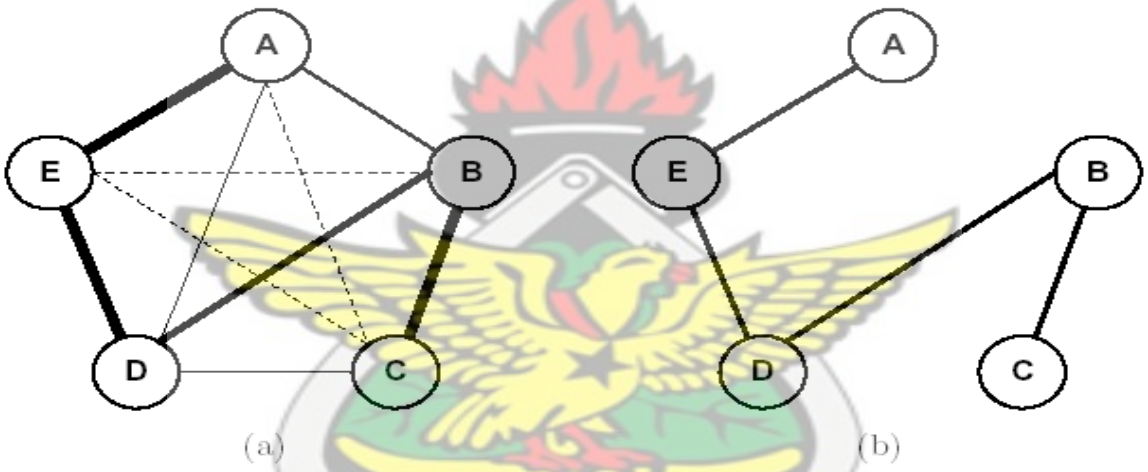


Figure 3.7 (a) shows the visualization of pheromone values on the edges.

3.11.8: The thickness of these lines corresponds to their high pheromone values. On the other hand, the lines representing edge AC, BE and CE are very thin. Since no ant is using these edges, there is no additional pheromone given.

In addition, pheromone evaporation reduces the intensity of pheromone values on these edges.

From Figure 3.7(a), it can be seen that the best solution for the given TSP problem will likely be equal to the one illustrated in Figure 3.7 (b).

In the next chapter,we consider the use of an ACO algorithm called Min-Max Ant System (MMAS) to solve a symmetric TSP problem involving twelve cities.This is illustrated as Algorithm 2 of section 3.9.9.

CHAPTER FOUR

DATA COLLECTION AND ANALYSIS

4.0 Introduction

In this chapter, we shall look at how the data for the work was obtained, how it was used for the intended analysis based on the method(s) discussed in the previous chapter

4.1 Data Collection

We considered a twelve city node graph (major sales point of Ghacem) with the nodes representing the twelve cities, and the edges representing the major roads linking the cities (figure 4.1). Based on this graph, we collected secondary data of the inter-city driving distances from the Ghana Highway Authority. In table 4.1 we have designated each city with a number for convenience.

Table 4.1: Twelve major sales points of Ghacem in Ghana and their numerical representation

City	Allocated number
Tema	1
Accra	2
Cape Coast	3
Takoradi	4
Obuasi	5
Kumasi	6
Koforidua	7
Sunyani	8
Wa	9
Bolgatanga	10
Tamale	11
Ho	12

4.2 Data Analysis

The figure 4.1 below shows the Road Network of the twelve (12) major sales points of Ghacem and their geographical locations on the map of Ghana.

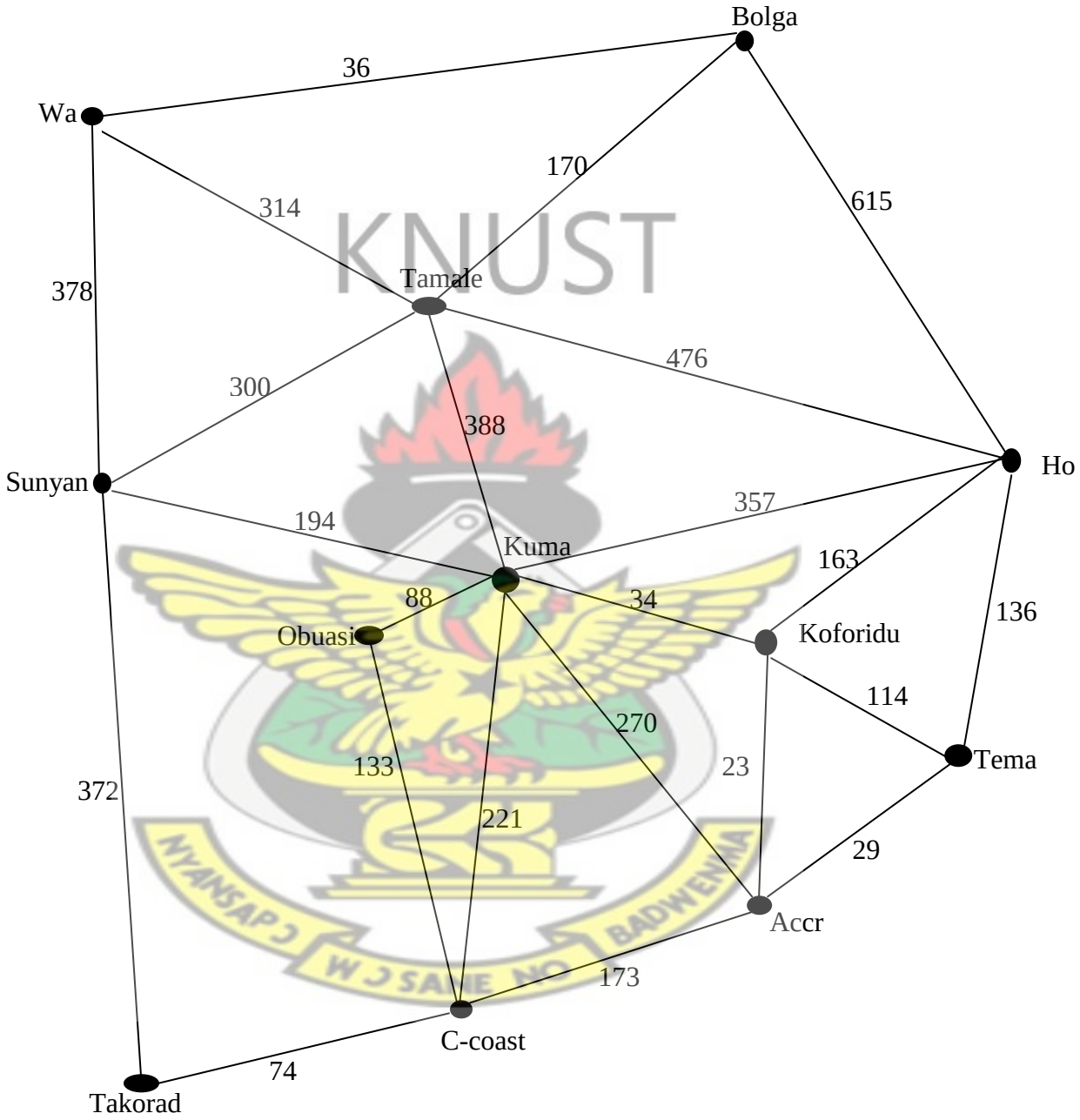


Figure 4.1 Road Network of the twelve (12) major sales points of Ghacem

Table 4.2: Data from the Ghana Highways Authority indicating the matrix for the weighted graph of the major roads linking twelve major Sales of points of Ghacem in Ghana in Kilometers

City/cityj	Tema	Accra	C-coast	Takoradi	Obuasi	Kumasi	Koforidua	Sunyani	Wa	Bolga	Tamale	Ho
Tema	0	29	inf	Inf	Inf	Inf	Inf	inf	inf	Inf	inf	136
Accra	29	0	144	Inf	Inf	270	85	inf	inf	Inf	inf	165
C-coast	inf	144	0	74	133	221	Inf	inf	inf	Inf	inf	inf
Takoradi	inf	Inf	74	0	Inf	242	Inf	inf	inf	Inf	inf	inf
Obuasi	inf	Inf	133	Inf	0	88	Inf	inf	inf	Inf	inf	inf
Kumasi	inf	270	221	242	88	0	194	130	inf	Inf	388	inf
Koforidua	inf	85	inf	Inf	Inf	194	0	inf	inf	Inf	inf	inf
Sunyani	inf	Inf	inf	Inf	Inf	130	Inf	0	378	Inf	388	inf
Wa	inf	Inf	inf	Inf	Inf	Inf	Inf	378	0	368	314	inf
Bolga	inf	Inf	inf	Inf	Inf	Inf	Inf	inf	368	0	170	614
Tamale	inf	Inf	inf	Inf	Inf	388	Inf	300	314	170	0	476
Ho	136	165	Inf	Inf	Inf	Inf	163	inf	inf	614	476	0

C_{ij} = The cost matrix representing the distance from city i to city j.

Where $C_{ij} = C_{ji} = 0$ .ie no direct link from one city to the other.

4.3: Connectivity matrix for the twelve major sales points cities of Ghacem in Kilometers (Km)

(**4.3** The distance matrix was formulated from the connectivity graph of figure4.2 .Where the cities have no direct link ,the minimum distance along the edges are considered .The cells indicated inf shows that there is no direct distance ,thus $C_{ij} = C_{ji} = 0$

Table 4.3: Connectivity matrix for the twelve major sales points cities of Ghacem in Kilometers (Km) (All pair shortest path from table 4.2 by Floyd Warshall's Algorithm)

City/cityj	1	2	3	4	5	6	7	8	9	10	11	12
1	0	29	173	247	352	299	114	429	816	750	612	136
2	29	0	144	218	358	270	85	400	778	770	641	165
3	173	144	0	74	133	221	229	351	729	779	609	309
4	247	218	74	0	213	242	303	372	750	800	630	383
5	352	358	133	213	0	88	282	218	596	646	476	445
6	299	270	221	242	88	0	194	130	508	558	388	357
7	114	85	229	303	282	194	0	324	702	752	582	163
8	429	400	351	372	218	130	324	0	378	470	300	487
9	816	778	729	750	596	508	702	378	0	368	314	790
10	750	770	779	800	646	558	752	470	368	0	170	615
11	612	641	609	630	476	388	582	300	314	170	0	476
12	136	165	309	383	445	357	163	487	790	615	476	0

4.4 In this study each edge in the graph is given an initial pheromone value $\tau_0 = \frac{1}{n} = \frac{1}{12} = 0.0833$ .where $n = 12$. Let heuristic value (η) be equal to the reciprocal of the

distance, ie . $\eta_{ij} = \frac{1}{d_{ij}}$. where d_{ij} is the distance between city(i) to city(j). The probability of selecting an edge is given by $p_{ij}^{k} = \frac{[\tau_{ij}]^{\alpha} * [\eta_{ij}]^{\beta}}{\sum_{l \in N} [\tau_{il}]^{\alpha} * [\eta_{il}]^{\beta}}$, (4.1)

where N = 12 (the set of neighboring Cities (nodes) to be visited by the artificial Ants (The inspectional Team of Ghacem,Ghana) α and β are parameters that control the relative weight of pheromone trial and heuristic value. In this study, the values of α and β are set be 1. Again τ_{Max} and τ_{Min} are set to be 1.0 and 0.01 respectively. In this work, we considered several values for the evaporation rate such as 0.1, 0.02, 0.1,0.2, …

4.4 Heuristic value (η) between nodes of the twelve major Sales of points of Ghacem in Ghana in table

The table 4.4 shows the **Heuristic value (η) between nodes of the twelve major Sales of points of Ghacem in Ghana in table 4.3 above,** These values were obtained through the use of the model $\eta_{ij} = \frac{1}{d_{ij}}$.

Table 4.4 shows the heuristic value (η) for each edge in Figure 4.1

City/cityj	1	2	3	4	5	6	7	8	9	10	11	12
1	0.000	.0.034	0.006	0.004	0.003	0.004	0.009	0.002	0.001	0.001	0.002	0.007
2	0.034	0.000	0.007	0.005	0-003	0.004	0.012	0.003	0.001	0.001	0.002	0.006
3	0.006	0.007	0.000	0.014	0.008	0.005	0.004	0.003	0.001	0.001	0.002	0.003
4	0.004	0.005	0.014	0.000	0.005	0.004	0.003	0.003	0.001	0.001	0.002	0.003
5	0.003	0.003	0.008	0.005	0.000	0.011	0.004	0.005	0.002	0.002	0.002	0.002
6	0.004	0.004	0.005	0.004	0.011	0.000	0.005	0.008	0.002	0.002	0.003	0.003
7	0.009	0.012	0.004	0.003	0.004	0.005	0.000	0.003	0.001	0.001	0.002	0.006
8	0.002	0.003	0.003	0.003	0.005	0.008	0.003	0.000	0.003	0.002	0.003	0.002
9	0.001	0.001	0.001	0.001	0.002	0.002	0.001	0.003	0.000	0.003	0.003	0.001
10	0.001	0.001	0.001	0.001	0.002	0.002	0.001	0.002	0.003	0.000	0.006	0.002
11	0.002	0.002	0.002	0.002	0.002	0.003	0.002	0.003	0.003	0.006	0.000	0.002
12	0.007	0.006	0.003	0.003	0.002	0.003	0.006	0.002	0.001	0.002	0.002	0.000

4.6 The table 4.5 shows the :initial pheromone value (τ_0) for each edge. In this study each edge in the graph is given an initial pheromone value $\tau_0 = \frac{1}{n} = \frac{1}{12} = 0.0833$ .where n is the number of cities to be visited by the Ants.

Table 4.5: Initial pheromone value (τ_0) for each edge is as shown in Figure 4.1.

City/cityj	1	2	3	4	5	6	7	8	9	10	11	12
1	0	0.083	0.083	0.083	0.083	0.083	0.083	0.083	0.083	0.083	0.083	0.083
2	0.083	0	0.083	0.083	0.083	0.083	0.083	0.083	0.083	0.083	0.083	0.083
3	0.083	0.083	0	0.083	0.083	0.083	0.083	0.083	0.083	0.083	0.083	0.083
4	0.083	0.083	0.083	0	0.083	0.083	0.083	0.083	0.083	0.083	0.083	0.083
5	0.083	0.083	0.083	0.083	0	0.083	0.083	0.083	0.083	0.083	0.083	0.083
6	0.083	0.083	0.083	0.083	0.083	0	0.083	0.083	0.083	0.083	0.083	0.083
7	0.083	0.083	0.083	0.083	0.083	0.083	0	0.083	0.083	0.083	0.083	0.083
8	0.083	0.083	0.083	0.083	0.083	0.083	0.083	0	0.083	0.083	0.083	0.083
9	0.083	0.083	0.083	0.083	0.083	0.083	0.083	0.083	0	0.083	0.083	0.083
10	0.083	0.083	0.083	0.083	0.083	0.083	0.083	0.083	0.083	0	0.083	0.083
11	0.083	0.083	0.083	0.083	0.083	0.083	0.083	0.083	0.083	0.083	0	0.083
12	0.083	0.083	0.083	0.083	0.083	0.083	0.083	0.083	0.083	0.083	0.083	0

4.5 Mathematical Formulation Of TSP Model

The problem can be defined as follows: Let *G = (V,E)* be a complete undirected graph with vertices *V, |V|=n*, where n is the number of cities, and edges *E* with edge length *dij* for *(i,j)*. We focus on the symmetric TSP case in which $C_{ij} = C_{ji}$, for all *(i,j)*.

The problem PI is;

$$\text{Minimize } Z = \sum_{i \in v} \sum_{j \in v} c_{ij} x_{ij} \qquad (4.2)$$

Subject to

$$\sum_{\substack{j \in v \\ j \neq i}} x_{ij} = 1 \qquad i \in v \qquad (4.3)$$

$$\sum_{\substack{i \in v \\ i \neq j}} x_{ij} = 1 \qquad j \in v \qquad (4.4)$$

$$\sum_{i \in s} \sum_{j \in s} x_{ij} \leq |s| - 1 \qquad \forall s \subset v, s \neq \varnothing \qquad (4.5)$$

$$x_{ij} \; 0 \; or \; 1 \qquad i, j \in v \qquad (4.6)$$

The problem is an assignment problem with additional restrictions that guarantee the exclusion of subtours in the optimal solution. Recall that a subtour in *V* is a cycle that does not include all vertices (or cities). Equation (4.2) is the objective function, which minimizes the total distance to be traveled.

Constraints (4.3) and (4.4) define a regular assignment problem, where (4,3) ensures that each city is entered from only one other city, while (4.3) ensures that each city is only

departed to on other city. Constraint (4.5) eliminates subtours. Constraint (4,6) is a binary constraint, where x_{ij} = 1 if edge *(i,j)* in the solution and x_{ij} = 0, otherwise.

Algorithm

Table 3.6: Shows the Pseudo-code of the algorithm applied to solve the MMAS

Procedure of MMAS

Step 1

The Directors Inspectional Tour (DIT), whose serve as Artificial Ants in this study, graph was transformed into a TSP graph

Step2

The initial pheromone matrix was computed in Table 3.5.

Set $L_g best = \infty$,iterate=TRUE, i=0

While iterate = TRUE

Set i=i+1

For h =1 to m

Set tabu _h= ∅

Step 3

A city , C_{ij} was randomly selected by ant k as the starting point of the path

C_{ij} was added to tabu_h

For j=I to n-1

Step 4

The next city ,C, was selected according to probability decision rule in (4.2)

City C was added to tabu_h

End-for

Step 5

Compute the length of the path L(h)

If L_g _best >Lb

Set L_g _best >Lb=Lb

If L_g _best has not been improved during the last 15 iterations

Set iterate = FALSE

end -if

If $Lx = Ly,\ {}^{y}\!/_{x} \neq y, 1 \leq x, y \leq m$

Step 6

Reset the pheromone matrix trails to the value τ_{Max}

else

update the pheromone matrix according to the expression in (4.5)

end-if

end while

Step 7

The TSP solution was then transformed into DIT solution.

4.6 .Computational Method

The MMAS proposed by Stuuzle and Hoos, (2000) was coded in Matlab language. The tests were performed on a personal computer, Dell core 5 Dua processor, 3.0GHZ with RAM 2G memory and working on Window7 Operating system.

4.7 Results

The MMAS algorithm was coded used to find the minimum tour of each ant and then selected the best ant tour. After performing 6652800 iterations the result for each ant is shown in table 4.7

Table 4.7 Shows both the tour of an individual Ant and their various distance covered

Ant tour		Dist. Cov. By ant
Ant 1	9 12 2 1 7 4 3 5 6 8 11 10	1874
Ant 2	9 10 11 8 6 5 3 4 7 2 1 12	2238
Ant 3	10 5 6 8 4 3 1 2 7 12 11 9	2272
Ant 4	10 8 6 5 7 2 1 12 3 4 11 9	2445
Ant 5	12 3 4 6 5 11 10 8 9 7 1 2	2908
Ant 6	5 9 10 11 6 4 3 1 2 7 12 8	2397
Ant 7	9 7 2 1 12 11 10 8 6 5 3 4	2541
Ant 8	12 1 2 7 6 5 3 4 10 11 8 9	3041
Ant 9	9 6 5 4 3 1 2 7 12 11 10 8	2319
Ant 10	8 4 3 5 6 1 2 7 12 10 11 9	2348
Ant 11	9 11 10 8 6 4 3 5 7 2 1 12	2541
Ant 12	12 2 1 7 3 4 5 6 8 11 10 9	2505

4.8. Discussions

Considering the total distances covered by the individual Ants, the optimal tour came out to be $12-2-1-7-4-3-5-6-8-11-10-9$

This was obtained by Ant 1.Thus the total tour distance came out to be optimal solution.

ie 1874*km*

Representing the tour

Ho – Accra – Tema – Koforidua – Takoradi – Cape – Coast
*–Obuasi – Kumasi – Sunyani – Temale – Bo*lg *a – Wa*

CHAPTER FIVE

CONCLUSION AND RECOMMENDATION

5.0 Introduction

This chapter basically talks about the conclusion and recommendation of the work

5.1. Conclusion.

- We conclude that the objective of finding the minimum tour from the symmetric TSP model by using Max-Min Ants System (MMAS) Algorithm was successfully achieved.
- We therefore suggested that plying the routes that came out the MMAS model would be of help to minimize their cost since those routes gave the optimal cost of 1874km .

The optimal route is represented as

$$Tema - Accra - Ho - Koforidua - Takoradi - Cape - Coast$$
$$-Obuasi - Kumasi - Sunyani - Temale - Bolga - Wa$$

The total cost distance of their usual tour is 2319KM . Thus

$$Tema \rightarrow Accra \rightarrow Cape - Coast \rightarrow Takoradi \rightarrow Obuasi \rightarrow$$
$$Kumasi \rightarrow Sunyani \rightarrow Wa \rightarrow Bolga \rightarrow Temale \rightarrow Ho \rightarrow Tema$$

5.2 Recommendation

After thorough study of TSP and Maxi-Min Ants System algorithm the following recommendation were made:

- The company are therefore advise to make use the of programme to obtain the optimal tour in the event of cities to be visit being perturbed

- In using the MMAS programme, we therefore advise the company to employ mathematicians who are very good in programming to update the model in event of cities to be visited are charged

- This shows that for the inspectional team of Ghacem ,Ghana to minimized cost in order maximized profit, must seize using their usual route and stick to the new one that came out of model.

- We once again recommend further research into this study by researchers.

REFFERENCE

1. Ackoff, R.L., Arnoff, E.L., and Sengupta, S.S. (1961). "Mathematical Programming". In: *Progress in Operations Research*. R.L. Ackoff, editor. John Wiley and Sons: New York: NY. 105-210.

2. Al-Hboub-Mohamad, H. and Selim Shokrik, Z. (1993). A Sequencing Problem in the Weaving Industry. *European Journal of Operational Research (The Netherlands).* 66(1):6571. Applegate, D., Bixby, R., Chv'atal, V., and Cook, W. (1999). "Finding Tours in the TSP". Technical Report 99885. Research Institute for Discrete Mathematics, Universitaet Bonn: Bonn, Germany.

3. Amponsah, S .K and F.K Darkwah (2007) .Lecture notes on operation Research ,IDL KNUST 62-67

4. Applegate .D, Bixby R.E., Chvatal V., and Cook W. (1994) "Finding cuts in the TSP" a preliminary report distributed at The Mathematical Programming Symposium, Ann Arbor, Michigan.

5. AppleGate D, Bixby R., Chvatal V and Cook W. (1998). On the Solution of the Traveling Salesman Problems. Documenta Mathematica – Extra Volume ICM, chapter 3, pp. 645-656

Applegate, D., Bixby, R., Chv'atal, V. and Cook, W. (2007). *The Traveling Salesman Problem*. Princeton University Press: Princeton, NJ.

6. Applegate, D., Bixby, R., Chvatal, V., Cook, W., and Helsghaun, K. (2004). "The Sweden Cities TSP Solution". http://www.tsp.gatech.edu//sweeden/cities/ cities.htm.

7. Applegate, D., Bixby, R., Chv´atal, V., and Cook, W. (1994): Finding Cuts in the TSP

(A preliminary report), Tech. rep., Mathematics, AT&T Bell Laboratories, Murray Hill, NJ.

Balas, E. and Simonetti, N. (2001). "Linear Time Dynamic Programming Algorithms for New Classes of Restricted TSPs: A Computational Study." *INFORMS Journal on Computing*. 13(1): 56-75.

8. Barachet, L.L. (1957). "Graphic Solution of the Traveling-Salesman Problem". *Operations Research*. 5:841-845.

9. Barahona, F., Gr¨otschel, M., J¨unger, M., and Reinelt, G. (1988): 'An application of combinatorial optimization to statistical physics and circuit layout design', Operations Research 36(3) ,493–513

10. Bellman, R. (1960). "Combinatorial Processes and Dynamic Programming". In: *Combinatorial Analysis*. R. Bellman and M. Hall, Jr., eds. American Mathematical Society: Washington, DC. 217-249.

11. Bellman, R. (1960). "Dynamic Programming Treatment of the TSP". *Journal of Association of Computing Machinery*. 9:66.

12. Bellmore .M and. Nemhauser G. L, (1968) The Traveling Salesman Problem: A Survey Operations Research, Vol. 16, No. 3 pp. 538-558. http://www.jstor.org/stable/168581

13. Bellmore, M. and Nemhauser, G.L. (1968). "The Traveling Salesman Problem: A Survey". *Operations Research*. 16:538-558.

14. Bock, F. (1958). "An Algorithm for Solving Traveling-Salesman' and Related Network Optimization Problems". Research Report, Operations Research Society of America

Fourteenth National Meeting: St. Louis, MO. Problems". Research Report, Operations Research Society of America Fourteenth National Meeting: St. Louis, MO.

15. Burkard, R.E. (1979). "Traveling Salesman and Assignment Problems: A Survey". In: *Discrete Optimization 1.* P.L. Hammer, E.L. Johnson, and B.H. Korte, eds. *Annals of Discrete Mathematics Vol. 4,* North-Holland: Amsterdam. 193-215.

16. Carpaneto, G., Dell'Amico, M. and Toth, P., (1995), "Exact Solution of Large-scale Asymmetric Traveling Salesman Problems", ACM Transactions on Mathematical Software, 21, pp.394–409.

17. Carpaneto, G., Toth, P., (1980)." Some new branching and bounding criteria for the asymmetric traveling salesman problem", Management Science 21, pp.736–743.

18. Cerny, V. (1985) "Thermodynamical Approach to the Traveling Salesman Problem: An Efficient Simulation Algorithm", *J. Opt. Theory Appl.*, 45, 1, 41-51.

19. Charles–Owaba, O.E. (2002). " Set-Sequencing Algorithm to the Cyclic TSP". *Nigerian Journal of Engineering Management.* 3(2):47-64.

20. Charles–Owaba, O.E.(2001). "Optimality Conditions to the Accyclic Travelling Salesman Problem". *Operational Research Society of India*. 38:5.

21. Clarker, R.J. and Ryan, D.M. (1989). "Improving the Performance of an X-ray Diffractometer". Asia-Pacific Journal of Operational Research (Singapore). 6(2):107-130.

22. Crama,Y. ,Van de Klundert, J., and Spieksma, F. C.R. (2002). "Production Planning Problems in Printed Circuit Board Assembly". *Discrete Applied Mathematics*. 123:339-361.

23. Croes, G.A. 1958. "A Method for Solving Travelling-Salesman Problems". Operations Research. 6:791-812.

24. Crowder, H. and Padberg, M.W. (1980). "Solving Large-Scale Symmetric Travelling Salesman Problems to Optimality". Management Science. 26:495-509.

25. Fleurent .C, and Ferland .J.A, (1994). Genetic hybrids for the quadratic assignment problem, Quadratic Assignment and Related Problems, DIMACS Series on Discrete Mathematics and Theoretical Computer Science, Vol. 16, American Mathematical Society, Providence, RI, 1994, pp. 173–187.

26. Freisleben.B and Merz .P,(1996). A genetic local search algorithm for solving symmetric and asymmetric traveling salesman problems, in: Proceedings of the IEEE International Conference on Evolutionary Computation (ICEC'96), IEEE Press, Piscataway, USA, 1996, pp. 616–621.

27. Gambardella .L.M and Marco Dorigo,(1996). Solving symmetric and asymmetric TSPs by ant colonies, in: Proceedings of the IEEE International Conference on Evolutionary Computation(ICEC'96), IEEE Press, Piscataway, USA, 1996, pp. 622–627.

28. Johnson .D.S and McGeoch .L.M,(1997). The travelling salesman problem: a case study in local optimization, in: E.H.L. Aarts, J.K. Lenstra (Eds.), Local Search in Combinatorial Optimization, Wiley, Chichester, UK, 1997, pp. 215–310.

29. Jones .T, and Forrest .S,(1995). Fitness distance correlation as a measure of problem difficulty for genetic algorithms, in: L.J. Eshelman (Ed.), Proceedings of the Sixth International Conference on Genetic Algorithms, Morgan Kaufmann, San Francisco, CA, 1995, pp. 184–192.

30. Kirkpatrick .S ,(1985). Configuration space analysis of travelling salesman problems, J. de Physique 46 (1985) 1277–1292.

31. Maniezzo .V, Dorigo .M and Colorni .A,(1994). The ant system applied to the quadratic assignment problem, Technical Report IRIDIA/94-28, Université de Bruxelles, Belgium, 1994.

32. Marco Dorigo and L.M. Gambardella, (1997). Ant colony system: a cooperative learning approach to the traveling salesman problem, Vol 53-66.1997. IEEE Trans. Evolut. Comput

33. Marco Dorigo, Maniezzo .V and Colorni. A, (1991). Positive feedback as a search strategy, Technical Report 91-016, Dip. Elettronica, Politecnico di Milano, Italy, 1991.

34. Marco Dorigo, Maniezzo .V, and Colorni .A. (1992). The ant system: optimization by a colony of cooperating agents, IEEE Trans.

35. Martin .O ,(1991). Large-step Markov chains for the traveling salesman problem, Complex Systems 5 (1991) 299–326.

36. Stuuzle and Hoos,(2000). Ant-Q: A reinforcement learning approach to the traveling salesman problem, Proceedings of the 11t International Conference on Machine Learning, Morgan Kaufmann, San Francisco, CA, 1995, pp.252–260.

37. Vitruvius, "The Ten Books of Architecture," Dover Publications, 1960.

APPENDIX

Matlab Programme

```
function ACOtest(inputMatrix)
clc
%% START declare of own Variable for testing
global ASAdded
[row,col] = size(inputMatrix);
if row > col || col > row
end

ASAdded.inputMatrix = inputMatrix;
ASAdded.row = row;
ASAdded.col = col - 1;
ASAdded.MaxDist = max(max(inputMatrix)) * ASAdded.col;
Dimension = ASAdded.row;


NodeWeight = [];


disp(['AS start at ',datestr(now)]);
%%%%%%%%%%% the key parameters of Ant System %%%%%%%%

data_input = ASAdded.inputMatrix


MaxITime=1e3;
AntNum=Dimension; %depends on # of nodes
alpha=1;
beta=5;
rho=0.65;
%%%%%%%%%%%% the key parameters of Ant System %%%%%%%%
fprintf('Showing Iterative Best Solution:\n');


finalOutput = ...
AS(NodeWeight,AntNum,MaxITime,alpha,beta,rho);

disp(['AS stop at ',datestr(now)]);

function
[GBTour,GBLength,Option,IBRecord]=AS(WeightMatrix,AntNum,Max
ITime,alpha,beta,rho)
%% (Ant System) date:070427
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%
% Referenceï¿½ï¿½
```

```
% Dorigo M, Maniezzo Vittorio, Colorni Alberto.
% The Ant System: Optimization by a colony of cooperating
agents [J].
% IEEE Transactions on Systems, Man, and Cybernetics--Part
B,1996, 26(1)
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%
global ASOption Problem AntSystem ASAdded
ASOption = InitParameter(AntNum,alpha,beta,rho,MaxITime);
Problem = InitProblem(WeightMatrix);
AntSystem = InitAntSystem();

ITime = 0;
ASAdded.ITime = ITime;
IBRecord = [];

while 1
 InitStartPoint();
 for step = 2:ASOption.n
 for ant = 1:ASOption.m
  P = CaculateShiftProb(step,ant);
  nextnode = Roulette(P,1);
  RefreshTabu(step,ant,nextnode);
 end
 end

 ITime = ITime + 1;
 CaculateToursLength();

 GlobleRefreshPheromone();
 ANB = CaculateANB();
 [GBTour,GBLength,IBRecord(:,ITime)]                    =
GetResults(ITime,ANB);

 %=============================================
 deltaTau = (ASAdded.MaxDist)./(AntSystem.lengths);
 [deltaMax_val,deltaMax_indx] = max(deltaTau);
 ASAdded.deltaMax_tour  =  AntSystem.tours(deltaMax_indx,:);
%update InitStartPoint
 %=============================================


 if Terminate(ITime,ANB)
 Ant_Tour = AntSystem.tours
 format bank
 Ant_Tour_Delta = [AntSystem.tours, deltaTau]
 format short
 Distance_Covered_By_Ant = AntSystem.lengths
 [BestVal,BestIdx] = max(Ant_Tour_Delta(:,end));
 BestTour = AntSystem.tours(BestIdx,:)
 break;
```

```
 end

end
Option = ASOption;
%% ------------------------------------------------------------
-----
function                          ASOption                      =
InitParameter(AntNum,alpha,beta,rho,MaxITime)
global ASAdded
ASOption.n = ASAdded.row;
ASOption.m = AntNum;
ASOption.alpha = alpha;
ASOption.beta = beta;
ASOption.rho = rho;
ASOption.MaxITime = MaxITime;
ASOption.OptITime = 1;
ASOption.Q = 10;
ASOption.C = 100;
ASOption.lambda = 0.15;
ASOption.ANBmin = 2;
ASOption.GBLength = inf;
ASOption.GBTour = zeros(ASAdded.row,1);
ASOption.DispInterval = 10;
rand('state',sum(100*clock));
%% ------------------------------------------------------------
-----
function Problem = InitProblem(WeightMatrix)
global ASOption
n = ASOption.n;
MatrixTau = (ones(n,n)-eye(n,n))*ASOption.C;
Distances = WeightMatrix;
SymmetryFlag = false;
if isempty(WeightMatrix)
 Distances = CalculateDistance;
 SymmetryFlag = true;
end
Problem                                                       =
struct('dis',Distances,'tau',MatrixTau,'symmetry',SymmetryFl
ag);
%% ------------------------------------------------------------
-----
function AntSystem = InitAntSystem()
global ASOption
AntTours = zeros(ASOption.m,ASOption.n);
ToursLength = zeros(ASOption.m,1);
AntSystem = struct('tours',AntTours,'lengths',ToursLength);
%% ------------------------------------------------------------
-----
function InitStartPoint()
global AntSystem ASOption ASAdded
AntSystem.tours = zeros(ASOption.m,ASOption.n);
```

```
rand('state',sum(100*clock));
if ASAdded.ITime == 0
 AntSystem.tours(:,1) = randperm(ASAdded.row)';
else

 AntSystem.tours(:,1) = ASAdded.deltaMax_tour';
end
AntSystem.lengths = zeros(ASOption.m,1);
%% ------------------------------------------------------------
-----
function Probs = CaculateShiftProb(step_i, ant_k)
global AntSystem ASOption Problem
CurrentNode = AntSystem.tours(ant_k, step_i-1);
VisitedNodes = AntSystem.tours(ant_k, 1:step_i-1);
tau_i = Problem.tau(CurrentNode,:);
tau_i(1,VisitedNodes) = 0;
dis_i = Problem.dis(CurrentNode,:);
dis_i(1,CurrentNode) = 1;
Probs                                                          =
(tau_i.^ASOption.alpha).*((1./dis_i).^ASOption.beta);
if sum(Probs) ~= 0
 Probs = Probs/sum(Probs);
else
 NoVisitedNodes = setdiff(1:ASOption.n,VisitedNodes);
 Probs(1,NoVisitedNodes) = 1/length(NoVisitedNodes);
end
%% ------------------------------------------------------------
-----
function Select = Roulette(P,num)
m = length(P);
flag = (1-sum(P)<=1e-5);
Select = zeros(1,num);
rand('state',sum(100*clock));
r = rand(1,num);
for i=1:num
 sumP = 0;
 j = ceil(m*rand);
 while (sumP<r(i)) && flag
 sumP = sumP + P(mod(j-1,m)+1);
 j = j+1;
 end
 Select(i) = mod(j-2,m)+1;
end
%% ------------------------------------------------------------
-----
function RefreshTabu(step_i,ant_k,nextnode)
global AntSystem
AntSystem.tours(ant_k,step_i) = nextnode;
%% ------------------------------------------------------------
-----
function CaculateToursLength()
```

```
global ASOption AntSystem
x = CalculateDistance;
p = AntSystem.tours;

Lengths = zeros(ASOption.m,1);

for j=1:ASOption.n
 pRow = p(j,:);
 sumRow = 0;
 for i=1:ASOption.n-1
 sumRow = sumRow + x(pRow(i),pRow(i+1));
 end
 Lengths(j) = sumRow;
end

AntSystem.lengths = Lengths;
%% ------------------------------------------------------------
-----
function [GBTour,GBLength,Record] = GetResults(ITime,ANB)
global AntSystem ASOption
[IBLength,AntIndex] = min(AntSystem.lengths);
IBTour = AntSystem.tours(AntIndex,:);
if IBLength<=ASOption.GBLength
     ASOption.GBLength = IBLength;
     ASOption.GBTour = IBTour;
     ASOption.OptITime = ITime;
end
GBTour = ASOption.GBTour';
GBLength = ASOption.GBLength;
Record = [IBLength,ANB,IBTour]';
%% ------------------------------------------------------------
-----
function GlobleRefreshPheromone()
global AntSystem ASOption Problem
AT = AntSystem.tours;
TL = AntSystem.lengths;
sumdtau=zeros(ASOption.n,ASOption.n);
for k=1:ASOption.m
 for i=1:ASOption.n

sumdtau(AT(k,i),AT(k,i))=sumdtau(AT(k,i),AT(k,i))+ASOption.Q
/TL(k);
 if Problem.symmetry
  sumdtau(AT(k,i),AT(k,i))=sumdtau(AT(k,i),AT(k,i));
 end
 end
end
Problem.tau=Problem.tau*(1-ASOption.rho)+sumdtau;
%% ------------------------------------------------------------
-----
function flag = Terminate(ITime,ANB)
```

```
global ASOption
flag = false;
if ANB<=ASOption.ANBmin || ITime>=ASOption.MaxITime
 flag = true;
end
%% ------------------------------------------------------------
-----
function ANB = CaculateANB()
global ASOption Problem
mintau                                                          =
min(Problem.tau+ASOption.C*eye(ASOption.n,ASOption.n));
sigma = max(Problem.tau) - mintau;
dis                =                Problem.tau                -
repmat(sigma*ASOption.lambda+mintau,ASOption.n,1);
NB = sum(dis>=0,1);
ANB = sum(NB)/ASOption.n;
%% ------------------------------------------------------------
-----
function Distances = CalculateDistance
global ASAdded
Distances = ASAdded.inputMatrix;
%% ------------------------------------------------------------
-----
```