# OPTIMAL CAMPAIGN VISITATION OF PRESIDENTIAL ASPIRANTS

## *CASE STUDY: CENTRAL REGION OF GHANA.*

**BY**

**ENNIN FRANCIS CORNELIUS**



**A thesis submitted to the Department of Mathematics Kwame Nkrumah University of Science and Technology Kumasi, Ghana in partial fulfilment of the requirements for the degree of MASTER OF PHILOSOPHY IN MATHEMATICS**

**College of Science**

**Department of Mathematics**

**June, 2013**

## DECLARATION

I hereby declare that I am the sole author of this thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners. I understand that my thesis may be made available to the public. All contributions and assistance made to the research have been dully acknowledged.



Ennin Francis Cornelius

(PG6201211)

| …… | …….. | …… |
|---|---|---|
| Student's name and ID | Signature | Date |

Prof. S.K Amponsah

| …… | …….. | …… |
|---|---|---|
| Supervisor's name | Signature | Date |

Prof. S.K Amponsah

| …… | …….. | …… |
|---|---|---|
| Name of Head of Department | Signature | Date |

## DEDICATION

I dedicate this work to my beloved mother, Mrs. Paulina Ennin and my only sister, Sarah Ennin.



## ACKNOWLEDGEMENT

To the Almighty God and Father be praise, Glory and Honour forever and ever. It is his bountiful endowment of grace through Christ Jesus that kept me through.

I would also be eternally grateful to my Supervisor, Prof. S.K. Amponsah for the words and acts of encouragement, useful suggestions and support.

I will also like to acknowledge my mother, Mrs. Paulina Ennin, Uncle Sam and Madam Naomi Antwi for their inspiration and encouragement.

## ABSTRACT

All over the world, political parties are seen as vital institutions for contemporary democratic dispensation and they play a vital role in the democratic process. Finance is regarded as the most essential resource for political parties (van Biezen, 2003). Yet for too long, commitment to financing of political parties in Ghana has remained rhetoric hence the need to minimize cost. This research work provides a solution to the problem of presidential aspirants having to tour all the twenty-three (23) constituency capitals in the Central Region of Ghana campaigning. Most of the cost incurred by presidential candidates as they visit all the constituency capitals in Central Region is as a result of transportation and minimizing the distances covered in such trips goes a long way to minimize cost, since transportation costs depend on distances traveled. This problem is formulated as a Travelling Salesman Problem (TSP). TSP involves finding an optimal route for visiting cities and returning to the point of origin. In the development of the algorithm, real road lengths were used instead of the norm-1 distances which are widely accepted for the solution of the TSP using Simulated Annealing. The formulation of the TSP in this work is based on Symmetric TSP. This research work presents the solution based on Simulated Annealing (SA) method. A Mat lab code for the TSP algorithm was used to solve the problem of a presidential aspirant visiting all the twenty-three (23) constituency capitals in the Central Region of Ghana campaigning. The result obtained in the study showed that the optimal route that can be considered is Elmina→ Essarkyir→ Apam→ Winneba→ Potsin→ Awutu Breku→ Kasoa→ Agona Swedru→ Agona Nsaba→ Afransi→ Asikuma→ Ajumako→ Saltpond→ Assin Foso→ Dunkwa-on-Offin→ Diaso→ Twifo Praso→ Jukwa→ Assin Breku→ Nsuaem Kyekyewere→ Abura Dunkwa→ Abura(Cape Coast) → Old Hospital Hill(Cape Coast) →Elmina with a total distance of 786km.

## TABLE OF CONTENTS

## LIST OF TABLES

## LIST OF FIGURES

# CHAPTER 1

# INTRODUCTION

## 1.0 Brief history of Ghana

Ghana became the first country in Africa south of the Sahara to gain independence from colonial rule in March 6, 1957. The total land area of Ghana is 238,538 square kilometres; the distance from the south to the north being 840 kilometres and 554 kilometres from east to west. Ghana is bordered on the east by Togo, West by Cote d'ivoire, North by Burkina Faso and South, the Gulf of Guinea. It has a population of 24,658,823 million (2010 census). Ghana has ten regions namely Central, Ashanti, Western, Eastern, Brong Ahafo, Northern, Upper West, Upper East, Volta and Greater Accra region.



## 1.1 Background of the study

The Central Region is one of the five regions of Akanland and the smallest region in Akanland, and the eight smallest in Ghana. Central Region is bordered by Ashanti and Eastern region to it's north, Western region to it's west, Eastern region to it's east, and to it's south by the Atlantic Ocean. Central Region is the home of Akans who are its natives, and are 99% of the Central Region's population. The main means of transport is by road and the road network is evenly spread. There is only one airport in the region. This means that it would not be possible to travel from one place of the region to another by air.

**Central Region**


**Figure 1.1: Location of Central Region**

The capital of Central Region is Cape Coast. Cape Coast is the third most populous settlement in Akanland, in terms of population, with a population of 217,032 people (2012 census). Cape Coast is linked to Accra, the nation's capital by a first class road and is about one hour and 30 minutes drive between them at a relatively regular pace.


**Figure 1.2: A section of Cape Coast town**

Central Region covers an area of 9,826 square kilometers and has a population of 2,201,863 representing 8.9 per cent of the entire Ghana's population (2010 census).

The region has twenty three (23) constituencies out of the two hundred and thirty constituencies in Ghana. The voter population in the region is 1,240,439 according to the 2012 biometric voter's register.

The twenty three (23) constituencies, their capitals and their respective voter population according to the electoral commission are tabulated below.

**Table 1.1: Constituencies, their capitals and voter population in Central Region.**

| Contituencies | Contituency capital | Voter population |
|---|---|---|
| Komenda/Edina/Eguafo/Abrem | Elmina | 77,789 |
| Cape Coast South | Old Hospital Hill, Cape Coast | 51,119 |
| Cape Coast North | Abura | 55,115 |
| Abura/Asebu/Kwamankese | Abura-Dunkwa | 59,841 |
| Mfantseman | Saltpond | 80,235 |
| Ekumfi | Essarkyir | 33,557 |
| Ajumako/ Enyan/ Esiam | Ajumako | 60,784 |
| Gomoa West | Apam | 67,117 |
| Gomoa East | Potsin | 48,690 |
| Gomoa Central | Afransi | 36,843 |
| Effutu | Winneba | 47,582 |
| Awutu /Senya West | Awutu Breku | 63,472 |
| Awutu Senya East | Kasoa | 82,223 |
| Agona West | Agona Swedru | 73,188 |
| Agona East | Agona Nsaba | 50,583 |
| Asikuma/Odoben/Brakwa | Asikuma | 59,523 |
| Assin North | Assin Breku | 36,022 |
| Assin Central | Assin Foso | 37,982 |
| Assin South | Nsuaem Kyekyewere | 49,578 |
| Twifo Atti Morkwaa | Twifo Praso | 50,638 |
| Hemang/ Lower /Denkyira | Jukwa | 33,576 |
| Upper Denkyira East | Dunkwa-On-Offin | 50,154 |
| Upper Denkyira West | Diaso | 34,828 |

The total number of registered voters in Central Region as of 2012 presidential elections was 1,240,439. Out of the twenty- three (23) constituencies, the NDC has eighteen (18) members of parliament and the NPP have eight (5)

The region also has twenty districts.

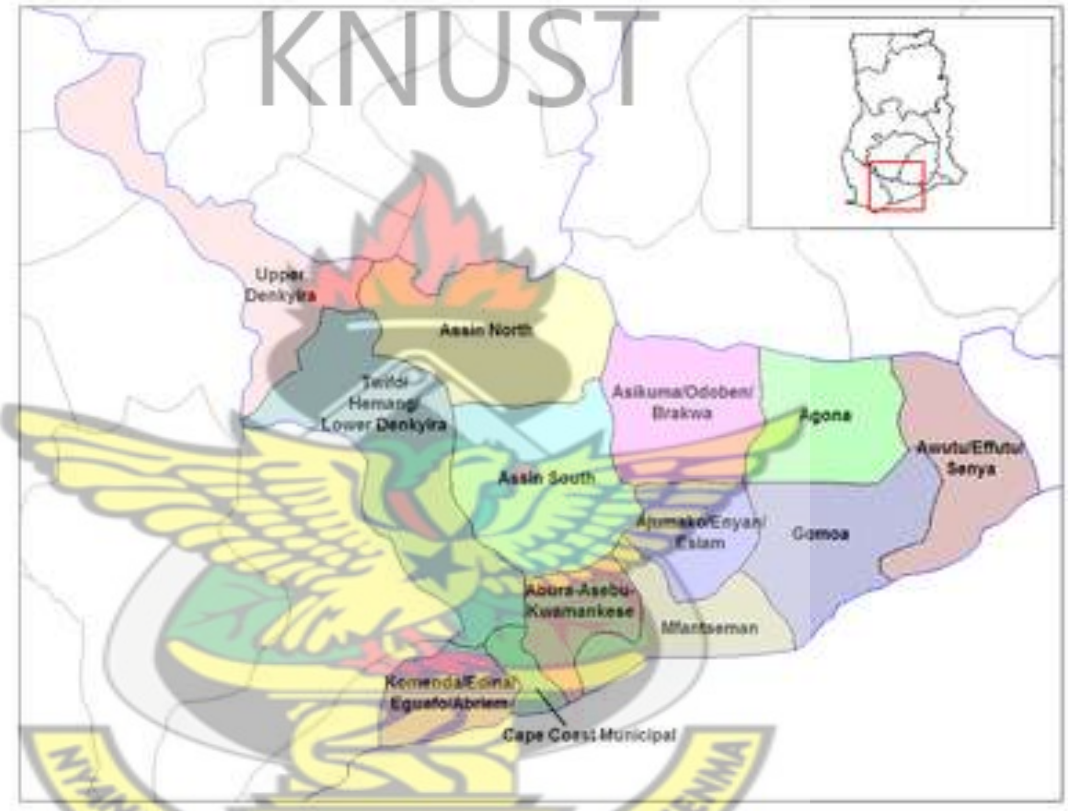


**Figure 1.3: Districts in Central Region**

## 1.2 Statement of the study

Democracy, which is a form of government in which all eligible citizens have an equal say in the decisions that affect their lives is very crucial to every nation's development. All over the world, political parties are seen as vital institutions for contemporary

democratic dispensation and they play a vital role in the democratic process. They are crucial for the organization of modern democracy and are relevant for the expression and manifestation of political pluralism. The strategic role that parties play in any democratic nation means that funding their activities cannot be discounted. Finance is regarded as the most essential resource for political parties (van Biezen, 2003). Yet for too long, commitment to financing of political parties in Ghana has remained rhetoric.

In many ways, political activities involve expenses which should be seen as the necessary and unavoidable cost of democracy. Because money is one of the most essential resources for political parties, which are principal actors of modern democracy, it plays a critical role in the democratic process (van Biezen, 2003). In order for political parties to maintain their party organizations, employ party personnel, conduct election campaigns and communicate with electorates at large, appropriate financial resources need to be available. These leave political parties with no option than to fall on multinational companies or rich individuals to finance their election campaign activities. Allowing multinational companies or rich individuals to finance election campaign is not the best since the financiers would have to be compensated. This could lead to political corruption. It is therefore a necessity and very prudent for political parties to find ways of minimizing cost.

The presidential aspirants' taking the optimal route in all of his /her campaign visitation is one surest way of minimizing cost. This is what this thesis seeks to achieve or do.

### 1.3 Objectives of the study

The aim of this research work is therefore to minimize the cost of visitation of the aspirants to all the twenty three constituencies in the central region by minimizing the route length for their visits using simulated annealing.

The objectives of this research work are;

(i) to formulate a mathematical model that takes into account the actual distances that are physically traversed by the campaign teams.

(ii) to determine the optimal route for the visitation based on the physical distances and actual links by the use of the simulated annealing method for central region of Ghana.

(iii) to enable presidential aspirants of the various political parties who visit the constituency capitals in the region to campaign cut down cost.

### 1.4 Justification

The major component of the cost incurred by the presidential campaign team is attributed to the cost of transportation between constituencies. The need to reduce the cost of campaign requires a critical look at minimizing the cost of transportation.

### 1.5 Methodology

The campaign visitation of presidential aspirants to the constituency capitals in the Central region will be modeled as a Travelling Salesman Problem (TSP). The Simulated Annealing Algorithm method which is a meta-heuristic based search algorithm will be

used to solve the TSP model. This is because the Simulated Annealing is capable of solving combinatorial optimization problems like the Travelling Salesman Problem (TSP).

The sources of data are the internet and libraries for the literature review, the electoral commission outfit for voter population in the region and the Department of Feeder Road, Cape Coast for inter-constituency capital distances.



### 1.6 Scope of the study

The study covers the entire distances between the twenty- three constituency capitals in Central Region of Ghana.

### 1.7 Limitations of the study

Out of over three hundred towns and villages in Central Region, only twenty- three towns which also happen to be constituency capitals were selected for the study due to time and money.

### 1.8 Organization of the study

This thesis is a result of a research carried out on the tour of an aspirant for a party flag bearer across the central region of Ghana with the specific application of the simulated annealing heuristic method. The thesis is organized into five chapters as follows:

Chapter 1 discusses the introduction: Brief history of Ghana background to the study, problem statement, problem definition, aims and objectives, methodology, justification, importance of presidential aspirants visit and the organization of the study.

Chapter 2 will deal with review of related literature on the area of study.

Chapter 3 will contain the research methodology.

Chapter 4 will present data collection, analysis of data, findings and discussion.

Chapter 5 contains the summary, conclusions and recommendations on the findings of the study.



### 1.9 Summary

In this chapter, we presented brief history of Ghana, background of the study, statement of the study, aims and objectives of the study, justification of the study, methodology, scope of the study, limitations of the study, organization of the study, definition of the problem and importance of a visit by a presidential aspirant.

In the next chapter, we shall review some related literature in the field of a travelling salesman problem.

# CHAPTER 2

# LITERATURE REVIEW

## 2.0 Introduction

In this chapter, we shall present the related literature review

The Travelling Salesman Problem (TSP) is an NP-hard problem in combinatorial optimization studied in operations research and theoretical computer science. The Traveling Salesman Problem (TSP) is one of the most widely studied combinatorial optimization problems. Its statement is deceptively simple, and yet it remains one of the most challenging problems in Operational Research. Hundreds of articles have been written on the TSP.

The book edited by Lawler and Wood (1985) provided an insightful and comprehensive survey of all major research results until date. Given a list of cities and their pairwise distances, the task is to find the shortest possible route that visits each city exactly once.

In the Travelling Salesman Problem (TSP), we are given $n$ nodes and for each pair $\{i, j\}$ of distinct nodes, a distance $d_{i,j}$. We desire a closed path that visits each node exactly once (that is a salesman tour) and incurs the least cost, which is the sum of the distances along the path. In the m-TSP problem, m-number of salesmen have to cover the given cities and each city must be visited by exactly one salesman. Every salesman starts from the same city, called depot, and must return at the end of his journey to this city again. The TSP has been a classic problem, which has influenced the emergence of fields such as operations research.

Since the 1970s, mounting evidence from complexity theory suggests that the problem is computationally difficult. Exact optimization is NP-hard (Karp, 1990).

The Travelling Salesman Problem (TSP) was first formulated as a mathematical problem in 1930 and is one of the most intensively studied problems in optimization. It is used as a benchmark for many optimization methods. Even though the problem is computationally difficult, a large number of heuristics and exact methods are known, so that some instances with tens of thousands of cities can be solved.



The TSP is one of the major success stories for optimization. Decades of research into optimization techniques, combined with the continuing rapid growth in computer speeds and memory capacities, have led to one new record after another. Over the past 33 years, the record for the largest nontrivial TSP instance solved optimality has increased from 318 cities (Crowder and Padberg, 1980) to 2392 cities (Padberg and Rinaldi, 1987) to 7397 cities (Applegate, Bixby, Chvatal, and Cook, 1994). Admittedly, this last result took roughly 3-4 years of CPU- time on a network of machines of the calibre of a SPARCstation 2. (SPARC is a trademark of SPARC international, Inc. and is licensed exclusively to Sun Microsystems, Inc.). However, the branch-and-cut technology developed for these record-setting performances has also had a major impact on the low end of the scale. Problems with 100 or fewer cities are now routinely solved within a few minutes on a workstation (although there are isolated instances in this range that take much longer) and instances in the 1, 000-city range typically take only a few hours (or days).

The Travelling Salesman Problem (TSP) has been studied for the past fifty-two (52) years and many exact and heuristic algorithms have been developed. These algorithms include construction algorithms, iterative improvement algorithms, branch- and -bound (B&B) and branch- and -cut (B&C) exact algorithms and many meta-heuristic algorithms, such as simulated annealing (SA), Tabu Search (TS), Ant Colony (AC) and Genetic Algorithm (GA).



The method of simulated annealing quickly gives a sub-optional answer, where 'sub-optional' means solution close enough to be the true minimum path. A number of results were developed by Chandra and Tovey (1999) some worst-case and some probabilistic, on the performance of 2- and k-opt local search for the TSP, with respect to both the quality of the solution and the speed with which it is obtained. Indeed, subsequent experiments suggest that when $N > 100$ the elastic net approach does not even outperform 2-opt, which typically gets within 5% of the Held-Karp bound for random Euclidean instances. Perterson (1990) compared the elastic net approach to various others on 200-city instances and obtained results that averaged 6% worse than those found by a version of the Mühlenbein et al., (1988) genetic algorithm. Vakhutinsky and Golden (1995) used a hierarchical scheme to speed up the process ( and slightly improve tour quality) and were able to handle 500-city random Euclidean instances, but they appear to have found tours that were more than 10% above the Held-Karp bound.

Boeres, de Carvalho, and Barbosa (1992) sped up the elastic net approach in much the same way that Bonomi and Lutton (1984) sped up simulated annealing, using a partition of the unit square into a grid of smaller cells to help narrow their searches. In particular, when computing the force imposed on a vertex by the cities, they restricted attention to

just those cities in nearby cells. This enabled them to 1000-city random Euclidean instances, and they claim that there is no significant loss in tour quality. Their tours, however, are 5-10% worse than those found by their implementation of Lin-Kernighan, and they are hence worse than what 2-opt could have provided, given that 2-opt typically only 3% or so behind Lin-Kernighan on random Euclidean instances. Moreover, 2-opt and even Lin-Kernighan would have been substantially faster: for 1000 cities the Boeres et al., (1992) running time is roughly 15, 000 seconds on a SPARCstation, versus an average of less than a second for Lin-Kernighan on our SGI Challenge, a machine that is at most 15 times faster. In their paper, Boeres et al.,(1992) claim significant speed advantages for elastic nets over Lin-Kernighan, but this comparison is based on an implementation of the latter that appears to have a worse than quadratic running time growth rate, as opposed to the decidedly subquadratic running time reported for Lin-Kernighan. An implementation of the Lin-Kernighan heuristic, one of the most effective methods for giving optimal or near optimal solutions for the symmetric TSP is described (Helsgaun,2000) , (Gambardella and Dorigo, 1996). Computational tests show that the implementation is highly valuable. A simulated annealing process starts with the cities connected in a random order, and then considers making random changes in that order. If changing the order of cities leads to a shorter path, we accept that change. If the modification yields a longer path, we give ourselves a certain probability of accepting the modification. The probability is gradually reduced over time in order to rule out shorter and shorter path increase thereby converging toward a path length close to the absolute minimum.

Belisle et al., (1993) discussed convergence properties of simulated annealing algorithms applied to continuous functions and applies these results to hit-and-run algorithms used in global optimization. His convergence properties are consistent with those presented by Hajek(1988) and he provides a good contrast between convergence in probability and the stronger almost sure convergence.

Zabinsky et al., (1993) extended this work to an improved hit-and-run algorithm used for global optimization. Fleischer and Jacobson (1996) proposes cybernetic optimization by simulated annealing as a method of parallel processing that accelerated the convergence of simulated annealing to the global optima.

Fleischer (1999) extended the theory of cybernetic optimization by simulated annealing into the continuous domain by applying probabilistic feedback control to the generation of candidate solutions. The probabilistic feedback control method of generating candidate solutions effectively accelerates convergence to a global optimum using parallel simulated annealing on continuous variable problems.

Locatelli (1996) presented convergence properties for a class of simulated annealing algorithms for continuous global optimization by removing the restriction that the next candidate point must be generated according to a probability distribution whose support is the whole feasible set.

Siarry et al., (1997) studied simulated annealing algorithms for globally minimizing functions of many- continuous variables. Their work focuses on how high-dimensionality can be addressed using variables discretization, as well as considering the design and implementation of several complementary stopping criteria. Yang (2000)

and Locatelli (2000) also provided convergence results and criteria for simulated annealing applied to continuous global optimization problems. Kiatsupaibul and Smith (2000) introduced a general purpose simulated annealing algorithm to solve mixed integer linear programmes.

Aarts and Korst (1989), in Boltzmann machine implementation, used extra neutrons to insure that locally optimal solutions correspond to connected graphs. Joppe, Cardon and Bioche (1990) propose something similar, with a second level of neutrons that can trigger changes to the external inputs for the first level. Neither approach seems to have been tried on instances with more than 30 cities, however. A more effective (if less neutral) approach seems to be to invoke the classical patching algorithms of Karp (1977). This was proposed and implemented by Xu and Tsai (1991), handled up to 150-city instances. Before switching to Karp's patching heuristic, Xu and Tsai perform five successive neutral net runs, each adding penalty terms to the energy function so as to inhibit the creation of the subtours seen in the previous rounds.

Some of the well known tour construction procedures are the nearest neighbour procedure by Rosenkratz et al., (1977), the Clark and Wright savings' algorithm, the insertion procedures, the partitioning approach by Karp and the minimal spanning tree approach by Christofides.

The branch exchange is perhaps the best known iterative improvement algorithm for the TSP. The 2-opt and 3-opt heuristics were described in Lin. Lin and Kernighan (1973) made a great improvement in quality of tours that can be obtained by heuristic methods. Even today, their algorithm remains the key ingredient in the successful approaches for

finding high quality tours and is widely used to generate initial solutions for other algorithms or developed a simplified edge exchange procedure requiring only $O(n^2)$ operations at each step, but producing tour nearly as good as the average performance of 3-opt algorithm. Determining a minimum-cost arborescence rooted at vertex $r$, and finding the minimum-cost arc entering vertex $r$. The first problem is easily solved in $O(n^2)$ time (Tarjan, 1977). This relaxation can be used in conjunction with Lagrangian relaxation. However, on asymmetric problems, the Assignment Problem (AP) relaxation would appear empirically superior to the r- arborescence relaxation (Balas and Toth, 1977).

Christofides (1970) and Held and Karp (1971) were among the first to propose a TSP algorithm based on this relaxation. Improvements and refinements were later suggested by Helbig Hansen and Krarup (1974), Smith and Thompson (1977), Volgenant and Jonger (1982), Gavish and Srikanth (1986), and Carpaneto, Fischetti and Toth (1989).

One of the earliest exact algorithms is by Dantzig et al., (1954), in which Linear Programming (LP) relaxation is used to solve the integer formulation by suitably choosing linear inequality to the list of constraints continuously. Branch and bound (B & B) algorithm are widely used to solve the TSP's. Several authors have proposed B & B algorithm based on assignment problem (AP) relaxation of the original TSP formulation. These authors include Eastrnan (1958), Held and Karp (1970), Smith et al.,(1977), Carpaneto and Toth (1980), Balas and Christofides (1981). Some Branch and Cut (B & C) based exact algorithms were developed by Crowder and Padberg (1980), Padberg and Hong (1980), Grötschel and Holland (1991). The lower- bounding procedure described by Fischetti and Toth was embedded within the Carpaneto and Toth (1980)

branch-and-boundalgorthm on a variety of randomly generated problems and on some problems described in the literature. The success of the algorithm depends on the type of problem considered. For the easiest problem type, the authors report having solved 2000- vertex problems in an average time of 8329 seconds on an Hp 9000/840 computer.



Apart from the exact and heuristic algorithms stated above, meta-heuristic algorithms have been applied successfully to the TSP by a number of researchers. Simulated Annealing (SA) algorithms for the TSP were developed by Bonomi and Lutton, Golden and Skiscim( 1986) and Nahar et al., (1989) etc. Tabu search meta-heuristic algorithms for TSP have been proposed by Knox and Fiechter (1994). The Ant Colony (AC) is a relative new meta-heuristic algorithm which is applied successfully to solve the TSP, some work based on SA technology was reported by Dorigo et al., (1996). Genetic algorithms for the TSP were reported by Goefenstetle et al., (1985).

Applegate et. al., (1994) solved a travelling salesman problem which models the production of printed circuit boards having 7, 397 holes (cities), and in 1998, the same authors solved a problem over the 13, 509 largest cities in the U.S. For problems with large number of nodes as cities the TSP becomes more difficult to solve.

In Homer's Ulysses (1993) problem of a 16 city travelling salesman problem, one finds that there are 653, 837, 184, 000 distinct routes (Gr ö tschel and Padberg, 1993). Enumerating all such roundtrips to find the shortest one took 92 hours on a powerful workstation.

The TSP and its solution procedures have continued to provide useful test grounds for many combinatorial optimization approaches. Classical local optimization techniques Rossman (1958); Applegate et al., (1999); Riera-Ledesma, (2005); Walshaw, (2002); Walshaw (2001) as well as many of the more recent variants on local optimization, such as simulated annealing by Tian and Yang (1993), tabu search by Kolohan and Liang, (2003), neural networks by Potvin, (1996) and genetic algorithms have all been applied to this problem, which for decades has continued to attract the interests of researchers.



Although a problem statement posed by Karl Menger on February 5, 1930, at a mathematical colloquium in Vienna, is regarded as a precursor of the TSP, it was Hassle Whitney, in 1934, who posed the travelling salesman problem in a seminar at Princeton University (Flood, 1956).

In 1949, Robinson, with an algorithm for solving a variant of the assignment problem is one of the earliest references that use the term "travelling salesman problem" in the context of mathematical optimization. ( Robinson, 1949) , however, a breakthrough in solution methods for the TSP came in 1954, when Dantzig et al., (1954) applied the simplex method (designed by George Dantzig in 1947) to an instance with 49 cities by solving the TSP with linear programming.

There were several recorded contributions to the TSP in 1955. Heller, (1955) discussed linear systems for the TSP polytope, and some neighbour relations for the asymmetric TSP polytope. Also Kuhn, (1955) announced a complete description of the 5-city asymmetric TSP polytope.

Morton and Land (1955) presented a linear programming approach to the TSP, alongside the capacitated vehicle routing problem. Furthermore, Robacker (1955)

reported manual computational tests of some 9 cities instance using the Dantzig-Fulkerson-Johnson method, with average computational times of about 3 hours. This time became the benchmark for the next few years of computational work on the TSP (Robacker, 1955).

Flood (1956) discussed some heuristic methods for obtaining good tours, including the nearest-neighbour algorithm and 2-opt while Kruskal, (1956) drew attention to the similarity between the TSP and the minimum-length spanning tress problem. The year 1957 was a quiet one with a contribution from Barachet, (1957) described an enumeration scheme for computing near-optimal tours.

Croes (1958) proposed a variant of 3-opt together with an enumeration scheme for computing an optimal tour. He solved the Dantzig-Fulkerson-Johnson 49-city example in 70 hours by hand. He also solved several of the Robacker examples in an average time of 25 minutes per example. Bock (1958) describes a 3-opt algorithm together with an enumeration scheme for computing an optimal tour. The author tested his algorithm on some 10-city instance using an IBM 650 computer.

By 1958, work related to the TSP had become serious research to attract Ph.D. students. A notable work was a Ph.D. thesis Eastman, (1958) where a branch-and-bound algorithm using the assignment problem to obtain lower bounds was described. The algorithm was tested on examples having up to 10 cities. Also that same year, Rossman and Twery (1958) solved a 13-city instance using an implicit enumeration while a step-by-step application of the Dantzig-Fulkerson-Johnson algorithm was also given for

Barachet's 10-city example. Bellman (1960) showed the TSP as a combinatorial problem that can be solved by dynamic programming method.

Miller et al., (1960) presented an integer programming formulation of the TSP and its computational results of solving several small problems using Gomory's cutting-plane algorithm was reported. Lambert (1960) solved a 5-city example of the TSP using Gomory cutting planes. Dacey (1960) reported a heuristic, whose solutions were on average 4.8 per cent longer than the optimal solutions. TSP in 1960 achieved national prominence in the United States of America when Procter and Gamble used it as the basis of a promotional context. Prizes up to $10,000.00 were offered for identifying the most correct links in a particular 33-city problem. A TSP researcher, Gerald Thompson of Carnegie Mellon University won the prize in Applegate et al., (2007).

Muller-Merbach (1961) proposed an algorithm for the asymmetric TSP, the dynamic programming approach gained attention. Gonzales solved instances with up to 10 cities using dynamic programming on an IBM 1620 computer by Gonzales , (1962). Similarly, Held and Karp (1962) described a dynamic programming algorithm for solving small instances and for finding approximate solutions to larger instances.

Little et al,. (1963) coined the term branch-and-bound. Their algorithm was implemented on an IBM 7090 computer and they gave some interesting computational tests including the solution of a 25-city problem that was in the Held and Karp test set. Their most cited success is the solution of a set of 30-city asymmetric TSP'S having random edge lengths. In an important paper (Lin, 1965): a heuristic method for the TSP was published.

The author defined k-optional tours, and gave an efficient way to implement 3-opt, extending the work of Croes (1958) with computational results given for instances with up to 105 cities.

The year 1966 was another fruitful one for the TSP in terms of published works. Roberts and Flores (1966) described an enumerative heuristic and obtained a tour for Karg and Thompson's 57-city example, having cost equal to the best tour found by Karg and Thompson. Also, in a D.Sc. thesis at Washington University, St. Louis, Shapiro (1966) describes an algorithm similar to Eastman's branch-and-bound algorithm.

Gomory (1966) gave a very nice description of the methods contained in Dantzig et al. (1954), Held and Karp (1962) and Little et al. (1963). Similarly, in Lawler and Wood (1966) descriptions of the branch-and-bound algorithms of Eastman 1958 and Little et al. (1963) were given. The authors suggested the use of minimum spanning tress as a lower bound in a branch-and-bound algorithm for the TSP.

Bellmore and Nemhauser (1968) presented an extensive survey of algorithms for the TSP. They suggested dynamic programming for TSP problems with 13 cities or less, Shapiro's branch-and-bound algorithm for larger problems up to about 70-100 and Shen Lin's '3-opt' algorithm for problems that cannot be handled by Shapiro's algorithm. Raymond (1969) is an extension to Karg and Thompson's 1964 heuristic for the TSP where computational results were reported for instances having up to 57 cities.

Held and Karp (1970) introduced the 1-tree relaxation of the TSP and the idea of using node weights to improve the bound given by the optimal 1-tree. Their computational results were easily the best reported up to that time. Another notable work on the TSP in

the 70s is the S. Hong, Ph.D. Thesis, at The Johns Hopkins University in 1972 written under the supervision of Bellmore, and the work was the most significant computational contribution to the linear programming approach to the TSP since the original paper of Dantzig et al., (1959).

The Hong's (1972) algorithm had most of the ingredients of the current generation of linear-programming based algorithms for the TSP. He used a dual LP algorithm for solving the linear-programming relaxations; he also used the Ford-Fulkerson max-flow algorithm to find violated subtour inequalities.

The algorithm of Held and Karp (1971) was the basis of some major publications in 1974. In one case, Hansen and krarup (1974) tested their version of Held-Karp (1971) on the 57-city instance of Karg and Thompson 1964 and a set of instances having random edge lengths. In 1976 a linear programming package written by Land and Powell was used to implement a branch-and-cut algorithm using subtor inequalities. Computational results for the 48-city instance of Held and Karp and the 57-city instance of Karg and Thompson (1964) were given.

Smith and Thompson (1977) presented some improvements to the Held-Karp algorithm tested their methods on examples which included the 57-city instance of Karg and Thompson 1964 and a set of ten 60-city random Euclean instances. In 1979, Land described a cutting-plane algorithm for the TSP. The decade ended with a survey on algorithms for the TSP and the asymmetric TSP in Buckard, (1979).

Crowder and Padberg (1980) gave the solution of a 318-city instance described in Lin and Kernighan (1973). The 318-city instance remained until 1987 as the largest TSP solved. Also, in 1980, Grötschel gave the solution of a 120-city instance by means of a

cutting-plane algorithm, where subtour inequalities were detected and added by hand to the linear programming relaxation in Grötschel (1980). In 1982, Volgenant and Jonker described a variation of the Held-Karp algorithm, together with computational results for a number of small instances by Volgenant and Jonker (1982). A very important work of 1985 is a book (Lawler et al., 1985) containing several articles on different aspects of the TSP as an optimization problem. Padberg and Rinaldi (1987) solved a 532-city problem using the so-called branch and cut method.



The approach for handling the subtours elimination constraints of the TSP integer LP is another area for re-examination. Researchers have identified the issue of feasibility or subtour elimination as very crucial in the formulation of the TSP or similar permutation sequence problem. "No one has any difficulty understanding subtours, but constraints to prevent them are less obvious "Radin, (1998).

Methodologies or theoretical basis for handling these constraints within the context of algorithm development has been the basis of many popular works on the TSP. A classical example of this approach is in Crowder and Padberg (1980) where a linear programming relaxation was adopted such that if the integral solution found by this search is not a tour, then the subtour inequalities violated by the solution are added to the relaxation and resolved.

Gr ö tschel (1980) used a cutting-plane algorithm, where cuts involving subtour inequalities were detected and added by hand to the linear programming relaxation. Hong (1972) used a dual LP algorithm for solving the linear-programming relaxations, the Ford –Fulkerson max-flow algorithm, for finding violated subtour inequalities and a

branch-and-bound scheme, which includes the addition of subtour inequalities at the nodes of the branch-and-bound tree. Such algorithms are now known as "branch-and-cut". The problem of dealing with subtour occurrences algorithm development has been a major one in the TSP studies in the literature.

The works in the 1990's were mostly application in nature. A large number of scientific/engineering problems and applications such as vehicle routing, parts manufacturing and assembly, electronic board manufacturing, space exploration, oil exploration, and production job scheduling, etc. have been modeled as the Machine Setup Problem (MSP) or some variant of the TSP are found in (AL-Haboub-Mohamad and Selim Shokrik (1993), Clarker and Ryan (1989), Keuthen (2003), Kolohan and Liang (2000), Mitrovic-Minic and Krishnamurti, (2006)).

One of the ultimate goals in computer science is to find computationally feasible exact solutions to all the known NP-Hard problems; a goal that may never be reached. Feasible exact solutions for the TSP have been found, but there are restrictions on the input sizes. An exact solution was found for a 318-City problem by Crowder and Padberg in (1980).

The basic idea in achieving this solution involves three phases. In the first phase, a true lower bound on the optimal tour is found. In the second phase, the result in the first phase is used to eliminate about ninety-seven per cent of all the possible tours. Thus, only about three per cent of the possible tours need to be considered. In the third phase, the reduced problem is solved by brute force. This solution has been implemented and used in practice. Experimental results by Apple Gate et al.,(1998) showed that running

this algorithm, implemented in the C programming language and executed on a 400MHz machine, would produce a result in 24.6 seconds of running time.

Other exact solutions have been found. As mention in 1998, a 120-city problem by Grotschel (1980), a 532-city problem by Padberg and Rinaldi (1987). However, none of the algorithms that provide an exact solution for input instances of over a thousand cities are practical for everyday use. Even with today's super computers, the execution time of such exact solution algorithms for TSPs involving thousands of cities could take days.

Computer hardware researchers have been making astonishing progress in manufacturing evermore powerful computer chips. Moores Law in (http://en. Wikipedia.org/wiki/Moore's_law), which states that the number of transistors that can fit on a chip will double after every 18 months, has held ground since 1965. This basically means that computing power has doubled every 18 months since then. Thus, we have been able to solve larger instances of NP-hard problems, but algorithm complexity has still remained exponential. Moreover, it is highly speculated that this trend will come to an end because there is a limit to the miniaturization of transistors. Presently, the sizes of transistors are approaching the size of atoms. With the speeds of computer processors rounding the 5GHz mark, and talks about an exponential increase in speeds of up to 100GHz (http://en. Wikipedia.org/wiki/Moore's law), one might consider the possibility of us exceeding any further need of computational performance. However, this is not the case. Although computing speeds may increase exponentially, they are, and will continue to be, surpassed by the exponential increase in algorithmic

complexity as problem sizes continue to grow. Moore's law may continue to hold true for another decade or so, but different methods of computing are being researched.



# CHAPTER 3

# METHODOLOGY

## 3.0 Introduction

In this chapter, we shall present the research methodology of the study.

Combinatorial optimization is the process of finding the globally optimal configuration of discrete variables with respect to some function of the variables. Many combinatorial optimization problems are very difficult and are NP- hard.



A large number of combinatorial problems are of practical interest and importance, examples are the Traveling Salesman Problem, timetabling, routing and scheduling, and layout and placement problems.

In the Travelling Salesman Problem (TSP), we are given $n$ nodes and for each pair $\{i, j\}$ of distinct nodes, a distance $d_{i,j}$. We desire a closed path that visits each node exactly once (that is a salesman tour) and incurs the least cost, which is the sum of the distances along the path.

This task of visiting the constituencies can be modeled as a classical Traveling Salesman problem. An aspirant wishes to visit the twenty three (23) constituencies in the central region of Ghana. Minimizing the total travel distance to each constituency for campaign purposes saves time and reduces the cost of the campaign trip.

The TSP is to find the shortest circuitous path connecting n-number of cities. This means that a salesman following that path would visit each city only once. For smaller number of cities ($n \geq 4$) there is the possibility of considering even a manual solution of the TSP. However when the number of cities is large ($n \geq 10$) the method of manual

computation cannot be applied and computerized methods of finding all routed length can be impractically slow.

Manual computation is not practical because the number of circuits grows so fast that even for n=25 cities, it would take longer than the age of the universe ($\sim$10 billion years) to check all paths at a rate of one million paths per second since there are $(\frac{1}{2})n!$ paths according to Gato(1991).

A presidential aspirant's visit to a constituecy comes with a lot of benefits to both the aspirant and the electorate. Some of these benefits are catalogued below.

(i) It affords the voters an opportunity to have knowledge of issues. Of all the information voters obtain through the mass media during a presidential campaign, knowledge about where the candidates stand is most vital by Patterson and McClure (1976).

(ii) It gives the aspirant a fair idea of the specific challenges in the various constituencies and assures the electorate as to how such challenges would be addressed.

(iii) The aspirant seizes the opportunity to deliver his/her campaign message or policies and discuss his/her policy positions.

(iv) Some of the electorates get to see the aspirant for the first time as he/she is introduced to the electorates.

(v) The visit enables the aspirant to canvass for votes.

(vi) Party foot soldiers are motivated to hit the campaign trail even in the absence of the aspirant.

### 3.1 Variants of TSP

The TSP has provided a test bed for the development of algorithm such as the nearest neighbour rule that approximate optimal solutions of combinatorial optimization problems and on the other hand has prompted questions concerning the performance of such algorithms. The versatility of the application of TSP is briefly discussed below.



#### 3.1.1 The Vehicle Routing Problem

The Vehicle Routing Problem (VRP) is the m-TSP, where a demand is associated with each city or customer and each vehicle has a certain capacity. As a further constraint to the minimization of the distance covered in a typical TSP, the VRP also considers the minimization of the number of vehicles used. The constraints may include the available fuel capacity of each vehicle and available time windows for customers. TSP based algorithms have been applied in this kind of problem and may also be applied to routing problems in computer networks. (Gerard 1994).

Figure 3.1 shows an example of Vehicle Routing Problem (VRP) with four routes where the square in the middle denotes the source node.

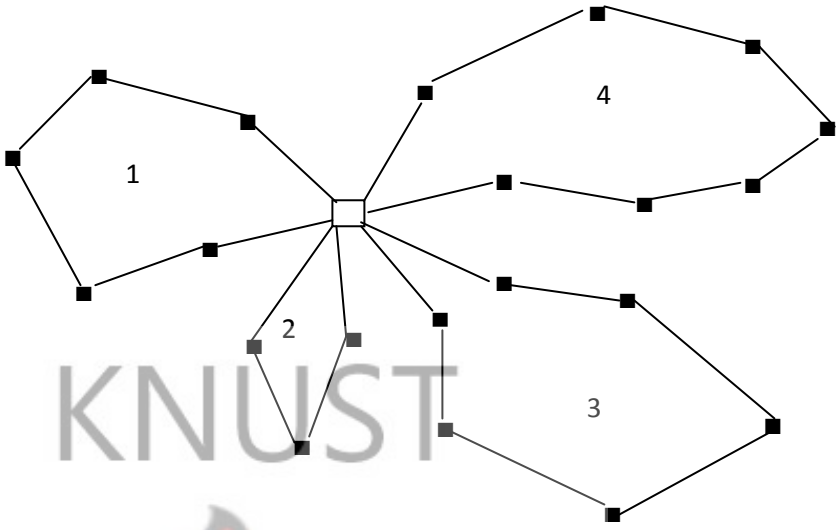




**Figure 3.1: A typical solution for a VRP with 4 routes. The square in the middle denotes the source node.**

Intrinsically, the VRP is a spatial problem. During the last few decades, however, temporal aspects of routing problems have become increasingly important. Specific examples of problems with time windows include bank deliveries, postal deliveries, industrial refuse collection, school-bus routing and situations where the customer must provide access, verification, or payment upon delivery of the product or service. In these problems customers could be served only during certain hours or the day, such as office hours or the hours before the opening of a shop. For example, a warehouse may only accept deliveries between 10:00 am and 4:00pm.Much attention however has been given to the Vehicle Routing Problem with Time Windows (VRPTW). The time windows can be hard or soft. In the hard time window case, if a vehicle arrives too early at a customer, it is permitted to wait until the customer is ready to begin service. However, a vehicle is not permitted to arrive at a customer after the latest time to begin service. The

field of multi-objective optimization is attracting more and more attention, notably because it offers new opportunities for defining problems, Jozefowiez (2008)..

In contrast, in the soft time window case, the time windows can be violated at a cost. The VRP can be modeled using the binary variables $x_{nm}^{v}$ and $y_{n}^{v}$ according to Goel (2006). $x_{nm}^{v}$ indicates whether $m \in \mathbb{N}$ is visited immediately after node $n \in \mathbb{N}$ by vehicle $v \in V$ $(x_{nm}^{v} = 1)\ or\ not\ (x_{nm}^{v} = 0)$. $y_{n}^{v}$ indicates whether node $n \in \mathbb{N}$ is visited by vehicle $v \in V$ $(y_{n}^{v} = 1)$ or not $(y_{n}^{v} = 0)$. For each $n \in \mathbb{N}$ the VRP contains the variables $t_n\ and\ P_n$. If node $n \in \mathbb{N}$ is visited by a vehicle, $t_n$ specifies the arrival time and $P_n$ specifies the current load of the vehicle. If no vehicle visits node $n \in \mathbb{N}$, both $t_n$ and $P_n$ are without meaning. The contribution of each vehicle $v \in V$ to the objective function is

$$\sum_{o \in O} y_{n(0,1)}^{v} P_o - \sum_{(n,m) \in A} x_{nm}^{v} C_{nm}^{v}$$

The first term represents the accumulated revenue of served orders; the second term represents the accumulated costs for the vehicle movements.

The VRP is maximize

$$\sum_{v \in V} \left( \sum_{o \in O} y_{n(0,1)}^{v} P_o - \sum_{(n,m) \in A} x_{nm}^{v} C_{nm}^{v} \right) \quad subject\ to$$

$$\sum_{(n,m) \in A} x_{nm}^{v} = \sum_{(m,n) \in A} x_{mn}^{v}\ for\ all\ v \in V, n \in \mathbb{N}$$

$$y_{n}^{v} = \sum_{(n,m) \in A} x_{nm}^{v}\ \ for\ all\ v \in V, n \in \mathbb{N}$$

$$\sum_{v \in V} y_{n}^{v} \leq 1\ for\ all\ n \in \mathbb{N}.$$

### 3.1.2 Arc routing problems

Arc Routing Problems (ARPs) are a special kind of vehicle routing problem in which the vehicles are constrained to traverse certain arcs, rather than visit certain nodes as in the standard Vehicle Routing Problem. The arcs represent streets which require some kind of treatment or service. Examples of ARPs are the Chinese Postman Problem, the Rural Postman Problem and the Capacitated Arc Routing Problem. In Arc Routing Problems the aim is to determine a least cost traversal of all edges or arcs of a graph, subject to constraints.

Compared to more common node routing problems, customers are here modeled as arcs or edges. Such problems arise naturally in several applications related to garbage collection, mail delivery, snow clearing, meter reading, school bus routing, police patrols etc. In contrast to the VRP, in the pickup and delivery problems each transportation request specifies both the locations where the load is to be picked up and the locations where it is to be delivered. Each load has to be transported by one vehicle from its set of origins to its set of destinations without any transshipment at other locations.

### 3.1.3 Computer Wiring

This type of problem is common in the design of computers and digital systems. The systems comprise a number of modules which in turn consists of several pins. The physical module position has already been determined. However a given subset of pins has to be interconnected by wires. Assuming two wires are attached to each pin in order to avoid signal cross talk and to improve ease of wiring, the aim is to minimize the total

wire length. Let $C_{ij}$ symbolize the actual distance between pin $i$ and $j$. The requirements imply that a minimum Hamiltonian path length must be found. This is done by introducing a dummy pin 0 where $c$0j = $c$j0 for all j. The problem of wiring thus becomes an ($n$+1) city symmetric TSP. A difficulty may arise if the position of the modules is a variable which must be chosen to minimize the total wire length for all subsets of the pins that must be connected (Gerard 1994).



### 3.1.4 Overhauling gas turbine engines

An application found by (Gerard 1994) is overhauling gas turbine engines in aircraft. Nozzle-guide vane assemblies, consisting of nozzle guide vanes fixed to the circumference, are located at each turbine stage to ensure uniform gas flow. The placement of the vanes in order to minimize fuel consumption can be modeled as a symmetric TSP.

### 3.1.5 Scheduling of jobs.

The scheduling of jobs on a single machine given the time it takes for each job and the time it takes to prepare the machine for each job is also a TSP. The total time to process each job is minimized. A robot must perform many different operations to complete a process. In these 22 applications, as opposed to the scheduling of jobs on a machine, we have precedence constraints. This is an example of a problem that cannot be modeled by a TSP but methods used to solve the TSP may be adapted to solve this problem (Gerard 1994).

### 3.1.6 Circuit partition

The cell partition problem is one of the problems reported in kirkpatrick (1983). The input to this problem is quite similar to that for NOLA. The essential difference is that each circuit element (called a cell) has a size associated with it. The cells are to be partitioned into two groups A and B. Let *Nets* be the number of nets that are in both A and B. Let $\Delta$ *Size* denote the magnitude of the difference in size between A and B. The objective is to find a partition (A, B) that minimizes r (A, B) = *N*ets + $\Delta$ *S*ize. In this measure, *N*ets and $\Delta$ *S*ize are weighted equally. Kirkpatrick et al. assigned different weights to these two components.



### 3.1.7 Optimal Linear Arrangement (OLA)

In the optimal linear arrangement (OLA) problem, we are given *n* circuit elements (cells, boards, chips, etc) and a set of nets which interconnect the circuit elements. For any linear ordering of these *n* elements, the maximum number of nets that cross between any pair of adjacent elements is called the *density* of the linear arrangement. We are required to find a linear ordering with minimum density. This problem is identical to the board permutation problem studied in Goto (1977) and Cohon (1983).

### 3.1.8 Ant colonies

Real ants are capable of finding the shortest path from a food source to the nest (Beckers et al., 1992; Goss et al., 1989) without using visual cues (Hölldobler and Wilson, 1990). Also, they are capable of adapting to changes in the environment; example is finding a new shortest path once the old one is no longer feasible due to a new obstacle. Consider

a situation where ants move in a straight line that connects a food source to their nest. It is well known that the primary means for ants to form and maintain the line is a pheromone trail. Ants deposit a certain amount of pheromone while walking, and each ant probabilistically prefers to follow a direction rich in pheromone. This elementary behaviour of real ants can be used to explain how they can find the shortest path that reconnects a broken line after the sudden appearance of an unexpected obstacle has interrupted the initial path. In fact, once the obstacle has appeared, those ants which are just in front of the obstacle cannot continue to follow the pheromone trail and therefore they have to choose between turning right or left. In this situation we can expect half the ants to choose to turn right and the other half to turn left. A very similar situation can be found on the other side of the obstacle. It is interesting to note that those ants which choose, by chance, the shorter path around the obstacle will more rapidly reconstitute the interrupted pheromone trail compared to those who choose the longer path. Thus, the shorter path will receive a greater amount of pheromone per time unit and in turn a larger number of ants will choose the shorter path. Due to this positive feedback (autocatalytic) process, all the ants will rapidly choose the shorter path.



### 3.1.9 Routing Algorithms

Routing is the process of deciding which path is more suitable and efficient to send a packet from a source to a destination and traveling through an unknown number of routers to achieve it. Packets carry a field in their header called destination field which carries the IP address of the destination device. A router must look at this field and decide which of the next available hop will be picked to efficiently deliver the packet to

its final destination. In order to do this, a router must look at its router table. The main problem with routers takes place in dynamically changing networks where a link failure can affect the performance of the entire network. Static shortest path first would be considered a static routing, where an administrator has to configure the network routing and change it manually in case a failure or any other negative event takes place in the network. Dynamic Shortest path is a routing algorithm that uses Dijsktra's algorithm to figure out the most optimal path to send a packet from source to destination.



### 3.2 Methods of solution of the TSP

Many attempts have been made at solving the TSP. Here are a few of the popular attempts to solving the problem:

#### 3.2.1 Branch-and-Bound

The general algorithm finds the global minimum of function $f: R^m \rightarrow R$ over an m-dimensional rectangle $\Re_{init}$. For a rectangle $\Re \subseteq \Re_{init}$ we define

$\Phi_{min}(\Re_{init}) = \min_{q \in \Re} f(q)$.

Then the algorithm computes $\Phi_{min}(\Re_{init})$ to within an absolute accuracy of $\epsilon > 0$, using two functions $\Phi_{lb}(\Re)$ and $\Phi_{ub}(\Re)$ defined over $\{\Re \mid \Re \subseteq \Re_{init}\}$. These two functions satisfy the following conditions.

i). $\Phi_{lb}(\Re) \leq \Phi_{min}(\Re) \leq \Phi_{ub}(\Re)$. Thus the functions $\Phi_{lb}(\Re)$ and $\Phi_{ub}(\Re)$ computes a lower and upper bound respectively on $\Phi_{min}(\Re)$.

ii). As the maximum half- length of the sides of $\mathfrak{R}$, denoted by size ($\mathfrak{R}$), goes to zero, the difference between upper and lower uniformly converges to zero, that is $\forall \in > 0\ \exists\ \delta > 0$ such that $\forall \mathfrak{R} \subseteq \mathfrak{R}_{init}$, size $(\mathfrak{R}) \leq \delta \Rightarrow \Phi_{ub}(\mathfrak{R}) - \Phi_{lb}(\mathfrak{R}) \leq \in$ .

The algorithm is as follows:

$$k = 0;$$

$$L'_{o} = \{\mathfrak{R}_{init}\};$$

$$L_{o} = \Phi_{lb}\{\mathfrak{R}_{init}\};$$

$$U_{o} = \Phi_{ub}\{\mathfrak{R}_{init}\};$$

While $U_k - L_k > \in$, {

Pick $R \in L'_k$ such that $\Phi_{lb}(\mathfrak{R}) = L_k$;

Split $\mathfrak{R}$ along one of its longest edges into $\mathfrak{R}_I$ and $\mathfrak{R}_{II}$;

Form $L'_{k+1}$, from $L'_k$ by removing $\mathfrak{R}_k$ and adding $\mathfrak{R}_I$ and $\mathfrak{R}_{II}$;

$L_{k+1} := min_{\mathfrak{R} \in L'_{k+1}} \Phi_{lb}(\mathfrak{R})$;

$U_{k+1} := min_{\mathfrak{R} \in L'_{k+1}} \Phi_{ub}(\mathfrak{R})$;

$k := k + 1$;

}

### 3.2.2 Genetic Algorithm

The genetic algorithm (GA) is an evolutionary algorithm inspired by Darvin (1859) and recently discussed by Dawkins (1986). Holland (1975) invented Genetic Algorithm as an adaptive search procedure. There has been a lot of intensive research on the use of GA to solve problems such as the TSP and transportation Problem by (Rachev and Ruschendorf 1993, Datta 2000). Generalized chromosome genetic algorithm (GCGA) was proposed for solving generalized traveling salesman problems (GTSP).

Theoretically, the GCGA could be used to solve classical traveling salesman problem (CTSP) by Yang (2008).

According to Amponsah and Darkwah (2007), the GA has the following simulations of the evolutionary principles:



**Table 3.1: The relationship between Evolution and Genetic Algorithm.**

| Evolution | Genetic Algorithm |
|---|---|
| An **individual** is a genotype of the species. | An **individual** is a solution of the optimization problem. |
| **Chromosomes** define the structure of an individual. | **Chromosomes** are used to represent the data structure of the solution. |
| Chromosome consists of sequence of cells called **genes** which contain the structural information. | Chromosome consists of a sequence of gene species which are **placeholder boxes** containing string of data whose unique combination gives the solution value. |
| The genetic information or trait in each gene is called an **allele**. | An **allele** is an element of the data structure stored in a gene placeholder. |
| **Fitness** of an individual is an interpretation of how the chromosomes have adapted to the competition environment. | **Fitness** of a solution consists in evaluation of measures of the objective functions for the solution and comparing it to the evaluations for other solutions. |
| A **population** is a collection of the species found in a given location. | A **population** is a set of solutions that form the domain search space. |
| A **generation** is a given number of individuals of the population identified over a period of time. | A **generation** is a set of solutions taken from the population (domain) and generated at an instant of time or in an iteration. |
| **Selection** is pairing of individuals as parents for reproduction. | **Selection** is the operation of selecting parents from the generation to produce offsprings. |
| **Crossover** is mating and breeding of offspring by pairs of parents whereby chromosome characteristics are exchanged to form new individuals. | **Crossover** is the operation whereby pairs of parents exchange characteristics of their data structure to produce two new individuals as offsprings. |
| **Mutation** is a random chromosomal process of modification whereby the inherited genes of the offspring from their parents are distorted. | **Mutation** is random operation whereby the allele of a gene in a chromosome of the offspring is changed by a probability pm. |
| **Recombination** is a process of nature's survival of the fittest. | **Recommendation** is the operation whereby elements of the generation and elements of the offspring form an intermediate generation and less fit chromosomes are taken from the generation. |

Given a population at time t, genetic operators are applied to produce a new population at time t+1. A step-wise evolution of the population from time t to t+1 is called a generation. The Genetic Algorithm for a single generation is based on the general GA framework of Selection, Crossover, Mutation and Recombination.

### 3.2.2.1 Representation of individuals



For the purpose of crossover and mutation operation the variables in the genetic algorithm may be represented by amenable data structure.

Suppose we have the search space $x = 0, 1, 2, \ldots, 10$ then the $x$ values form the individuals. The elements of the search space in a binary sequence are encoded by expressing $x = 10\ and\ x = 0$ in the binary sequence to obtain $10 = 1010_2\ and\ 0 = 0000_2$. Thus $x = 10$ is an individual and 1010 is its chromosome representation. The chromosome has 4 genes placeholders for the alleles. The allele information in the genes will be the binary numbers 0 and 1. The chromosome for $x = 9$ is therefore

1 0 0 1

There are $2^4$ permutations for a binary string of length 4. These $2^4$ permutations consist of both infeasible and feasible solutions. There are 11 feasible solutions which constitute the search space and the rest for the infeasible set. Since the solution set is restricted to the integers we look for suboptimal solution. In general the data structure used for the representation of individual depends on variables of the problem at hand.

### 3.2.2.2 Fitness function

This is the measure associated with the collective objective functions of the optimization problem. The measure indicates the fitness of a particular chromosome representation of a particular individual (solution). In the TSP, the fitness function is the sum of the path between the cities.

$$f = \sum_{i=1}^{n-1} d(c_i, c_{i+1})$$

$d(.)$ is a distance function

$n$ is the number of cities

$c_i$ is the $ith$ city.



### 3.2.2.3 Initial population

A genetic algorithm begins with a population of potential solutions. These solutions are encoded in chromosomes. For function optimization, the solution $x$ in the objective function $f(x)$ will be encoded in a chromosome consisting of binary string. Thus $x = 13$ is represented as

$x = 13_{10} = 1101_2$. For a tour of five cities in the TSP, the index $1,2,3,4,5$ may be used for the cities and represent a tour by the permutation $T = [1, 2, 3, 4, 5]$. Each potential solution must be a feasible solution as well as being a unique solution.

### 3.2.2.4 Population size

The population size indicates how much of the search space the GA will search in each iteration. Smaller size could mean the algorithm takes smaller a shorter time to find the optimal solution. Similarly when the size is large the algorithm take a longer time in

sampling the large number of chromosomes in order to obtain the best chromosome. Research efforts are needed to establish the relationship between the population size, the number of variables in the algorithm and the number of possible states in the solution space.

#### 3.2.2.5 Selection process



The general selection process involves reproduction, crossover and mutation operations. The selection process is used to generate a new population from the current one. The objective is to select individuals with high fitness. It is used for selecting individuals for crossover and mutation. The following are some selection processes used;

##### 3.2.2.5.1 Reproduction (Elitist) selection

A percentage of the current population which highly fits is copied directly as part of the new generation.

##### 3.2.2.5.2 Proportional Fitness (Roulette wheel) selection.

This is biased towards chromosomes with best fitness values. However a wide range of chromosomes are selected. In the first stage, a roulette wheel is constructed by computing the relative fitness of each chromosome as

$$w_i = {f_i}/{\sum_{k=1}^{n} f_k}$$

Where $f_k$ is the fitness of $kith$ chromosome.

We then find the cumulative fitness $(c_j)\ of\ the\ jth$ chromosome as

$$c_j = \sum_{i=1}^{j} w_i$$

This creates the roulette wheel.

In the second stage a random number $r_j$ is chosen and if $r_j > c_j$ then the $ith$ chromosome is selected.

The above calculation is based on maximization problems. For minimization problem define



$$F_i = (f_{max} - f_i) + 1$$

$$And \quad w_i = {F_i}/{\sum_{k=1}^{n} F_k}$$

Where $f_{max}$ is the maximum fitness of all chromosomes.

$F_k$ is the reverse magnitude fitness.

Selection is a process of choosing a pair of organism to reproduce. The selection function can be any increasing function and proportional fitness selection is a clear example.

**3.2.2.5.3 Tournament Selection**

Two chromosomes are chosen at random. The one with the higher fitness is selected.

The process is repeated until the required numbers of chromosomes are obtained.

**3.2.2.5.4 Random selection**

Chromosomes may be selected randomly irrespective of their fitness.

### 3.2.2.6 Crossover.

After the required selection process the crossover is used to divide a pair of selected chromosomes into two or more parts. Parts of one of the pairs are joined to parts of the other chromosome with the requirement that the length should be preserved. The point between two alleles of a chromosome where it is cut is called crossover point. There can be more than one crossover point in a chromosome. The crossover point, $I$ is the space between the allele in the $ith$ position and the one in $(i + I)th$ position. For two chromosomes the crossover point are the same and the crossover operation is the swapping of similar parts between the two chromosomes. The crossover operation may produce new chromosomes, which are less fit. In that sense the crossover operation results in non-improving solution.



#### 3.2.2.6.1 Single point crossover

A single point along the chromosome is selected. The parts of the parents on the left or right of the crossover point are swapped to get new chromosomes.

#### 3.2.2.6.2 Double point crossover

Two points are chosen as crossover points. This separates the chromosomes into three parts. The middle parts are swapped to obtain new chromosomes.

#### 3.2.2.6.3 Uniform crossover.

Single alleles in the same position are considered for swapping. The probability of selecting an allele for swapping is called Mixing Rate. Mixing rates are set for the allele

positions. Random numbers are then generated and a position satisfying the mixing rate has the allele in the two chromosomes swapped. Crossover operation is an explanatory operation that allows the GA to take 'large jumps' during the search. As convergence is approached the explanatory power of the crossover diminishes.

#### 3.2.2.7 Mutation



Mutation operation is performed on the individual chromosome whereby the alleles are changed probabilistically.

##### 3.2.2.7.1 Random swap mutation

In random swap two loci (position) are chosen at random and their values swapped.

##### 3.2.2.7.2 Move-and-insert gene mutation

Using move-and-insert, a locus is chosen at random and its value is inserted before or after the value at another randomly chosen locus.

##### 3.2.2.7.3 Move-and-insert sequence mutation

Sequence mutation is very similar to the gene move-and-insert but instead of a single locus a sequence loci is moved and inserted before or after the value at another randomly chosen locus.

#### 3.2.2.7.4 Uniform mutation probability

A probability parameter is set and for all the loci an allele with greater or same probability as the parameter is mutated by reversing its allele.

### 3.2.2.8 Termination conditions

The algorithm terminates when a set of conditions are satisfied. At that point the solution with highest fitness among the current generation of the population is taken as the global solution or the algorithm may terminate if one or more of the following are satisfied;

(i) A specified number of total iteration is completed.

(ii) A specified number of iteration is completed within which the solution of best fitness has not changed.

(iii) The standard deviation of the generation of the population approaches a given large value.

(iv) The average fitness of the generation of population does not differ significantly from the solution of best fitness.

Goldberg 1989 presented a Standard Genetic Algorithm, which was also called Simple Genetic Algorithm (SGA). It is an algorithm that captures the most essential components of every genetic algorithm. The steps in SGA are;

(i) Start with a population of n random individuals ($x$) each with L-bit chromosome representation.

(ii) Calculate the fitness $f(x)$ of each individual.

(iii) Choose, based on fitness, two individuals and call them parents. Remove the parents from the population.

(iv) Use a random process to determine whether to perform crossover. If so, refer to the output of the crossover as children. If not, simply refer to the parents as the children.

(v) Mutate the children probability $P_m$ of mutation for each bit.

(vi) Put the children into an empty set called the new generation.

(vii) Return to step ii) until the new generation contains n individuals. Delete one child at random if n is odd. Then replace the old population with the new generations. Return to step i).

The Simple Genetic algorithm can be summarized in the following steps.

**Table 3.2: Summary of steps in SGA.**

| |
|---|
| Step 1: Code the individual of the search space. |
| Step 2: Initialize the generation counter (g=1). |
| Step 3: Choose initial generation of population (solution). |
| Step 4: Evaluate the fitness of each individual in the population. |
| Step 5: Select individuals of best fitness ranking by fitness proportionate probability. |
| Step 6: Apply crossover operation on selected parents. |
| Step 7: Apply mutation operation on offspring |
| Step 8: Evaluate fitness of offspring |
| Step 9: Obtain a new generation of population by combining elements of the offspring and the old generation by keeping the generation size unchanged |
| Step 10: Stop, if termination condition is satisfied |
| Step 11: Else $g = g + 1$ |

### 3.2.3 Omicron Genetic Algorithm

The literature in evolutionary computation has defined a great variety of Gas that maintain the same philosophy of varying operators and adding different principles like elitism in [Goldberg, (1989) and $M¨uhlenbein$ and Hans-Michael Voigt, (1995)]. Using the Simple Genetic Algorithm as a reference, this section presents a new version, the Omicron Genetic Algorithm (OGA), a Genetic Algorithm designed specifically for the TSP.



#### 3.2.3.1 Codification

The OGA has a population $P$ of $p$ individuals or solutions, as the SGA does. Every individual $Px\ of\ P$ is a valid TSP tour and is determined by the arcs $(I, j)$ that compose the tour. Unlike the SGA, that uses a binary codification, the OGA uses an $n$-ary codification. Considering a TSP with 5 cities $c1, c2, c3, c4\ and\ c5$, the tour defined by the arcs $(c1, c4), (c4, c3), (c3, c2), (c2, c5) and\ (c5, c1)$ will be codified with a string containing the visited cities in order, which is $[c1; c4; c3; c2; c5]$.

#### 3.2.3.2 Reproduction

The OGA selects randomly two parents ($F1$ and $F2$) from the population $P$, as does an SGA reproduction. The selection of a parent is done with a probability proportional to the fitness of each individual $Px$, where $fitness\ (P_x) \propto 1/l(P_x)$. Unlike the SGA, where two parents generate two offspring, in the OGA, both parents generate only one offspring. In the SGA, $P$ offspring are obtained first to completely replace the old generation. In the OGA, once an offspring is generated, it replaces the oldest element of

*P*. Thus, the population will be a totally new one in *P* iterations and it would be possible to consider this population a new generation. In conclusion, the same population exchange as in the SGA is made in the OGA, but in a progressive way.

### 3.2.3.3 Crossover and Mutation

The objective of crossover in the SGA is that the offspring share information of both parents. In mutation, the goal is that new information is added to the offspring, and therefore is added to the population. In the SGA, the operator's crossover and mutation are done separately. To facilitate the obtaining of offspring that represent valid tours, the crossover and the mutation operators are done in a single operation called Crossover-Mutation in OGA. Even so, the objectives of both operators previously mentioned will stay intact.

To perform Crossover-Mutation, the arcs of the problem are represented in roulette, where every arc has a weight *w* or a probability to be chosen. Crossover-Mutation gives a weight *w* of one to each arc $(i; j)$ belonging to set *A*, that is $w_{ij} = 1 \; \forall \; (i; j) \in A$. Then, a weight of $O/2$ is added to each arc $(i; j)$ of *F*1, that is $w_{ij} = w_{ji} + O/2 \; \forall \; (i; j) \in F1$, where Omicron (*O*) is an input parameter of the OGA. Analogously, a weight of $O/2 \; \forall j \in N_i$ is added to each arc $(i; j)$ of *F*2.

Iteratively, arcs are randomly taken using the roulette to generate a new offspring. While visiting city *i*, consider *Ni* as the set of cities not yet visited and that allows the generation of a valid tour. Therefore, only the arcs $(i; j) \; \forall j \in N_i$ participate in the roulette, with their respective weights $w_{ij}$. Even so the crossover is done breaking the parents and interchanging parts in the SGA instead of taking arcs iteratively with high

probability from one of the parents in the OGA, the philosophy of both crossover operators is the same.

To generate an offspring SI, an arc of one of the parents will be selected with high probability (similar to crossover). But it is also possible to include new information since all the arcs that allow the creation of a valid tour participate in the roulette with probability greater than 0 (similar to mutation). The value $O/2$ is used because there are two parents, and then $w_{max} = O + 1$ can be interpreted as the maximum weight an arc can have in the roulette (when the arc belongs to both parents). When the arc does not belong to any parent, it obtains the minimum weight $w_{min}$ in the roulette, that is $w_{min} = 1$. Then, O determines the relative weight between crossover and mutation.

Formally, while visiting city $i$, the probability of choosing an arc $(i; j)$ to generate the offspring SI is defined by equation (1) below;

$$P_{ij} = \begin{cases} \frac{W_{ij}}{\sum_{\forall h \in N_i} W_{ij}} & if\ j \in N_i \\ 0 & otherwise \end{cases}$$

**3.2.3.4 Example**

To clarify the previous procedure, an example considering the TSP with 5 cities mentioned above is presented next. $O = 4\ and\ p = 4$ are considered for this case.

**3.2.3.4.1 Reproduction**

The example assumes an initial population $P = \{P_x\}$ composed of 4 randomly selected individuals with their respective fitness's $fx$. This initial population is presented next.

First randomly chosen individual: $P1=\{c1; c4; c3; c2; c5\}\ with\ f1 = 10$.

Second randomly chosen individual: $P2=\{c1; c3; c2; c5; c4\} with\ f2 = 8.$

Third randomly chosen individual: $P3=\{c3; c5; c1; c2; c4\} with\ f3 = 1.$

Fourth randomly chosen individual: $P4=\{c2; c5; c4; c1; c3\} with\ f4 = 5.$

Two parents are randomly selected through roulette, where the weights of the individuals in the roulette are their fitness. It is assumed that individuals $P1$ and $P4$ are selected to be parents.

$F1=\{c1; c4; c3; c2; c5\} = \{(c1; c4); (c4; c3); (c3; c2); (c2; c5); (c5; c1)\}$

$F2=\{c2; c5; c4; c1; c3\} = \{(c2; c5); (c5; c4); (c4; c1); (c1; c3); (c3; c2)\}.$

**3.2.3.4.2 Crossover- Mutation.**

**Iteration 1**

First, an initial city is randomly chosen to perform Crossover- Mutation. $c4$ is assumed as the initial city. Then, $Nc4$ is composed by $[c1; c2; c3; c5]$, that is the set of not yet visited cities. The arc $(c4; c2)$ has a weight of 1 in the roulette because it does not belong to any parent. Arcs $\{(c4; c3); (c4; c5)\}$ have a weight of $1 + 0/2 = 3$ in the roulette because they belong to one parent. Finally, the arc $(c4; c1)$ has a weight of $1+O = 5$ in the roulette because it belongs to both parents. It is assumed that the arc $(c4; c3)$ is randomly chosen through the roulette.

**3.2.3.4.3 Crossover- Mutation.**

**Iteration 2**

From $c3$ we do crossover mutation operation.

*Nc*3 is composed by *{c1; c2; c5}*. The arc (*c*3; *c*5) has a weight of 1in the roulette because it does not belong to any parent. The arc (*c*3; *c*1) has a weight $1 + O/2 = 3$ in the roulette because it belongs to one parent. Finally, the arc (*c*3; *c*2) has a weight of 1+*O* = 5 in the roulette because it belongs to both parents.

It is assumed that the arc (*c*3; *c*2) is randomly chosen through the roulette.



**3.2.3.4.4 Crossover- Mutation.**

**Iteration 3**

From *c*2 we do crossover mutation operation.

*Nc*2 is composed by *[c1; c5]*. The arc (*c*2; *c*1) has a weight of 1 in the roulette because it does not belong to any parent. Finally, the arc (*c*2; *c*5) has a weight of 1+*O* = 5 in the roulette because it belongs to both parents. It is assumed that the arc (*c*2; *c*1) is randomly chosen through the roulette.

**3.2.3.4.5 Crossover- Mutation.**

**Iteration 4**

*Nc*1 is composed by *[c5]*. The arc (*c*1; *c*5) has a weight of $1 + O/2 = 3$ in the roulette because it belongs to one parent. The arc (*c*1; *c*5) is chosen because it is the unique arc represented in the roulette. The new offspring is *S*1= *[c4; c3; c2; c1; c5]* = *{*(*c*4; *c*3); (*c*3; *c*2); (*c*2; *c*1); (*c*1; *c*5); (*c*5; *c*4)*}*. Notice that *S*1 has 3 arcs of *F*1 {(c4; c3); (c3; c2); (c1; c5)} and 2 arcs of *F*2 {(*c*3; *c*2); (*c*1; *c*5)}. Also, *S*1 has an arc {(c2; c1)} that does not belong to any parent. This shows that the objectives of the operators (crossover and mutation) have not been altered.

### 3.2.3.4.6 Population Update

The new individual $S1$ replaces the oldest individual $P1$. Next, the new population is shown.

$P1=\{c4; c3; c2; c1; c5\}\ with\ f1 = 7.$

$P2= \{c1; c3; c2; c5; c4\} with\ f2 = 8.$

$P3=\{c3; c5; c1; c2; c4\} with\ f3 = 1.$

$P4= \{c2; c5; c4; c1; c3\} with\ f4 = 5.$



The entire procedure above is done iteratively until an end condition is satisfied.

## 3.3 Simulated Annealing

Simulated Annealing resulted from observation of the analogy between the physical process of annealing and of finding a global optimum for a combinatorial optimization problem (Kirkpatrick et al., (1983) and Cerny (1985)).

Simulated Annealing is a general purpose combinatorial optimization technique that has been proposed by Kirkpatrick et al. (1983). This method is an extension of a Monte Carlo method developed by Metropolis et al. (1953), to determine the equilibrium state of a collection of atoms at any given temperature *T*. Since the method was first proposed in Kirkpatrick et al. (1983), much research has been conducted on its use and properties. Most of the papers that report on an application of the method deal with problems that arise in the CAD area. This is not surprising as most CAD problems are known to be NP-complete, Sahni (1980). Hence, CAD problems are likely targets of solution by heuristic methods.

Simulated annealing as proposed by Kirkpatrick et al. (1983) is a special case of a wider class of *adaptive heuristics* for combinatorial optimization. This wider class is formulated below. The term *adaptive*, in this context, simply means that some of the parameters of the heuristic may be modified. This modification may be done by the algorithm itself using some learning mechanism or may be done by the user using his/her own learning mechanism.



Consider any optimization problem. Suppose we wish to find a solution that minimizes the objective function $h(\ )$ subject to certain constraints (a maximization problem may be solved by minimizing the negative of the objective function). Solutions that satisfy the constraints are called *feasible solutions* and a feasible solution with minimum $h(\ )$ value is called an *optimal solution*. The *optimal value* of $h(\ )$ is its minimum value. The general form of an adaptive heuristic to find a feasible solution with value close to optimal takes the form shown in Table 3.3 below. The significance of the variables, functions, and procedures used in this algorithm are described below:

**Table 3.3: The general adaptive heuristic**

```
Procedure General Adaptive Heuristic ;
    { General form of an adaptive heuristic for combinatorial optimization }
    S := S_o; {initial solution}
    Initial heuristic parameters;
    repeat
        repeat
            NewS := perturb (S);
            if accept (NewS, S ) then S := NewS;
        until "time to adapt parameters";
        Adapt Parameters;
    until "terminating criterion";
end; {of General Adaptive Heuristic }
```



The class of simulated annealing heuristics proposed by Kirkpatrick et al. (1983) is obtained from the general adaptive heuristic of Table 1 by making the following parameter selections:

1) The acceptance function, accept, has the form:

   **if** $\left(h\,(NewS) < \boldsymbol{h}\,(S)\right)\boldsymbol{or}\,(random < e^{(h(S)-h(NewS))/T})$

   **then** accept := **true**

   **else** accept := **false**;

   where T is a heuristic parameter called "temperature" and random is a uniformly generated pseudo-random number in the range [0, 1]

2) The "time to adapt parameters" criterion is the number of iterations of the inner **repeat** loop that have been performed since the last adaptation.

3) The procedure *AdaptParameters* does the following;

   The "temperature", *T*, used in the acceptance function is updated to $\alpha^{*}T$ for some constant $\alpha$, $0 < \alpha < 1$ and the number, *iterations*, of iterations of the

inner **repeat** loop that are to be performed before the next adaptation is changed to $\beta^*\ iterations$. $\beta$ is a constant that is at least 1.

4) The "terminating criterion" is when we have used up the amount of computing time we wish to spend.

Substituting these selections into Table 1, the form of simulated annealing heuristic in Table 2 below is obtained.

Simulated annealing is simpler to use than the general adaptive heuristic as there are fewer decisions to be made. We essentially need to determine the following:

(i) How is $S_o$ to be generated?

(ii) What are the values of $T_o\ ;\ i_o;\ \alpha, and\ \beta$.

(iii) How much computer time is the heuristic allowed?

These choices have a significant impact on the quality of (that is the value of S at termination) of the solution produced, Nahar (1985). Determining optimal choices for these is not possible as the optimal choice depends not only on the particular problem being solved but also on the particular instance being solved. The time required to optimize the choices is, perhaps, better spent running the algorithm for a longer time.

**Table 3.4: Simulated annealing**

**Procedure** *Simulated Annealing* ;

{ General form of Simulated Annealing }

$S$ := $S_o$; {initial solution}

$T$ :=$T_o$ ; {initial temperature}

*iterations* := $i_o$ ; {initial number of iterations of inner loop, ≥ 1 }

**repeat**

**repeat**

*NewS* := *perturb* (S);

**if** $\left(h\,(NewS) < \boldsymbol{h}\,(S)\right)\boldsymbol{or}\,(random\ <\ e^{(h(S)-h(NewS))/\,T})$

**then** accept := **true**

**else** accept := **false**;

**until** inner loop has been repeated iterations times;

$T$ := $\alpha^{*}T$; iterations := $\beta^{*}\ iterations$

**until** "*out of time*";

**end**; {of *Simulated Annealing* }

Simulated annealing is a heuristic- based search algorithm, motivated by an analogy of physical annealing in solids. It is capable of solving combinatorial optimization problem. The method of simulated annealing has been used to find the global minima cost configuration for NP- complete problems with many local minima.

According to Amponsah and Darkwah (2007) the concept of Simulated is derived from Statistical mechanics in the area of natural science. A piece of regular metal in its natural state has the magnetic directions of its molecules aligned in a uniform direction. In the preparation of alloys the metals are heated to a very high temperature where the molecules acquire higher energy state. The basic structure of the metallic bonds break down and magnetic directions of the molecules are oriented randomly. Annealing is the

slow cooling of the metallic material so that at natural temperature conditions the metal will achieve regularity of the alignment of the magnetic direction so as to make the metal stable for use. Hasty cooling of solids results in defective crysal structure. In 1953 Metropolis and others recognized the use of Boltzman Law to simulate the efficient equilibrium condition of a collection of molecules at a given temperature and thus facilitated annealing. When the metal is heated to higher temperature with higher energy state and it is being cooled slowly, it is assumed that for a finite drop in temperature the system state change in the sense that the molecules assume new configuration of arrangement. The configuration depends on parameters like temperature, the energy of the system and others. Combining the parameters, an energy function is obtained from which the configuration can be obtained.

Many variations of the SA have evolved. Lin (1994) discovered the hybrid algorithm (IIA), which is based on a hybrid mechanism which combines conventional heuristics with low temperature simulated annealing (LTSA). A major disadvantage of the technique is that it is extremely slow and hence not suitable for complex optimization problems such as scheduling. There are many attempts to develop parallel versions of the algorithm. Many of these algorithms are problem dependent in nature, Ram (1996).

In 1983, Kirk Patrick showed how Simulated Annealing of Metropolis (1953) could be adapted to solve problems in combinatorial optimization.

The following analogy was made:

(i) a) Annealing looks for system state at a given temperature and energy.

b) Optimization looks for feasible solution of the combinatorial problems.

(ii) a) Cooling of the metal is to move from one system state to another.

b) Search procedure (algorithm scheme) tries one solution after another in order to find the optimal solution.

(iii) a) Energy function is used to determine the system state and energy.

b) Objective (cost) function is used to determine a solution and the objective functions value.

(iv) a) Energy results in evaluation of energy function and the lowest energy state corresponds to stable state.



b) Cost results in evaluation of objective function and the lowest objective function value corresponds to optimal solution.

(v) a) Temperature controls the system state and the energy.

b) A control parameter is used to control the solution generation and the objective function value.

Simulated annealing (SA) is a generic probabilistic meta-heuristic 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 (example is 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.

### 3.3.1 Using Simulated Annealing to solve TSP

To be able to use simulated annealing to find good solutions for the Traveling Salesman Problem it is necessary to describe a configuration, a neighbourhood or neighbour generation mechanism and a cost function.

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



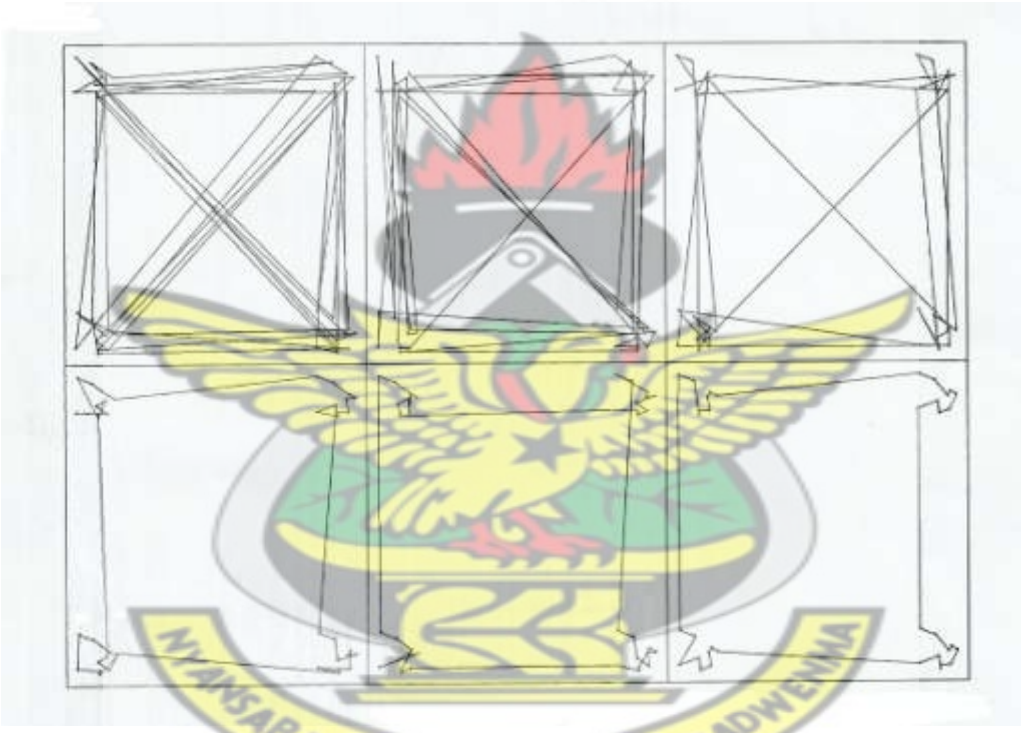

**Figure 3.2: Evolution of a Travelling Salesman Problem Solution using Simulated Annealing.**

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 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, neighbour, and random*. Note that the test in Step g is designed so that large steps uphill are unlikely to be taken except at high temperatures *t*. The probability that an uphill move with a given cost Δ 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.



**3.3.2 General schema for a simulated annealing algorithm.**

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

b. Determine a starting temperature *T*.

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

d. Choose a *random neighbour* $S^*$ of the current solution.

e. Set Δ = *Length* ($S^*$) = *Length (S)*.

f. If $\leq 0$ (downhill move):

Set S = $S^*$

g. If *L*ength (*S*) < *L*ength ($S^*$), set $S^* = S$.

h. If *l*ength (*S*) < *l*ength ($S^*$) (uphill move):

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

If $r < e^{-\Delta/T}$, set *S* = $S^*$.

i.End "While not yet at equilibrium '' loop.

j. Lower the temperature *T*.

k. End "While not yet frozen" loop.

l. Return $S^*$.

### 3.3.3 Configuration

The application of the simulated annealing method to any optimization problem requires definition of four major components:

(i) *Problem Configuration*: a clear specification of the domain over which the optimal solution is searched. Constraint equations are mainly used to express this domain for the optimal solution.

(ii) *Neighbourhood Configuration*: the random method of iteratively perturbing the design vector to create new trail points which will be the options presented to the system.

(iii) *Objective function*: (analog of energy) whose minimization is sought by the procedure. This is a scalar equation that weighs all of the design variables to provide a measure of goodness for each option.

(iv) *Annealing Schedule*: (analog of temperature) a controlled parameter which tells how the parameter will be decremented in each iteration of the outer loop and specify the number of inner loop iterations.

The successful implementation of Simulated Annealing depends on:

(i) The choice of neighbourhood solutions.

(ii) The cooling schedule, which is defined by the following parameters;

(i) Initial temperature ($T_o$)

(ii) Final temperature ($T_f$)

(iii) The temperature update at each iteration.

(iv) The stopping criterion.

### 3.3.4 Cooling Schedules

A cooling schedule is a description of the values of the control parameter and number of transitions performed at each value of control parameter by a simulated annealing algorithm.

The cooling schedule of a simulated annealing algorithm consists of four components:

**(i) Starting temperature**.

The starting temperature is an input by the user of the programme. The temperature must be high enough to allow a move to almost any neighbourhood state. However, if the starting temperature value is too high, then the search can move to any neighbour and thus transform the search (at least in the early stages) into a random search. Effectively, the search will be random until the temperature is cooled enough to start acting as a simulated annealing algorithm.

**(ii) Final temperature**.

It is usual to let the temperature decrease until it reaches zero. However, this can make the algorithm run for a lot longer. In practice, it is not necessary to let the temperature reach zero because as it approaches zero, the chances of accepting a worse move are almost the same as the temperature being equal to zero. Therefore, the stopping criteria can either be a suitably low temperature or when the system is frozen at the current temperature (that is no better or worse moves are being accepted).

**(iii) Temperature decrement**.

Once the starting and stopping temperatures are known, we need to get from one to the other. Theory states that we should allow enough iteration at each temperature so

that the system stabilizes at that temperature. Theory also states that the number of iterations at each temperature to achieve this might be exponential to the problem size. As this is impractical, we need to compromise. We can either do this by performing a large number of iterations at a few temperatures, a small number of iterations at many temperatures or a balance between the two.

**(iv) Iterations at each temperature**.

In our specific TSP, this can only be a finite number of cities that can be a link to any one city. The number of iterations of the TSP is therefore unrestricted.

The initial temperature may be obtained by computing the objective function values of several neighbourhood solutions (M) of the initial solution x and taking the difference.

If $\Delta_{-}$ is the maximum difference then we may take:

(i) $T_o = \frac{\Delta_{max}}{L_n(P)}$ , $where \;\; P \in (0,1) \;\; say \; P = 0.8 \;\; or$

(ii) $T_o = \alpha\Delta_{max}$ , $where \;\; \alpha \geq 1$ .

The final temperature is fixed a priori as a small value and used as stopping criteria alone or with other parameters including a given number of iterations. Amponsah (2007) discussed that some of the methods used in updating the temperature are:

(i) $T_{k+1} = T_{k-a}$,

$where \; a = \langle \frac{T_{o-T_f}}{M} \rangle \, L \;\; where \; L \; runs \; are \; used \; for \; each \; iteration.$

(ii) $T_{k+1} = \frac{T_k}{(1+\beta T_k)}$ , $\beta = 0.02$

(iii)$T_{k+1} = \beta T_k$ , $with \; \beta \in [0.50, 0.99] \; chosen \; once \; say \; \beta = 0.95.$

### 3.3.5 Solutions and Acceptance Criteria.

In each step of the algorithm, at every given temperature, a new distance is calculated for a given configuration and then the configuration is given a small random disturbance (city swap). The new distance is computed and the difference between the newly computed distance and the old distance ($\delta f$) is noted. The number of iterations for a particular temperature is left out of the algorithm because there are only a finite number of routes joining them. The temperature is reduced by a factor until the temperature tit reaches zero. If $\delta f \leq 0$, the displacement is accepted. The case $\delta f > 0$ is treated probabilistically. The probability that the configuration is accepted is given in (1).

$$P = \exp\left(-\frac{\delta f}{T}\right) > r,$$

*where*

$\delta$ $= the\ change\ in\ objective\ function$

$T$ $= the\ current\ tempeature$

$r$ $= a\ random\ number\ uniformly\ distributed\ between\ 0\ and\ 1.$

The probability of accepting a worse move is a function of both the temperature of the system and of the change in the objective function. As the temperature of the system decreases, the probability of accepting a worse move is decreased. If the temperature is zero, then only better moves will be accepted.

The choice of a solution in the neighbourhood of $N(x)$ of $x$ may be:

(i) one solution at a time.

(ii) the best subset of solutions from $N(x)$.

Simulated annealing has been applied to many engineering problems including scheduling, image correction, mechanism synthesis, design of integrated circuits, and path generation for robotic obstacle avoidance.

### 3.3.6 Combining Simulated annealing with other methods.

Simulated annealing can be used to either provide a good starting configuration for another method or improve upon a configuration found by another method. The former situation might be where simulated annealing generates a starting point for a branch and bound exact algorithm. The latter requires that the initial control parameter value be lower than normal otherwise the starting configuration's good features are quickly lost as cost increasing transitions are possibly accepted.



## 3.4 Model and Algorithms

### 3.4.1 Distance

The distance between two objects could be described as the length of physical separation between the objects. In the most general terms however, it is a numerical description of this separation referring to length, period of time etc. This is a scalar quality. In geometry, the minimum distance between two points is the length of the line segments joining the two points. The distance between two points $(x_1, y_1, z_1)$ and $(x_2, y_2, z_2)$ in three dimensional spaces is given by Deza( ) as:

$$d = \sqrt{(\Delta x)^2 + (\Delta y)^2 + (\Delta z)^2} = \sqrt{(x_1 - x_2) + (y_1 - y_2) + (z_1 - z_2)}$$

……(1)

For two points in the xy- plane, $\Delta z = z_1 = z_2 = 0$ and equation (1) holds.

### 3.4.2 Types of Distances

The distance between two points is usually given by the 2-norm distance called the Euclidean distance. Table 3.5 below shows other distances in n dimensional space given by other norms according to Deza (1985).

**Table 3.5: Types of Distances 1**

| | |
|---|---|
| 1-norm distance | $= \sum_{i=1}^{n} \mid x_i - y_i \mid$ |
| 2-norm distance | $= m \langle \sum_{i=1}^{n} \mid x_i - y_i \mid^{2} \rangle^{1/2}$ |
| p-norm distance | $= m \langle \sum_{i=1}^{n} \mid x_i - y_i \mid^{p} \rangle^{1/p}$ |
| Infinity norm distance | $= \lim_{p \to \infty} \langle \sum_{i=1}^{n} \mid x_i - y_i \mid^{p} \rangle^{1/p}$ $= \max(\mid x_1 - y_1 \mid, \mid x_2 - y_2 \mid, \ldots\ldots, \mid x_n - y_n \mid)$ |

The 2-norm distance is the type of distance that has been used in many TSP problems and it represents the shortest distance between any two points and is the distance that would have been measured physically with say a ruler. The use of such a distance type in the TSP assumes that the two points (cities) are connected by a perfectly straight road or other transportation means. This is not practical for our purposes.

The 1-norm is also referred to Manhattan distance or taxicab norm (since a taxi plying on rectangular parallel roads such as in Manhattan cannot reach another point on a different street by taking the shortest route to it by going through lots, blocks, buildings or cuts through roads). See Figure 3.3 for a graphical illustration of both norms.

### 3.4.3 Distance vs. Displacement



For the purposes of this thesis however, we shall restrict ourselves to the actual road distances between cities as represented on the Central Region section on the map of Ghana.

In figure 3.3, the two cities A and B are linked by a road represented by a solid line from A to B. The distance traveled by a vehicle using the road is then represented by a broken line along the road. The displacement however represents the direct link calculated by the 1 norm, which is not representative of the practical distance travel. Candidate methods for this are repetitive use of one-to-all shortest path algorithms such as Dijkstra's algorithm, use of all-to-all shortest path algorithms such as the Floyd-Warshall algorithm, and use of specifically designed some-to-some shortest path algorithms (Kim, ).

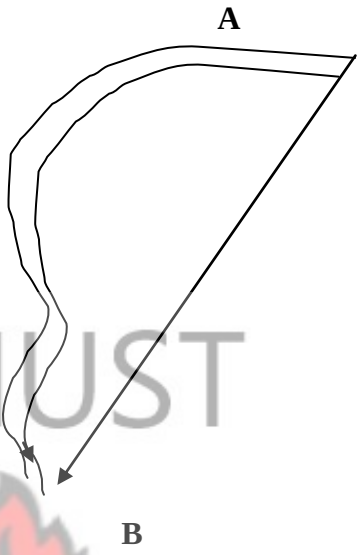


**Figure 3.3: Displacement Vs Distance.**

### 3.4.4 Connectivity of Cities

In the general TSP there is the assumption that it is possible to reach any and all of the cities from any city. In the practical application to our particular problem however, the main means of travel is assumed to be by road and as such only cities with direct road links will be considered to have a distance between them. Thus in figure 3.4 below, where as there are direct links between A and all other cities, there are no links between B and E or D since all connections should be through other cities.

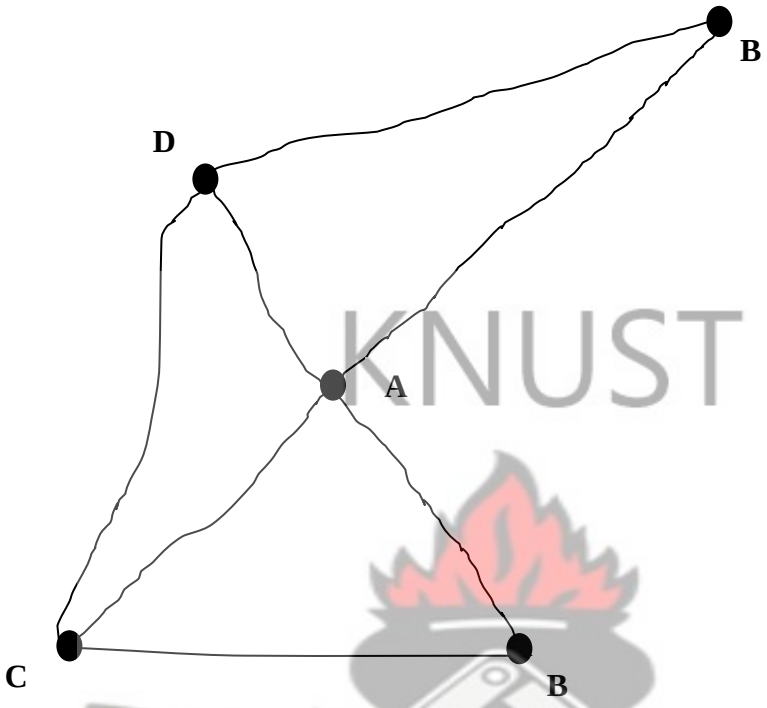


**Figure 3.4: Sample connection of cities.**

### 3.4.5 The Objective Function

The object function, which is represented as the total distance covered by a tour is not a function of the coordinates of the cities but rather the length of the roads linking the cities

$$D = min \sum_{i=1}^{n} \sum_{j=1}^{n} x(i,j) I(i,j)$$

Constrained by

$$\sum_{i=1}^{n} x(i,j) = 1, i = 1,2, \dots, n$$

$$\sum_{j=1} x(i,j) = 1, j = 1,2, \dots, n$$

*Where D is the total distance for the tour,*

$I_{i,j}$ *is the length or distance between cities i and j.*

### 3.4.6 The Configuration

The constituencies are numbered from 1 to 23 and the various distances between the constituencies with direct road links modeled into a 23 X 23 matrix with elements representing the length of the road between the constituencies. As expected all elements on the diagonal are zero since these represents the distances between the same constituencies. Thus $I_{i,i} = I_{j,j} = 0$ . For any symmetric TSP, $I_{i,j} = I_{j,i}$. The configuration is therefore given by a random arrangement of the constituencies respecting the road link.

### 3.4.7 Generating the Configuration

The configuration is generating from the matrix that models the network of roads linking the constituencies. The procedure is given by:

(i) Starting from any non-zero element $I_{i,j}$ in the matrix and noting the column $j$ and row $i$ as the starting leg of the tour.

(ii) Move from the $I_{i,j}$ elements to another element $I_{i,k}$ or $I_{k,j}$ also containing a non-zero element. K being a column or row respectively.

(iii) Repeat step 1 to 2 keeping the recently identified row or column and move to the next element until you come back to an element in the starting row or column.

(iv) Remembering that there are exactly $n$ links for any $n$ cities in any TSP, sum the $n$ elements identified in the tour.

### 3.4.8 Method of Solution of TSP

The method used to solve the TSP is based on a simulated annealing process which is performed on tour distances generated from actual distances traveled be the aspirants. The following section gives the details of how the tour distances were generated.



#### 3.4.8.1 The Tour Distance Pseudo code

The distances between the constituencies are put into a matrix as shown in table 3.3.4 for the road network shown in figure 3.4. The elements in the matrix then represent the distances between the constituencies in the row and column. The tour distances generated by following steps:

Input : Matrix of city links lengths.

- Initialize : length = 0, order = []
- Select $I_{i,j}$ from matrix and set $I = I_{i,j}$
- For $e = 1$ to $n$ do
    - $length = length + I_{i,j}, order = j, i$
    - $if\ e\ is\ odd, do$
        - Set all elements of column $i$ and $j$ and all row elements of row $j$ to zero
        - Select $I_{i,j}$ and set $I = I_{i,j}$
    - else

- Set all elements of rows $i$ and $j$ and all column elements of column $j$ to zero
- Select $I_{i,j}$ and set $I = I_{i,j}$

- *end if loop*
- *end for loop*
- *print length order.*

*Output*: *total tour length, order of tour.*



**3.4.8.1.1 An Example of the Implementation of the Tour Distance Algorithm**

Consider the network of roads in figure 3.4 joining the six cities represented as A, B, C, D, E and F with corresponding distances between cities.

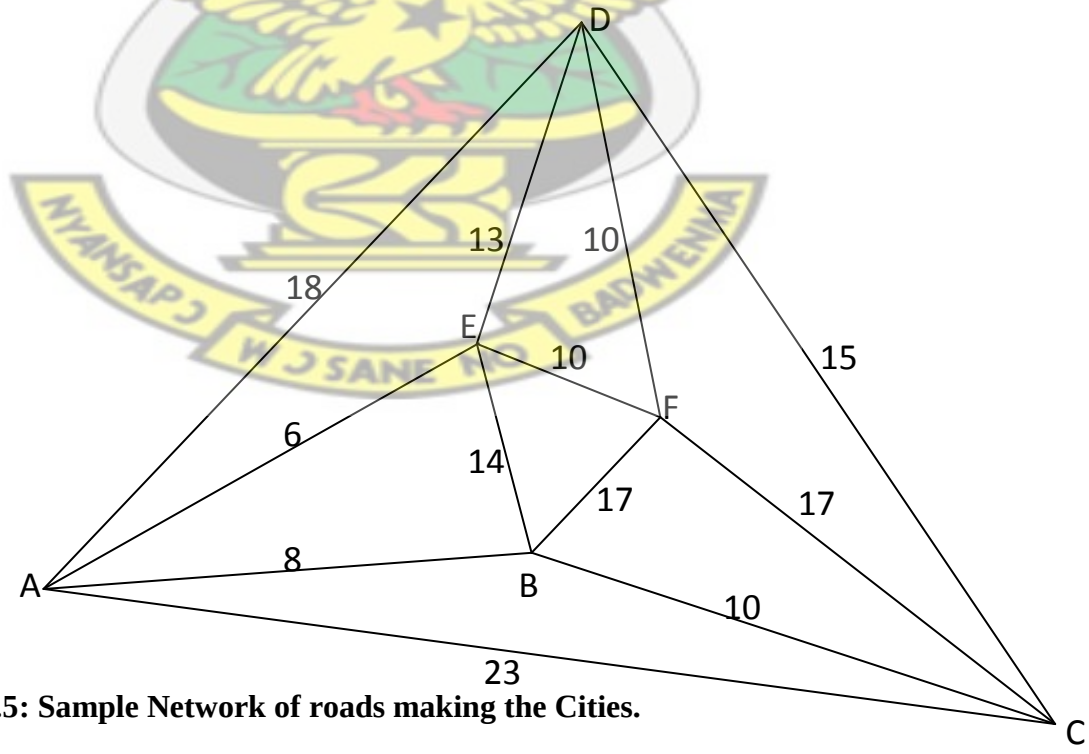


**Figure 3.5: Sample Network of roads making the Cities.**

The network can be represented in the 6x6 matrix as found in table 3. 6 below.

**Table 3.6: 6x6 Distance Matrix 1**

| | A | B | C | D | E | F |
|---|---|---|---|---|---|---|
| A | 0 | 8 | 23 | 18 | 6 | ∞ |
| B | 8 | 0 | 10 | ∞ | 14 | 17 |
| C | 23 | 10 | 0 | 15 | ∞ | 17 |
| D | 18 | ∞ | 15 | 0 | 13 | 10 |
| E | 6 | 14 | ∞ | 13 | 0 | 10 |
| F | ∞ | 17 | 17 | 10 | 10 | 0 |



The algorithm is thus performed on the matrix in table 3.7 as follows:

- The choice is made to start the tour from point B (home city) to visit each city exactly once before returning to point B (home city).
- The length between point B and A (8) is stored as shown in table 3.3.6 and all connections from B are broken by setting all elements of the B row to zero. All elements in the A column are also set to zero. All elements in column B are set to zero (only shown and bolded in the table below to be a reminder that it is stored).

**Table 3.7: The Distance Algorithm 1**

↓

| | A | B | C | D | E | F |
|---|---|---|---|---|---|---|
| A | 0 | **8** | 23 | 18 | 6 | ∞ |
| B | 0 | 0 | 0 | 0 | 0 | 0 |
| C | 0 | 0 | 0 | 15 | ∞ | 17 |
| D | 0 | 0 | 15 | 0 | 13 | 10 |
| E | 0 | 0 | ∞ | 13 | 0 | 10 |
| F | 0 | 0 | 17 | 10 | 10 | 0 |



Move on the row to another city E and store the distance AE (6) and set all elements of row E to zero as shown in table 3.8.

**Table 3.8: The Distance Algorithm 2**

| | A | B | C | D | E | F |
|---|---|---|---|---|---|---|
| A | 0 | **8** | 23 | 18 | **6** | ∞ |
| B | 0 | 0 | 0 | 0 | 0 | 0 |
| C | 0 | 0 | 0 | 15 | ∞ | 17 |
| D | 0 | 0 | 15 | 0 | 13 | 10 |
| E | 0 | 0 | 0 | 0 | 0 | 0 |
| F | 0 | 0 | 17 | 10 | 10 | 0 |

Move through column E to row D and store distance ED (13) and set all elements in row E to zero as shown in Table 3.9.

**Table 3.9: The Distance Algorithm 3**

| | A | B | C | D | E | F |
|---|---|---|---|---|---|---|
| A | 0 | **8** | **23** | 0 | **6** | ∞ |
| B | 0 | 0 | 0 | 0 | 0 | 0 |
| C | 0 | 0 | 0 | 0 | ∞ | 17 |
| D | 0 | 0 | 15 | 0 | **13** | 10 |
| E | 0 | 0 | 0 | 0 | 0 | 0 |
| F | 0 | 0 | 17 | 0 | 10 | 0 |



Now from element ED (13), move along row D to any non-zero element in that row. The only non- zero element is 10 in column F as shown in Table 3.10.

**Table 3.10: The Distance Algorithm 4**

| | A | B | C | D | E | F |
|---|---|---|---|---|---|---|
| A | 0 | **8** | **23** | 0 | **6** | ∞ |
| B | 0 | 0 | 0 | 0 | 0 | 0 |
| C | 0 | 0 | 0 | 0 | ∞ | 17 |
| D | 0 | 0 | 0 | 0 | **13** | **10** |
| E | 0 | 0 | 0 | 0 | 0 | 0 |
| F | 0 | 0 | 0 | 0 | 0 | 0 |

Similarly, we move in column F from element 10 to the only non- zero element in the column 17 in row C. This is illustrated in Table 3.11.

**Table 3.11. The Distance Algorithm 5**

| | A | B | C | D | E | F |
|---|---|---|---|---|---|---|
| A | 0 | **8** | **23** | 0 | **6** | ∞ |
| B | 0 | 0 | 0 | 0 | 0 | 0 |
| C | 0 | 0 | 0 | 0 | ∞ | **17** |
| D | 0 | 0 | 0 | 0 | **13** | **10** |
| E | 0 | 0 | 0 | 0 | 0 | 0 |
| F | 0 | 0 | 0 | 0 | 0 | 0 |



Elements of column B (the starting point or the home city) which was set to zero are then replaced so the algorithm can return to the home city. Table 3.12 shows the update of Table 3.11 with elements of column B replaced.

**Table 3.12: The Distance Algorithm 6**

| | A | B | C | D | E | F |
|---|---|---|---|---|---|---|
| A | 0 | **8** | **23** | 0 | **6** | ∞ |
| B | 0 | 0 | 0 | 0 | 0 | 0 |
| C | 0 | 10 | 0 | 0 | ∞ | **17** |
| D | 0 | ∞ | 0 | 0 | **13** | **10** |
| E | 0 | 14 | 0 | 0 | 0 | 0 |
| F | 0 | 17 | 0 | 0 | 0 | 0 |

The last move is then made along row C from 17 in column F. The move ends up on element 10 on row C. Now element 10 on row C is also found on column B (the starting point) and this marks the end of the tour. Table 3.13 shows the complete tour (represented by the lines joining cities in the rows and columns with the distances between the cities shown as elements in the matrix).

**Table 3.13: The Distance Algorithm 7**

| | A | B | C | D | E | F |
|---|---|---|---|---|---|---|
| A | 0 | **8** | **23** | 0 | **6** | 0 |
| B | 0 | 0 | 0 | 0 | 0 | 0 |
| C | 0 | **10** | 0 | 0 | 0 | **17** |
| D | 0 | 0 | 0 | 0 | **13** | **10** |
| E | 0 | 0 | 0 | 0 | 0 | 0 |
| F | 0 | 0 | 0 | 0 | 0 | 0 |



The sum of the distances between the cities visited is the total tour length. Table 3.14 shows the order or configuration of the tour and distances traveled and total tour distances.

**Table 3.14: Examples of Solution 1**

| | Configuration | Distances | Total Tour Distance |
|---|---|---|---|
| 1 | B-A-E-D-F-C-B | 8+6+13+10+17+10 | **64** |
| 2 | B-A-C-D-E-F-B | 8+23+15+13+10+17 | **86** |
| 3 | B-A-D-C-F-E-B | 8+18+15+17+10+14 | **82** |
| 4 | B-A-F-E-D-C-B | 8+17+10+13+15+23 | **86** |

### 3.4.9 The Simulated Annealing Algorithm

The algorithm implemented for the simulated annealing is as specified by the flow chart of Figure 3.5. The input of solution is generated by the distance algorithm. The code for the implementation is Appendix A.

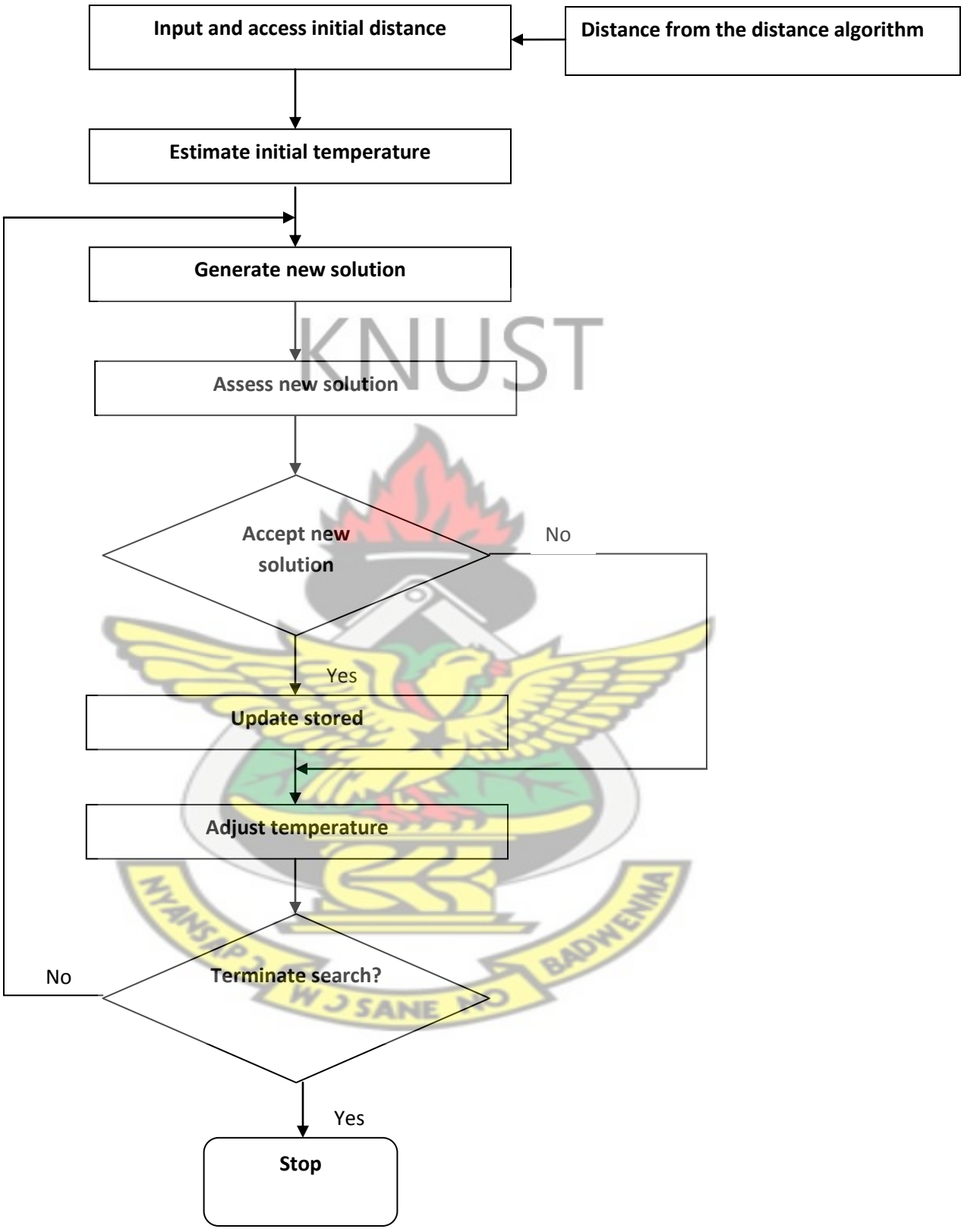


**Figure 3.6 SA Algorithm flow chart.**

### 3.4.9.1 Pseudo Code

Given an optimization problem, we put it in the form $f(x)$ such that $x \in S,\ S = feasible\ solution.$

The basic SA algorithm is detailed below with the following parameter identification:

$x^{(1)} = Configuration.$

$D(x) = Objective\ function$

$k = Iteration\ number$

$\delta = d(x^1) - d(x^0)\ (Energy\ change\ between\ states\ x^1\ and\ x^0)$

$T = Control\ parameter\ (Temperature\ of\ system)$

$g(T) = Control\ parameter\ function\ (Temperature\ function)$

$e^{-(\delta/T_k)} = Choice\ probability\ function\ (Boltzmann\ probability\ function).$

It provides the condition under which a non- improvement solution is not discarded.

Step 1:

(i) Select an initial solution $x^{(0)}$ evaluated by the distance algorithm based on the objective function and assign $x^{(b)} = x^{(0)}$

(ii) Set $k = 0$, select an initial temperature (control parameter)

$T_k = T_0\ for\ k = 0\ assign\ T_b = T_0.$

(iii) Select a temperature function $g(T_k)$

Step 2:

Choose a solution $x^{(1)}\ in\ N(x^{(0)}\ and\ compute\ \delta = d\left(x^{(1)}\right) - d\left(x^{(0)}\right)$

Step 3:

$$If\ \delta = 0\ or\ \left[\delta < 0\ and\ e^{-(\delta/T_k)} \geq\ \theta: \theta \leftarrow U = (0,1)\right],$$

$accept\ the\ new\ solution\ x^{(1)}.$

$Assign\ x^{(0)} \leftarrow x^{(1)}\ and\ keep\ the\ \ new\ x^{(0)}\ such\ \ that\ \ x^{(0)} =\ x^{(b)}\ set\ T_b =\ T_k.$

Step 4:

If some stopping criteria are satisfied, stop.

Step 5:

Update the temperature $T_{k+1} = g(T_k) \leq T_k\ and\ set\ k = k + 1.$

**3.4.9.2 Application of SA to an example.**

The starting iteration $k = 0$

(i) The initial solution is randomly chosen from table 3.14 as

$x^{(0)} = E - A - D - C - B - F - E\ with\ a\ d(x^{(0)}) = 76.\ Assign\ T_0$

$= 20\ and\ in$

$T_0 = \alpha T,\ \alpha$ = 0.5, stopping at $T\ <\ 0.1.$

(ii) Compute a new $d(x^{(1)}) = 86\ with\ configuration$

$x^{(1)} = A - B - F - E - D - C - A$

(iii) Compute $\delta = d(x^{(1)}) - d(x^{(0)}) = 86 - 76 = 10.$

(iv) Since $\delta = 10 > 0$, we test whether or not to discard the non- improvement solution.

(v) We compute $m = e^{-(\delta/T_k)} = e^{-(10/20)} = 0.6065$

(vi) We generate the random number $\theta = 0.7187$ since $m < \theta$ we retain the initial solution.

(vii) Since the stopping criterion is not met, move to the next step.

(viii) Update the temperature get $T_1 = \alpha T_0 = 0.5\ (20) = 10\ and\ iterations\ k = k + 1 = 0 + 1 = 1.$

The second iteration $K = 1$

(i) Maintain initial solution.

(ii) Choose a new random solution $d(x^{(2)}) = 64$ with configuration

$x^{(2)} = B - A - E - D - F - C - B$

(iii) Compute $\delta = d(x^{(2)}) - d(x^{(0)}) = 64 - 76 = -12\ since\ \delta = -12 < 0,\ x^{(2)} = 64$ is an improved solution.

(iv) Set $x^{(0)} \leftarrow x^{(2)} = B - A - E - D - F - C - B\ also\ set.$

(v) Update temperature $T_2 = \alpha T_1 = 0.5\ (10) = 5\ and\ iterations\ k = k + 1 = 1 + 1 = 2$

We continue the iterations until the stopping condition is met. For this example the solution is met.

New solution adapted is $x^{(0)} = B - A - E - D - F - C - B$

Choose a new random solution $d(x^{(3)}) = 59\ with\ configuration\ x^{(1)} = E - A - B - C - D - F - E$ with the following steps:

Compute $\delta = d(x^{(1)}) - d(x^{(0)}) = 59 - 64 = -5\ since\ \delta = -5 < 0,\ x^{(1)} = 59\ is\ an\ improved$ solution.

This cannot be improved further till the stopping criterion is met.

# CHAPTER 4

# COLLECTION OF DATA, ANALYSIS OF DATA AND RESULTS

## 4.0 Introduction

In this chapter, we shall present data collection, data analysis and the results.

## 4.1 Numerical representation of constituency capitals

Each of the twenty three constituency capitals in Central Region below has been assigned numbers for the purpose of this research work. This is illustrated in the table below.

**Table 4.1: Numbers assigned to constituency capitals in the Central Region.**

| Constituency capital | Number assigned |
|---|---|
| Cape Coast(Old Hospital Hill) | 1 |
| Elmina | 2 |
| Saltpond | 3 |
| Abura Dunkwa | 4 |
| Nsuaem Kyekyewere | 5 |
| Essarkyir | 6 |
| Ajumako | 7 |
| Jukwa | 8 |
| Apam | 9 |
| Twifo Praso | 10 |
| Asikuma | 11 |
| Assin Foso | 12 |
| Afransi | 13 |

| | |
|---|---|
| Winneba | 14 |
| Agona Swedru | 15 |
| Awutu Breku | 16 |
| Kasoa | 17 |
| Agona Nsaba | 18 |
| Dunkwa-On-Offin | 19 |
| Diaso | 20 |
| Potsin | 21 |
| Assin Breku | 22 |
| Cape Coast(Abura) | 23 |



**4.2 Distance Matrix for the 23 Constituency Capitals in Central Region in kilometres (km).**

The table below shows the distance matrix obtained from distances between the capitals of the twenty-three constituencies. For cities without direct link, the minimum distance along the edges is considered. The cells indicating zero shows that there is no distance.

$C_{ij}$= The distance from city $i$ to city $j$

$C_{ii} = C_{jj} = 0$ =There is no distance.

**Table 4.2: Distance Matrix for the 23 Constituency capitals in Central region in kilometers(km)**

| $C_{ij}$ | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 20 | 21 | 22 | 23 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0 | 15 | 25 | 29 | 45 | 53 | 55 | 55 | 70 | 70 | 75 | 77 | 84 | 85 | 97 | 105 | 110 | 113 | 135 | 200 | 97 | 95 | 5.2 |
| 2 | 15 | 0 | 40 | 44 | 60 | 68 | 70 | 70 | 85 | 85 | 90 | 92 | 99 | 100 | 112 | 120 | 125 | 128 | 150 | 215 | 112 | 110 | 17.2 |
| 3 | 25 | 40 | 0 | 19 | 59 | 28 | 30 | 80 | 45 | 95 | 50 | 56 | 85 | 60 | 67 | 80 | 85 | 88 | 160 | 225 | 72 | 78 | 30.2 |
| 4 | 29 | 44 | 19 | 0 | 16 | 47 | 49 | 73 | 64 | 88 | 69 | 48 | 78 | 79 | 91 | 99 | 104 | 78 | 138 | 203 | 91 | 66 | 34.2 |
| 5 | 45 | 60 | 59 | 16 | 0 | 63 | 65 | 100 | 80 | 115 | 100 | 32 | 120 | 125 | 117 | 115 | 120 | 123 | 180 | 245 | 135 | 50 | 50.2 |
| 6 | 53 | 68 | 28 | 47 | 63 | 0 | 58 | 108 | 17 | 123 | 106 | 140 | 57 | 32 | 44 | 52 | 57 | 60 | 188 | 253 | 44 | 138 | 150 |
| 7 | 55 | 70 | 30 | 49 | 65 | 58 | 0 | 110 | 75 | 125 | 20 | 97 | 29 | 54 | 42 | 74 | 79 | 58 | 190 | 255 | 66 | 155 | 59.2 |
| 8 | 55 | 70 | 80 | 73 | 100 | 108 | 110 | 0 | 125 | 15 | 125 | 32 | 139 | 140 | 145 | 160 | 165 | 159 | 80 | 145 | 152 | 42 | 49.8 |
| 9 | 70 | 85 | 45 | 64 | 80 | 17 | 75 | 125 | 0 | 140 | 51 | 102 | 40 | 15 | 27 | 35 | 40 | 43 | 205 | 270 | 27 | 125 | 75.2 |
| 10 | 70 | 85 | 95 | 88 | 115 | 123 | 125 | 15 | 140 | 0 | 145 | 28 | 154 | 155 | 167 | 175 | 180 | 183 | 65 | 130 | 165 | 43 | 75.2 |
| 11 | 75 | 90 | 50 | 69 | 100 | 106 | 20 | 125 | 51 | 145 | 0 | 117 | 9 | 74 | 62 | 54 | 59 | 38 | 210 | 275 | 46 | 162 | 78.2 |
| 12 | 77 | 92 | 56 | 48 | 32 | 140 | 97 | 32 | 102 | 28 | 117 | 0 | 176 | 155 | 167 | 175 | 180 | 173 | 90 | 155 | 167 | 18 | 82.2 |
| 13 | 84 | 99 | 85 | 78 | 120 | 57 | 29 | 139 | 40 | 154 | 9 | 176 | 0 | 25 | 13 | 45 | 50 | 29 | 204 | 169 | 37 | 200 | 87.2 |
| 14 | 85 | 100 | 60 | 79 | 125 | 32 | 54 | 140 | 15 | 155 | 74 | 155 | 25 | 0 | 12 | 20 | 25 | 28 | 220 | 285 | 12 | 173 | 90.2 |
| 15 | 97 | 112 | 67 | 91 | 117 | 44 | 42 | 145 | 27 | 167 | 62 | 167 | 13 | 12 | 0 | 32 | 37 | 16 | 232 | 297 | 24 | 185 | 100 |
| 16 | 105 | 120 | 80 | 99 | 115 | 52 | 74 | 160 | 35 | 175 | 54 | 175 | 45 | 20 | 32 | 0 | 5 | 48 | 240 | 305 | 8 | 193 | 110 |
| 17 | 110 | 125 | 85 | 104 | 120 | 57 | 79 | 165 | 40 | 180 | 59 | 180 | 50 | 25 | 37 | 5 | 0 | 53 | 230 | 310 | 13 | 198 | 115 |
| 18 | 113 | 128 | 88 | 78 | 123 | 60 | 58 | 159 | 43 | 183 | 38 | 173 | 29 | 28 | 16 | 48 | 53 | 0 | 248 | 313 | 40 | 213 | 118 |
| 19 | 135 | 150 | 160 | 138 | 180 | 188 | 190 | 80 | 205 | 65 | 210 | 90 | 204 | 220 | 232 | 240 | 230 | 248 | 0 | 65 | 220 | 108 | 137 |
| 20 | 200 | 215 | 225 | 203 | 245 | 253 | 255 | 145 | 270 | 130 | 275 | 155 | 169 | 285 | 297 | 305 | 310 | 313 | 65 | 0 | 289 | 173 | 202 |
| 21 | 97 | 112 | 72 | 91 | 135 | 44 | 66 | 152 | 27 | 165 | 46 | 167 | 37 | 12 | 24 | 8 | 13 | 40 | 220 | 289 | 0 | 177 | 102 |
| 22 | 95 | 110 | 78 | 66 | 50 | 138 | 115 | 42 | 125 | 43 | 162 | 18 | 200 | 173 | 185 | 193 | 198 | 213 | 108 | 173 | 177 | 0 | 100 |
| 23 | 5.2 | 17.2 | 30.2 | 34.2 | 50.2 | 150 | 59.2 | 49.8 | 75.2 | 75.2 | 78.2 | 82.2 | 87.2 | 90.2 | 100 | 110 | 115 | 118 | 137 | 202 | 102 | 100 | 0 |

### 4.3 Formulation of the 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 $d_{ij}$ for $(i, j)$. We focus on the symmetric TSP case in which $C_{ij} = C_{ji,}$ for all $(i, j)$.

We formulate this minimization problem as an integer programming, as shown in Equations (1) to (5)



$$P1: min \sum_{i \in v} \sum_{j \in v} c_{ij} x_{ij} \qquad (1)$$

Subject to

$$\sum_{\substack{j \in v \\ j \neq i}} x_{ij} = 1 \qquad i \in v \qquad (2)$$

$$\sum_{\substack{i \in v \\ i \neq j}} x_{ij} = 1 \qquad j \in v \qquad (3)$$

$$\sum_{i \in s} \sum_{j \in s} x_{ij} \leq |s| - 1 \qquad \forall s \subset v, s \neq \emptyset \; x_{ij} = 0 \; or \; 1 \qquad i, j \in v \qquad (4)$$

$$x_{ij} = 0 \; or \; 1 \qquad i, j \in v \qquad (5)$$

The problem is an assignment problem with additional restrictions that guarantee the exclusion of sub tours in the optimal solution. Recall that a sub tour in $V$ is a cycle that does not include all vertices (or cities). Equation (1) is the objective function, which minimizes the total distance to be traveled.

Constraints (2) and (3) define a regular assignment problem, where (2) ensures that each city is entered from only one other city, while (3) ensures that each city is only departed onto other city. Constraint (4) eliminates sub tours. Constraint (5) is a binary constraint, where $x_{ij} = 1$ if edge $(i, j)$ in the solution and $x_{ij} = 0$, otherwise.



**4.4 Analysis**

To satisfy the constraints (2) and (3) we choose the random

Initial tour $(x^0) = 22 - 2 - 5 - 6 - 17 - 8 - 7 - 22 - 10 - 9 - 11 - 20 - 12 - 21 - 13 - 15 - 14 - 23 - 18 - 4 - 3 - 1 - 19 - 22$

From objective function (1) the initial distance $= d(x^0) = \mathrm{d}(22,2) + \mathrm{d}(2,5) + \mathrm{d}(5,6) + \mathrm{d}(6,17) + \mathrm{d}(17,8) + \mathrm{d}(8,7) + \mathrm{d}(7,22) + \mathrm{d}(22,10) + \mathrm{d}(10,9) + \mathrm{d}(9,11) + \mathrm{d}(11,20) + \mathrm{d}(20,12) + \mathrm{d}(12,21) + \mathrm{d}(21,13) + \mathrm{d}(13,15) + \mathrm{d}(15,14) + \mathrm{d}(14,23) + \mathrm{d}(23,18) + \mathrm{d}(18,4) + \mathrm{d}(4,3) + \mathrm{d}(3,1) + \mathrm{d}(1,19) + \mathrm{d}(19,22) = 2031.2\mathrm{km}$

The initial temperature is taken to be $(T_o) = 4069.00$, $\alpha = 0.99$

Temperature is updated by using the formula $T_{k+1} = \alpha T_k$ where k is the number of iteration.

Stop when $T \leq 42.03$

Simulated annealing algorithm was used to obtain the final solution. Probook hp laptop computer (CORE i3) was used in finding the solution after 1339 iterations in 102.367856 seconds. The execution time varied with the number of iterations.

**4.5 Results**

After performing 1339 iterations, the optimal tour= $2-6-9-14-21-16-17-15-18-13-11-7-3-12-19-20-10-8-22-5-4-23-1-2$

Thus,

$$= d(2,6) + d(6,9) + d(9,14) + d(14,21) + d(21,16) + d(16,17) + d(17,15) + d(15,18) + d(18,13) + d(13,11) + d(11,7) + d(7,3) + d(3,12) + d(12,19) + d(19,20) + d(20,10) + d(10,8) + d(8,22) + d(22,5) + d(5,4) + d(4,23) + d(23,1) + d(1,2) = 786\text{km}$$

There was no change in the last ten iterations for the optimal tour.

All things being equal, the optimal tour is therefore as follows:

Elmina→ Essarkyir→ Apam→ Winneba→ Potsin→ Awutu Breku→ Kasoa→ Agona Swedru→ Agona Nsaba→ Afransi→ Asikuma→ Ajumako→ Saltpond→ Assin Foso→ Dunkwa-on-Offin → Diaso → Twifo Praso → Jukwa → Assin Breku → Nsuaem Kyekyewere → Abura Dunkwa → Abura(Cape Coast) → Old Hospital Hill(Cape Coast) →Elmina

# CHAPTER 5

# CONCLUSION AND RECOMMENDATION

## 5.0 Introduction

In this chapter, we shall present the conclusion and recommendation of the study.

## 5.1 Conclusion

Simulated annealing is a heuristic- based search algorithm, motivated by an analogy of physical annealing in solids. It is capable of solving combinatorial optimization problem. The method of simulated annealing has been used to find the global minima cost configuration for NP- complete problems with many local minima.

The Simulated annealing algorithm can be a useful tool which is applied to hard combinatorial problems like the TSP. Using simulated annealing as a method in solving the symmetric TSP model has proved that it is possible to converge to the best solution.

We conclude that the objective of finding the minimum tour from the symmetric TSP model by the use of simulated annealing algorithm was successfully achieved. The study shows clearly that, any presidential aspirant who visits the Central Region must visit the constituencies in the order below to minimize cost.

The order is as follows:

Elmina → Essarkyir → Apam → Winneba → Potsin → Awutu Breku → Kasoa → Agona Swedru → Agona Nsaba → Afransi → Asikuma → Ajumako → Saltpond → Assin Foso → Dunkwa-on-Offin → Diaso → Twifo Praso → Jukwa → Assin Breku → Nsuaem Kyekyewere → Abura Dunkwa → Abura(Cape Coast) → Old Hospital Hill(Cape Coast) →Elmina

## 5.2 Recommendations

After a comprehensive study of TSP and Simulated annealing algorithm, the following recommendations should be considered.

(i) Presidential Candidates who visit the Central Region to campaign should consider the routes below in order to minimize cost.

Elmina→ Essarkyir→ Apam→ Winneba→ Potsin→ Awutu Breku→ Kasoa→ Agona Swedru→ Agona Nsaba→ Afransi→ Asikuma→ Ajumako→ Saltpond→ Assin Foso→ Dunkwa-on-Offin → Diaso → Twifo Praso → Jukwa → Assin Breku → Nsuaem Kyekyewere → Abura Dunkwa → Abura(Cape Coast) → Old Hospital Hill(Cape Coast) →Elmina

(ii) Students can use this work for further research covering all key towns in the Central Region of Ghana.

## REFERENCES


1. Aarts, E. H. L and Korst, J. (1989). *Simulated Annealing and Boltzmann Machines*. John Wiley Sons, 1989. Reprinted February 1990.
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.
3. Amponsah, S.K and Darkwah, F.K (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., Chv'atal, V., and Cook, W. (1999). "Finding Tours in the TSP". Technical Report 99885. Research Institute for Discrete Mathematics, Universitaet Bonn: Bonn, Germany.
6. Applegate D., Bixby R., Chvatal , V., and 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., Chvatal V and Cook W. (1998). On the Solution of the Traveling Salesman Problems. Documenta Mathematica-Extra Volume ICM, chapter 3, pp. 645-656.
8. Applegate, D., Bixby, R., Chv'atal, V. and Cook, W. (2007). *The Traveling Salesman Problem*. Princeton University Press: Princeton, NJ.

9. 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.
10. 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.
11. Barachet, L.L. (1957). "Graphic Solution of the Traveling Salesman Problem". *Operations Research*. 5:841-845.
12. Barahona, F., Grötschel, M., Jünger, M., and Reinelt, G. (1988): "An application of combinatorial optimization to statistical physics and circuit layout design", Operations Research 36(3), 493-513.
13. Beasley, J. E. A heuristic for Euclidean and rectilinear steiner problems. *European Journal of Operational Research*, 58:284-292, 1992.
14. Bellman, R. (1960). "Combinatorial Process and Dynamic Programming". In: *Combinatorial Analysis*. R. Bellman and M. Hall, Jr., eds. American Mathematical Society: Washington, DC. 217-249.
15. Bellman, R. (1960). "Dynamic Programming Treatment of the TSP". *Journal of Association of Computing Machinery*. 9:66.
16. Bellmore, M. and Nemhauser, G.L. (1968). "The Traveling Salesman Problem: A survey". *Operations Research*. 16:538-558.
17. 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.

18. Biezen, I.V.(2003). *Political Parties in New Democracies: Party Organization in Southern and East-Central Europe*. London: Palgrave Macmillan.
19. Boeres, M.C.S, De Carvalho, L.A.V. and Barbosa, V.C, "A faster elastic-net algorithm for the traveling salesman problem, " in Proc. Int. Joint Conf. on Neutral Networks, IEEE, Piscataway, NJ, 1992, II:215-220.
20. 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.
21. Bonomi, E., and Lutton, J.L. (1984), "The N-city travelling salesman problem: Statistical mechanics and the Metropolis algorithm", *SIAM Review* 26, 551-568.
22. 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.
23. Carpaneto, G., Toth, P., (1980). "Some new branching and bounding criteria for the asymmetric traveling salesman problem", Management Science 21. pp. 736-743.
24. 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.
25. Cerny, V. (1985) "Thermodynamical Approach to the Traveling Salesman Problem: An Efficient Simulation Algorithm", *J. Opt. Theory Appl*., 45, 1, 41-51.

26. Christofides, N. (1970). "The shortest Hamiltonian chain of a graph", *SIAM Journal on Applied Mathematics* 19,689-696.
27. 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.
28. Croes, G.A. 1958. "A Method for Solving Travelling-Salesman Problems". Operations Research. 6:791-812.
29. Crowder, H. and Padberg, M.W. (1980). "Solving Large-Scale Symmetric Travelling Salesman Problems to Optimality". Management Science. 26:495-509.
30. Dacey, M.F. 1960. "Selection of an Initial Solution for the Traveling-Salesman Problem". Operations Research. 8:133-134. Darwin. C, (1859). *On the origin of species*, first edition (facsimile-1964), Harvard University Press, MA.
31. Dantzig, G.B., Fulkerson, D.R., and Johnson, S.M. (1954). "Solution of a Large-Scale Traveling-Salesman Problem". Operations Research. 2:393-410.
32. Datta, S, (2000). Application of operational Research to the Transportation Problems in Developing Countries: A review, Global Business Review, Volume 18, No. 1 pages 113-132.
33. David Goldberg, (1989). Genetic Algorithms in Search, Optimization, and Machine Learning.
34. Dawkins, R, (1986). The Blind Watchmaker, Penguin, London.
35. Eastman, W.L. (1958). "Linear Programming with Pattern Constraints". Ph.D. Dissertation. Harvard University: Cambridge, MA.

36. Ferreir, J.V. (1995). “A Travelling Salesman Model for the Sequencing of Duties in Bus Crew Rotas”. Journal of Operational Research Society (UK). 46(4): 415-426.
37. Fischetti, M., Lodi, A., Toth, P., (2002). Exact Methods for the Asymmetric Traveling Salesman Problem. In: Gutin, G., Punnen, A.P. (Eds.), The Traveling Salesman Problem and its Variations. Kluwer, Dordrecht, pp. 169-194 (Chapter 9).
38. Fischetti, M. and Toth, P., 1992, An additive bounding procedure for the asymmetric travelling salesman problem. Math. Program., 53, 173-197.
39. Fleischer, M.A. (1999) Generalized cybernetic optimization: solving continuous variable problems. In: S. Voss, S. Martello, C. Roucairol, H. Ibrahim, and I.H. Osman (eds.), Meta-heuristics: Advances and Trends in Local Search Paradigms for Optimization, Kluwer Academic Publishers, pp. 403-418.
40. Flood, M.M. (1956). “The TSP”. Operation Research. 4:6.
41. Gambardella, L.M. and Dorigo, M., (1995), Ant-Q: a reinforcement learning approach to the traveling salesman problem, in:Proc. ML-95, 12th Int. Conf. on Machine Learning, A. Prieditis and S. Russell (eds.) (Morgan Kaufmann, San Francisco, CA) pp.252-260.
42. Gerard Reinelt, (1994). The Traveling Salesman: Computational Solutions for TSP Applications. Springer-Verlag.
43. Goel, A., Gruhn V. (2006). *General Vehicle Routing Problem*. European Journal of Operational Research, Elsevier Science. November (2006).

44. Golden B.L. and Assad A.A., (1988). Vehicle Routing: Methods and Studies, Elsevier Science, Amsterdam.
45. Goldberg, D.E, (1989). Genetic Algorithm for Search, Optimization and Machine Learning, Addison-Wesley.
46. Golden B.L., Wasil E.A., Kelly J.P, and Chao I-M, (1998). Metaheuristics in Vehicle Routing. In Fleet Management and Logistics, T.G. Crainic and G. Laporte (eds), Kluwer, Boston, 33-56.
47. Gomory, R.E. (1996). "The Traveling Salesman Problem". In: Proceedings of the IBM Scientific Computing Symposium on Combinatorial Problems. IBM: White Plains, NY. 93-121.
48. Gomory, R.E. (1960). Solving linear programming problems in integers. In Bellman, R., Hall Jr., M. (eds) Combinatorial Analysis: Proceedings of Symposia in Applied Mathematics X. American Mathematical Society, Providence, Rhode Island, pp. 211-215.
49. Gonzales, R.H. (1962). "Solution to the Traveling Salesman Problem by Dynamic Programming on the Hypercube". Technical Report Number 18, Operations Research Centre, Massachusetts Institute of Technology: Boston, MA.
50. Goss, S., Aron, S., Deneubourg, J.L. and Pastels, J.M., (1989). Self-organized shortcuts in the Argentine ant. Naturwis-senschaften, 76, 579-581.
51. Grötschel, M., Jünger, M., and Reinelt, G. (1984): 'A cutting plane algorithm for the linear ordering problem', Operations Research 32, 1195.

52. Grötschel, M., and Holland, (1991): "Solution of large-scale travelling salesman problems", Mathematical Programming 51(2), 141-202.
53. Grötschel, M., Martin, A., and Weismantel, R. (1996),: 'Packing Steiner trees: a cutting plane algorithm and computational results', Mathematical Programming 72, 125-145.
54. Grötschel, M. (1980). "On the Symmetric Traveling Salesman Problem: Solution of a 120-City Problem". Mathematical Programming Study. 12: 61-77.
55. Grötschel, M and Padberg M., (1993). "Ulysses 2000: In Search of Optimal Solutions to Hard Combinatorial Problems," Technical Report, New York University Stem School of Business.
56. Hajek, B. (1988) Cooling schedules for optimal annealing. Mathematics of operations Research, 13, 311-329.
57. Heinz Mühlenbein and Hans-Michael Voigt, (1995). Gene Pool Recombination in Genetic algorithms. In Ibrahim H. Osman and James P. Kelly, editors, Proceedings of the Meta-heuristics Conference, pages 53-62, Norwell, USA,. Kluwer Academic Publishers.
58. Helbig, H.K. and Krarup, J. (1974). "Improvements of the Held-Karp Algorithm for the Symmetric Traveling-Salesman Problem". *Mathematical Programming*. 7:87-96.
59. Held, M. and Karp, R.M. (1962). "A Dynamic Programming Approach to Sequencing Problems". Journal of the Society of Industrial and Applied Mathematics. 10:196-210.

60. Held, M. and Karp, R.M. (1970). "The Traveling-Salesman Problem and Minimum Spanning Trees". Operations Research. 18:1138-1162.
61. Held, M. and Karp, R.M. (1970). "The Traveling-Salesman Problem and Minimum Spanning Trees: Part II". Mathematical Programming. 1:6-25.
62. Heller, I. (1955). "On the Travelling Salesman's Problem". Proceedings of the Second Symposium in Linear Programming: Washington, D.C. Vol. 1.
63. Helsgaun, K. (2000) "An effective implementation of the Lin- Kernighan traveling salesman heuristics," European Journal of Optimizational Research, vol. 126, no. 1, pp. 106-130.
64. Hoffman A. J. and Wolfe P. (1985), "History" in *The Traveling Salesman Problem*, Lawler, Lenstra, Rinooy Kan and Shmoys, eds., Wiley, 1-16.
65. Hoffman K.L. and Padberg M., (1991) LP-based Combinatorial Problem Solving, Annals Operations Research, 4, 145-194.
66. Holland, J. (1975). Adaptation *in Natural and Artificial Systems*, Michigan University Press.
67. Holldobler, D.S. and Wilson, E.O., (1990). The Ants (Springer-Verlag, Berlin).
68. Hong. S. (1972). "A Linear Programming Approach for the Traveling Salesman Problem". Ph.D. Thesis. The Johns Hopkins University: Baltimore, MD.
69. Johnson, D.S., Gutin, G. McGeoch, L.A., Yeo, A., Zhang, W., and Zverovitch, A. (2002). "Experimental Analysis of Heuristics for the Asymmetric traveling Salesman Problem". In: Gutin G and Punnen H. (eds). The Traveling Salesman Problem and it Variations. Kluwer Academic Publishers.

70. Karg, R.L. and Thompson, G.L. (1964). "A heuristic Approach to Solving Travelling Salesman Problems". Management Science. 10:225-248.
71. Karp, R.M., Steele, J.M., (1990), "Probabilistic Analysis of Heuristics. In: The Traveling Salesman Problem", Wiley, New York, pp. 181-205.
72. Keuthen, R. (2003). "Heuristic Approaches for Routing Optimization". PhD thesis at the University of Nottingham: UK.
73. Kiatsupaibul, S. and Smith, R.L. (2000) A general purpose simulated annealing algorithm for integer Linear aprogrammimng. Technical Report, Department of Industrial and Operations Engineering, University of Michican, Ann Arbor, Michigan.
74. Kirkpatrick, S., C. D. Gelatt Jr., M. P. Vecchi, (1983) "Optimization by Simulated Annealing" Science, 220, 4598, 671-680.
75. Kirkpatrick, S., Gelatt, C.D. Jr., and Vecchi, M.P. (1983). Optimizations by simulated annealing. Science, 220, 671-681.
76. Kolohan, F. and Liang, M. (2000). "Optimization of Hole Making: A Tabusearch Approach". International Journal of Machine Tools & Manufacture. 50:1735-1753.
77. Krustal, J.B. (1956). "On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem". *Proceedings of the American Mathematical Society*. 2:48-50.
78. Lambert, F. (1960). "The Traveling- Salesman Problem". Cahiers du centre de Recherché Operationelle. 2:180-191.

79. Lawler, E.L. and Wood, D.E. (1966). "Branch-and-Bound Methods: A Survey". *Operations Research*. 14:699-719.

80. Lawler, E.J., Lenstra, J.K., Rinnoy Kan, A.H.G., and Shmoys, D.B. (1985). "The Travelling Salesman Problem: A Guided Tour of Combinatorial Optimization". John Wiley & Sons: New York, NY. 111. Lawler and Wood D.E, (1966) Branch-and-Bound Methods: A Survey, Opns. Res. 14, 69-719.

81. Lawler E.L., Lenstra J.K., Rinnooy Kan A.H.G, and Shmoys D.B., (1986). The Traveling Salesman. John Wiley and Sons.

82. Lawler and Wood D.E, (1966) Branch-and-Bound Methods: A Survey, Opns. Res. 14, 69-719.

83. Lin, S. and Kernighan, B.W. (1973). "An Effective Heuristic Algorithm for the Traveling –Salesman Problem". *Operations Research*. 21:498-516.

84. Little, J.D.C., Murty, K.G., Sweeney, D.W., and Karel, C. (1963). "An Algorithm for the Traveling Salesman Problem". *Operations Research*. 11:972-989.

85. Locatelli, M. (2000) Simulated annealing algorithms for continuous global optimization: convergence conditions. Journal of Optimization Theory and Applications, 104, 121-133.

86. Metropolis, M., Rosenbluth, A., Rosenbluth, M., Teller, A., and Teller, E. (1953). Equation of state calculations by fast computing machines. Journal of Chemical Physica, 21, 1087-1092.

87. Metropolis, N.,A. Rosenbluth, M., Rosenbluth, A. Teller, E. Teller, (1953). "Equation of State Calculations by Fast Computing Machines". *J. Chem. Phys*., 21, 6, 1087-1092.

88. Miller, C.E., Tucker, A.W., and Zemlin, R.A. (1960), "Integer Programmimng formulations and traveling salesman problems", *Journal of the Association for Computing Machinery* 7, 326-329.

89. Miller, D. and Pekny, J. (1991), Exact Solution of Large Asymmetric Traveling Salesman Problems, Science, 251:754-761.

90. Mitrovic-Minic, S. and Krishnamurti, R. (2006). "The Multiple TSP with Time Windows: Vehicle Bounds Based on Precedence Graphs". *Operations Research Letters*. 34(1): 111-120.

91. Morton, G. and Land, A.H. (1955). "A Contribution to the Travelling-Salesman Problem". Journal of the Royal Statistical Society, Series B. 17:185-194.

92. Nahar, S., Sahni, S., and Shragowitz, E. (1989), "Simulated annealing and combinatorial optimization", *International Journal of Computer Aided VLSI Design* 1, 1-23.

93. Notes on Tabu Search Retrieved from http//itc.ktu.it/itc32/Misev32.pdf.

94. Padberg, M., and Rinaldi, G. (1991): A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems, SIAM Review 33, 60-100.

95. Padberg, M. W. and Rinaldi, G. (1987). "Optimization of a 532-city Symmetric Traveling Salesman Problem by Branch and Cut". Operations Research Letters. 6:17.

96. Padberg, M., and Rinaldi, G. (1991): A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems, SIAM Review 33(1), 60-100.

97. Peterson, C. "Parallel distributed approaches to combinatorial optimization: Benchmark studies on traveling salesman problem," Phys. *Rev. Lett*., to appear.

98. Potvin, J.Y. (1996). "The Traveling Salesman Problem: A Neutral Network Perspective". ORSA *Journal on Computing*. 5:328-347.

99. Radin, L.R. (1998). Optimization in Operations Research. Prentice Hall Inc. New Jersey.

100. Raymond, T.C. (1969). "Heuristic Algorithm for the Traveling-Salesman Problem". IBM Journal of Research and Development. 13:400-407.

101. Riera-Ledesma, J. and Salazar-Gonzalez, J.J. (2005). "A Heuristic Approach for The Travelling Purchaser Problem". *European Journal of Operations Research*. 162(1): 142-152.

102. Robacker, J.T. (1955). "Some Experiments on the Traveling-Salesman Problem". RAND Research Memorandum.

103. Roberts, S.M. and Flores, B. (1966). "An Engineering Approach to the Traveling Salesman Problem". *Management Science*. 13:269-288.

104. Robinson, B. (1949). "On the Hamiltonian Game (A Traveling-Salesman Problem)". RAND Research Memorandum.

105. Rosenkrantz, D.J., Stearns, R.E., and Lewis, II, P.M. (1977), "An analysis of several heuristics for the traveling salesman problem", SIAM Journal on Computing 6, 563-581.

106. Rossman, M.J and Twery, R.J. (1958). "A Solution to the Travelling Salesman Problem". Operations Research. 6:687.

107. Shapiro, D. (1966). "Algorithms for the Solution of the Optimal Cost Traveling Salesman Problem". Sc.D. Thesis, Washington University: St. Loius, MO.

108. Siarry, P., Berthiau, G., Durbin, F. and Haussy, J. (1997) Enhanced simulated annealing for globally minimizing functions of many-continuous variables. ACM Transactions On Mathematical Software, 23, 209-228.

109. Smith, T.H.C. and Thompson, G.L. (1977). "A LIFO Implicit Enumeration Search Algorithm for the Symmetric Traveling Salesman Problem using Held and Karp's 1-Tree Relaxation". In: *Studies in Integer programming*. P.L. Hammer, E.L. Johnson, B.H. Korte, and G.L. Nemhauser (eds.). Annals of Discrete Mathematics 1: North-Holland, Amsterdam. 479-493.

110. Tian, P. and Yang, S. (1993). An Improved Simulated Annealing Algorithm with Generic Charisteristis and Travelling Salesman Problem. Journal of Information and Optimization Science. 14(3):241-254.

111. Volgenant, T. and Jonker, R. (1982). A Branch and Bound Algorithm for the Symmetric Traveline Salesman Problem Based on the 1-Tree Relaxation. *European Journal of Operational Research*. 9:83-89.

112. Walshaw, C.A. (2001). Multilevel Lin-Kernighan-Helsgaun Algorithm for the Travelling

113. Salesman Problem. CMS press Centre for Numerical Modelling and process Analysis. University of Greenwich: London, UK.

114. Walshaw, C.A. (2002). Multilevel Approach to the Travelling Salesman Problem, Operations Research. 50(5): 862-877.

115. Wikipedia, the free encyclopedia- Moore's law (2009). Retrieved from http://en.wikipedia.org/wiki/Moore's_law.

116. Xu, X. and Tsai, W.T. "Effective neutral algorithms for the traveling salesman problem", Neutral Networks 4 (1991), 193-205.

117. Zabinsky, Z.B., Smith, R.L., McDonald, J.F., Romeijn, H.E. and Kaufman, D.E. (1993) Improving hit-and-run for global optimization. Journal of Global optimization, 3, 171-192.

## APPENDIX A

## MATLAB PROGRAM

```
function
[tour,best_tour,best_obj,obj_prev,nbr,T,iter,iter_snc_last_chng,accpt_ratio,crnt_tour,crnt
_tour_cost] = init_solution(d1,cost,min_dist,short_path,tour,row,d)
% ********Generate Initial solution - find shortest path from each node****
% if node pair 1-2 is selected, make distance from 2 to each of earlier
%visited nodes very high to avoid a subtour
k = 1;
for i=1:row-1
   min_dist(i) = min(d1(k,:));
   short_path(i) = find((d1(k,:)==min_dist(i)),1);
   cost = cost+min_dist(i);
   k = short_path(i);
   % prohibit all paths from current visited node to all earlier visited nodes
   d1(k,1)=10e+06;
   for visited_node = 1:length(short_path);
      d1(k,short_path(visited_node))=10e+06;
   end
end
tour(1,short_path(1))=1;
for i=2:row-1
   tour(short_path(i-1),short_path(i))=1;
```

```
end
%Last visited node is k;
%shortest path from last visited node is always 1, where the tour
%originally started from
last_indx = length(short_path)+1;
short_path(last_indx)=1;
tour(k,short_path(last_indx))=1;
cost = cost+d(k,1);
% A tour is represented as a sequence of nodes startig from second node (as
% node 1 is always fixed to be 1
crnt_tour = short_path;
best_tour = short_path;
best_obj =cost;
crnt_tour_cost = cost;
obj_prev = crnt_tour_cost;
fprintf('\nInitial solution\n');
fprintf('%5d',crnt_tour)
fprintf('\nInitial tour cost = %d\t', crnt_tour_cost);
nbr = crnt_tour;
T0 = 1.5*crnt_tour_cost;
T=T0;
iter = 0;
iter_snc_last_chng = 0;
```

```
accpt_ratio =1;


function [d1,tour,cost,Lmax,ATmax,alfa,Rf,Iter_max,min_dist,short_path] =
init_parameter(data_siz,d)
% ******************Initialise all parameters**********************
d1=d;
tour = zeros(data_siz);
cost = 0;
min_dist=[];
short_path=[];
%*************************************************************************
%************Initialize Simulated Annealing paratemers************
%T0 Initial temperature is set equal to the initial solution value
Lmax = 400; %Maximum transitions at each temperature
ATmax = 200; %Maximum accepted transitions at each temperature
alfa = 0.99; %Temperature decrementing factor
Rf = 0.0001; %Final acceptance ratio
Iter_max = 1000000; %Maximum iterations 13


function [d,tour_sum,data_siz,row,start_time,d_orig] = distanceTable()
d = xlsread('dataNo.xlsx');
d_orig = d;
start_time = cputime;
```

```
tour_sum=0;
row = size(d,1);
data_siz = size(d);
for i=1:row
   d(i,i)=10e+06;
end
for i=1:row-1
   for j=i+1:row
      d(j,i)=d(i,j);
   end
end

function printResult(best_obj,start_time,best_tour)
fprintf('\nbest obj = %d', best_obj);
fprintf('\n best tour\n');
fprintf('%5d',best_tour);
end_time = cputime;
exec_time = end_time - start_time;
fprintf('\ntime taken = %f\t\n', exec_time);
```