

**KWAME NKRUMAH UNIVERSITY OF SCIENCE AND
TECHNOLOGY, KUMASI-GHANA**

**OPTIMAL CAMPAIGN VISITATION OF
PRESIDENTIAL ASPIRANTS**

Case Study: Brong Ahafo Region of Ghana

Presented By

AGYEMANG KOFI EMMANUEL

(PG2006008)

THESES SUPERVISOR: DR. S.K AMPONSAH

CHAPTER 1

INTRODUCTION

1.1 Background to the study

The Brong Ahafo Region is located in mid-western Ghana, between the Ashanti Region and Cote d'Ivoire border. Its capital is Sunyani. Sunyani is linked to Accra, the nation's capital by a first class road and is about seven hours drive between them at a relatively regular pace. It is the second largest region in Ghana in terms of landmass with a territorial size of 39,557km² with a population of about 1,824,857 according to the 2000-population census.

The region has twenty four (24) constituencies out of the two hundred and thirty constituencies in Ghana. According to the 2008 voter's register, the voter population in the region is 1,175,221. As at September 2008, there were fifteen registered political parties in Ghana. The twenty four (24) 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 Brong Ahafo

CONSTITUENCY	CAPITAL	VOTER POPULATION
Asutifi North	Kenyase	27,204
Asutifi South	Kukuom	28,288
Atebubu-Amantin	Atebubu-Amantin	43,285
Berekum	Berekum	72,451

Dormaa West	Dormaa-Ahenkro	70,103
Dormaa-East	Wamfie	29,020
Jaman North	Sampa	39,686
Jaman South	Drobo	48,825
Kintampo North	Kitampo	47,128
Kintampo South	Jema	36,543
Nkoranza North	Busuaa	28,919
Nkoranza South	Nkoranza	47,050
Pru	Yeji	56,771
Sene	Kwamedanso	43,889
Sunyani East	Suyani	79,829
Sunyani West	Domase	55,282
Tain	Nsokor	54,760
Tano North	Duayaw Nkwanta	40,470
Tano South	Bechem	41,818
Techiman North	Tuobodom	36,529
Techiman South	Techiman	86,113
Wenchi	Wenchi	52,380
Asunafo North	Goaso	64,190
Asunafo South	Hwediem	44,688

Out of the twenty-four(24) constituencies, the NPP has fifteen (15) members of parliament whereas the NDC has nine (9).

1.2 STATEMENT OF THE PROBLEM

Parliamentary democracy is very important to every nation's development. The main players in Ghana are the political parties. The political parties spend huge sums of money, time, energy and resources campaigning.

Political parties in Ghana rely on meager dues from party members and donation from individuals for the running of the day to day activities. At some point in time, it is difficult raising the money needed. Some interest groups in the Ghanaian politics have began asking government to support political parties. This is however difficult for a developing nation like Ghana whose per capital income was about seven hundred dollars as at 2007 and which rely on donors to finance about 40% of her budget.

The political parties must therefore find a way by which they can minimize cost they incur during campaigning or depend on multinational companies or rich individuals to finance their campaign activities. Allowing multinationals or rich individuals to finance campaign is not the best option since this opens the door to corruption as these financiers would have to be compensated. More often than not, their compensation is undue. It is therefore a necessity and as a matter of urgency for political parties to find ways by which they can minimize cost.

Taking the optimal rout by the presidential aspirant in all his/her campaign visitation is one sure way by which cost can be reduced. This is what this thesis seeks to do.

1.3 OBJECTIVES

The objectives of this thesis are

- i.to formulate a mathematical model that takes into consideration the actual distance between the constituency capitals of the twenty-four constituencies in the Brong Ahafo Region.
- ii. to determine the optimal rout for visiting all the twenty-four constituencies in the region.

1.4 METHODOLOGY

The campaign visitation by a presidential aspirant to the constituency capitals in the Brong Ahafo region will be modeled as a Travelling Salesman Problem(TSP). The Simulated Annealing Algorithm is the method that will be used for solving the TSP model. This is because the simulated annealing which is a metaheuristic- based search algorithm is capable of solving combinatorial optimization problems like the TSP.

The sources of data for the thesis are the internet and libraries for relevant literature, electoral commission and feeder roads will also be consulted for information on the voter population in the region and the distance of the network routes from one constituency capital to the other respectively.

1.5 IMPORTANCE OF A VISIT BY A PRESIDENTIAL ASPIRANT

1. An aspirant's visit to a constituency enables him/her to be introduced to the electorate in the constituency. Some people get to see him/her for the first time.
2. It affords the candidate an opportunity to deliver his or her message/policies to the people in the constituency.

3. Party members are energized to campaign even in the absence of the aspirant.
4. Electorates in the constituencies also feel the aspirant cares about the.
5. It gives the aspirant the opportunity to know the specific challenges in the various constituencies and assures the electorate as to how such challenges would be dealt with.
6. The visit also enables the aspirant to canvass for votes.

1.6 Organization of the thesis

Chapter one covers the introduction to the thesis, Chapter two the literature review, Chapter three methodology, Chapter four the collection of data, analysis of data and discussion. In Chapter five we shall put forward conclusions and recommendations of the thesis.

1.7 SUMMARY

In this chapter, we presented brief history of Ghana, background to the study, statement of the problem, objectives of the thesis, methodology, importance of a visit by a presidential aspirant and the organization of the thesis.

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

CHAPTER 2

LITERATURE REVIEW

The Travelling Salesman Problem (TSP) is a problem in combinatorial optimization studied in operations research and theoretical computer science. Given a list of cities and their pairwise distances, the task is to find the shortest possible tour that visits each city exactly once.

The Travelling Salesman Problem (TSP) has been studied during the last fifty years and many exact and heuristic algorithms have been developed. These algorithms include construction algorithms, iterative improvement algorithms, branch-and-bound and branch-and-cut exact algorithms and many metaheuristic algorithms, such as simulated annealing (SA), tabu search (TS), ant colony (AC) and genetic algorithm (GA).

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

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 $Q(n^2)$ operations at each step , but producing tour nearly as good as the average performance of 3-opt algorithm.

One of the earliest exact algorithms is due to Dantzig et al(1954), in which linear programming (LP) relaxation is used to solve the integer formulation by suitably chosen 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 Eastman (1958), Held and Karp(1970), Smith et al, Carpaneto and Toth, Balas and Christofides. Some Branch and Cut (B & C) based exact algorithms were developed by Crowder and Padberg, Padberg and Hong , Grotschel and Holland.

Besides the above mentioned exact and heuristic algorithms, metaheuristic algorithms have been applied successfully to the TSP by a number of researchers. SA algorithms for the TSP were developed by Bonomi and Lutton, Golden and Skiscim and Nahr et al. Lo and Hus etc. Tabu search metaheuristic algorithms for TSP have been proposed by Knox and Fiechter. The AC is a relative new metaheuristic algorithm which is applied successfully to solve the TSP. some work based on SA technology was reported by Dorigo et al. Genetic algorithms for the TSP were reported by Goefenstetle et al.

Applegate et. al (1994) solved a traveling 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 problem of a 16 city traveling salesman problem, one finds that there are 653,837,184,000 distinct routes (Grötschel 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 traveling 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 "traveling 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 neighbor 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-neighbor algorithm and 2-opt while Kruskal, (1956) drew attention to the similarity between the TSP and the minimum-length spanning trees 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.

In Miller et al. (1960), 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 percent longer than the optimal solutions. TSP in 1960 achieved national prominence in the United States of America when Procter & 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).

Müller-Merbach (1961) proposed an algorithm for the asymmetric TSP; he illustrated it on a 7-city example. Ackoff et al. (1961) gave a good survey of the computational work on the TSP that was carried out in the 1950's.

By 1962, when the computer was becoming a useful tool in exploring 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 TSPs having random edge lengths. In an important paper (Lin, 1965): a

heuristic method for the TSP was published. The author defined k -optimal 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 trees 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 in their 1970 paper 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 M. 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 algorithm (1972) 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 subtour 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 Euclidean 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).

A very impressive work heralded the 1980s. Crowder and Padberg (1980) gave the solution of a 318-city instance described in Lin and Kernighan (1973). The 318-city instance would remain 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, M. (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,” says Radin L.R in 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.

Grötschel (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), Crama et al, (2002), Ferreir (1995), Foulds and Hamacher (1993), Günther et al (1998), 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 percent of all the possible tours. Thus, only about three percent 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 Grötschel (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 todays 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 computing 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

The travelling salesman problem is combinatorial optimization problem. According to Hillier and Lieberman (2005) it has been given this picturesque name because it can be described in terms of a salesman (or saleswoman) who must travel to a number of cities during one tour. Starting from his or her home city, the salesman follows to visit each city exactly once before returning to his home city as to minimize the total length of the tour.

The figure below shows an example of a small travelling salesman problem with seven cities.

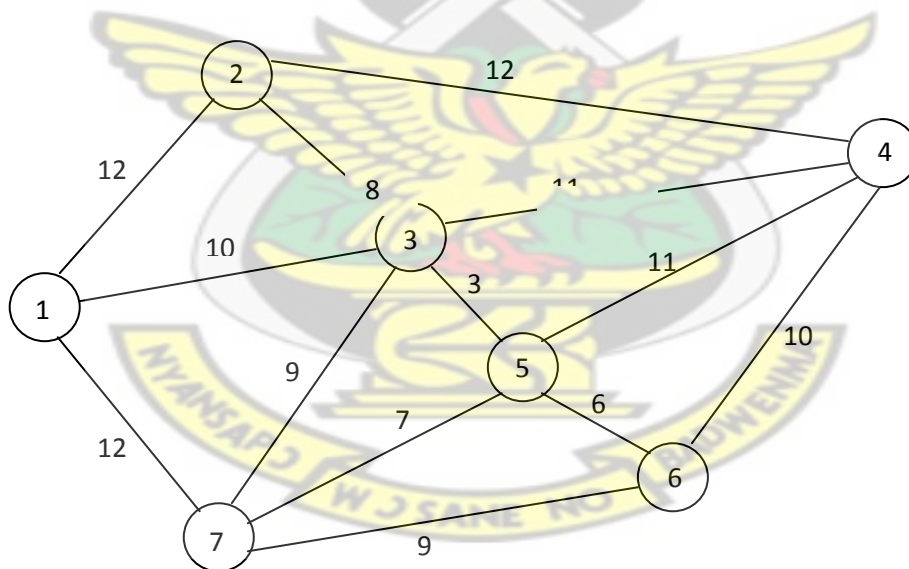


Figure 3.1 Traveling Salesman Problem

City 1 being the salesman's home city, he starts from this city, and must choose a route to visit each of the other cities exactly once before returning to city 1. The number next to each link between each pair of cities represents the distance between these cities. The objective is to determine which route will minimize the total distance

that the salesman must travel. The Sub-tour Reversal Algorithm, the Tabu Search and the simulated annealing will each be used to find the optimal solution for the travelling salesman problem above later in this chapter.

There have been a number of applications of travelling salesman problems that have nothing to do with salesmen. For example, when a presidential aspirant leaves his home city and visits a number of cities campaigning and returns to his home city after a period, the problem of determining the shortest route for doing this tour is a travelling salesman problem. Another example involves the manufacture of printed circuit boards for wiring chips and other components. When many holes need to be drilled into a printed circuit board, the process of finding the most efficient drilling sequence is a travelling salesman problem.

The difficulty of travelling salesman problems increases rapidly as the number of cities increases. For a problem with n cities, the number of feasible routes to be considered is $(n-1)!/2$ since there are $(n-1)$ possibilities for the first city after the home city, $(n-2)$ possibilities for the next city, and so forth. The denominator of 2 arises because every route has an equivalent reverse route with exactly the same distance. Thus, while a 10-city travelling salesman problem require less than 200,000 feasible solutions to be considered, a 20-city problem has roughly 10^{16} feasible solutions while a 50-city problem has about 10^{62} feasible solutions.

Formulation of TSP model

The problem can be defined as follows: Let $G = (V, E)$ be a complete undirected graph with vertices V , $|V|=n$, where n is the number of cities, and edges E with edge length d_{ij} for (i, j) . The focus is on the symmetric TSP in which case $d_{ij} = d_{ji}$, for all (i, j) . This minimization problem can be formulated as an integer programming as shown below in Equations (1) to (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 travelled.

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 to one 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.

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

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

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

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

$$x_{ij} \text{ or } 1 \quad i, j \in v \quad (5)$$

However, the difficulty of solving TSP is that sub-tour constraints will grow exponentially as the number of cities grows large, so it is not possible to generate or store these constraints. Many applications in real world do not demand optimal solutions.

3.2 The Sub-Tour Reversal Algorithm

This adjusts the sequence of cities visited in the current trial solution by selecting a sub-sequence of the cities and simply reversing the order in which that sequence of cities is visited.

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

Iteration: For the current trial solution, consider all possible ways of performing a sub-tour reversal except reversal of the entire tour. Select the one that provides the largest decrease in the distance travelled to the new trial solution

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

Applying this algorithm to the problem above and starting with 1-2-3-4-5-6-7-1 as the initial trial solution, there are four possible sub-tour reversals that would improve upon this solution as shown below

$$1-2-3-4-5-6-7-1 = 69$$

$$\text{Reverse 2-3: } 1-3-2-4-5-6-7-1 = 68$$

$$\text{Reverse 3-4: } 1-2-4-3-5-6-7-1 = 65$$

$$\text{Reverse 4-5: } 1-2-3-5-4-6-7-1 = 65$$

$$\text{Reverse 5-6: } 1-2-3-4-6-5-7-1 = 66$$

The solution with distance = 65 tie for providing the largest decrease in the distance travelled, so suppose that the first of these 1-2-4-3-5-6-7-1 is chosen to be the next trial solution. This completes the first iteration.

The second iteration has only one sub-tour reversal that will provide an improvement as shown below

$$1-2-4-3-5-6-7-1 = 65$$

$$\text{Reverse 3-5-6: } 1-2-4-6-5-3-7-1 = 64$$

At this point there is no sub-tour reversal that will improve upon this new trial solution. So the sub-tour reversal algorithm stops with this trial solution as the final solution even though by other methods, 1-2-4-6-5-3-7-1 is not the optimal solution.

3.3 Tabu Search

According to Hillier and Lieberman Tabu Search is a widely used metaheuristic that uses some common sense ideas to enable the search process to escape from a local optimum. The concept of tabu search (TS) is derived from artificial intelligence where intelligent use of memory

helps in exploiting useful historical information. The restrictions put on the information in the memory reminiscent of the definition of the word 'tabu' as "a set apart as charged with a dangerous supernatural power and forbidden to profane use or contact". Tabu search can also incorporate some more advanced concepts. One is intensification, which involves exploring a portion of the feasible region more thoroughly than usual after it has been identified as a particularly promising portion for containing very good solutions. Another concept is diversification, which involves forcing the search into previously unexplored areas of the feasible region. The focus will however be on the basic form of tabu search summarized below.

1. **Initialization** : A starting solution generated by choosing a random solution, $x \in S$. The evaluating function $f(x)$ is used to evaluate x . The solution is stored in the algorithm memory called the tabu list.
2. **Neighborhood exploration**: All possible neighbours $\mu(x)$ of the solution x are generated and evaluated. Solutions in the tabu list are considered unreachable neighbours, they are taboo (tabu). An immediate neighbor can be reached by making a sub-tour reversal.
3. **New Solution**: A new solution is chosen from the explored neighbourhood. This solution should not be found in the tabu list before it is discovered and has to have the best move evaluation value of $f(x)$ for all reachable neighbours of x .

Do tabu check on the new solution. If it is successful replace the current solution and update the tabu list and other tabu attributes. Here the new

solution evaluation value can be worse compared with that of current solution. This enables the solution not to be trapped at local optimum. The tabu check applied based on the move being the best move.

- (i) If the solution is in the tabu list then check the aspiration level. If successful replace the current solution and update the tabu list and other tabu attributes. The aspiration check uses the function evaluation and the success of the check depends on the function evaluation of the new solution being better than that of the current best solution.
- (ii) If checks (i) and (ii) are not successful then keep the current solution otherwise replace the current solution by the new solution.
- (iii) Compare the best solution to the current solution, if the current solution is better than the best solution, replace the best solution.
- (iv) Until loop condition is satisfied go to step Until termination condition is satisfied go to step 1.
- (v) Stop after three consecutive iterations without an improvement in the best objective function value. Also stop at any iteration where the current trial solution has no immediate neighbours that are not ruled out by their tabu status.

To apply this tabu search algorithm to the problem above,

Let initial trial solution = 1-2-3-4-5-6-7-1 Distance = 69

Tabu list : Blank at this point

Iteration 1:

reverse 3-4

Delete Links: 2-3 and 4-5

Added links: 2-4 and 3-5

Tabu list : Links 2-4 and 3-5

New trial solution: 1-2-4-3-5-6-7-1 Distance = 65

Iteration 2

Reverse 3-5-6

Delete links: 4-3 and 6-7

Added links: 4-5 and 3-7

Tabu list: links 2-4, 3-5, 4-6 and 3-7

New trial solution: 1-2-4-6-5-3-7-1 Distance = 64

The tabu search algorithm now escapes from this local optimum by moving next to the best immediate neighbor of the current trial solution even though its distance is longer. Considering the limited availability of links between pairs of cities in fig....., the current trial solution has only the two immediate neighbours listed below.

Reverse 6-5-3: 1-2-4-3-5-6-7-1 Distance = 65

Reverse 3-7: 1-2-4-6-5-7-3-1 Distance = 66

Reversing 2-4-6-5-3-7 to obtain 1-7-3-5-6-4-2-1 is ruled out since it is simply the same tour in the opposite direction. However the of these immediate neighbours must

be ruled out because it would require deleting links 4-6 and 3-7, which is tabu since both of these links are on the tabu list. This move could still be allowed if it would improve upon the best trial solution found so far but it does not.

Ruling out this immediate neighbor does not allow cycling back to the preceding trial solution. Therefore by default, the second of these immediate neighbours is chosen to be the next trial solution as summarized below.

Iteration 3

Reverse 3-7

Delete links: 5-3 and 7-1

Add links: 5-7 and 3-1

Tabu List: 4-5, 3-7, 5-7 and 3-1

New trial solution: 1-2-4-6-5-7-3-1 Distance = 66

The sub- tour reversal for this iteration can be seen in the fig....., where the dashed lines show the links being deleted (on the left) and added (on the right) to obtain the new trial solution.

The new trial solution has the four immediate neighbours listed below.

Reverse 2-4-6-5-6: 1-7-5-6-4-2-3-1 Distance = 65

Reverse 6-5: 1-2-4-5-6-7-3-1 Distance = 69

Reverse 5-7: 1-2-4-6-5-7-3-1 Distance = 63

Reverse 7-3: 1-2-4-6-5-3-7-1 Distance

Both of the deleted links 4-6 and 5-7 are on the tabu list. The second of these immediate neighbours is therefore tabu. The fourth immediate neighbor is also tabu. Thus, there are only two options, the first and the third immediate neighbours. The third immediate neighbor is chosen since it has shorter distance.

Iteration 4

Reverse 5-7

Delete links: 6-5 and 7-3

Add links: 6-7 and 5-3

Tabu list: 5-7, 3-1, 6-7 and 5-3

(4-6 and 3-7 are now deleted from the list)

New trial solution: 1-2-4-6-7-5-3-1 Distance = 63

The only immediate neighbor of the current trial solution would require deleting links 6-7 and 5-3, both of which are on the tabu list so cycling back to the preceding trial solution is prevented. Since no other immediate neighbours are available, the stopping rule terminates the algorithm at this point with 1-2-4-6-7-5-3-1 as the final solution with Distance = 63.

3.3 Simulated Annealing

According to Amponsah and Darkwah(2007) the concept of Simulated Annealing is derived from Statistical mechanics in the area of natural sciences. A piece of regular metal in its natural state has the magnetic direction of its molecules aligned in uniform

direction. As the metal is heated, the kinetic energy of the molecules increases and the cohesive force decreases till when the molecules are free to move about randomly. The magnetic directions of the molecules are oriented randomly.

To achieve regularity of alignment of the magnetic direction so as to make the metal stable for use, it must be cooled slowly. This slow cooling of the metallic material is called annealing. In 1953 Metropolis and others recognised the use of Boltzman's law to stimulate the efficient equilibrium condition of a collection of molecules at a given temperature and thus facilitate annealing. When the metal is heated to higher temperature 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. An energy function can be obtained by combining the parameters.

In 1983 Kirk Patrick showed how Simulated Annealing of Metropolis could be adapted to solve problems in Combinatorial Optimization.

The following analogy was made

1. a) Annealing looks for system state at a given temperature.
b) Optimization looks for feasible solution of the combinatorial problems
2. 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.
3. 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 function value.
- 4. 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 the optimal solution
- 5. 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 metaheuristic for the global optimization problem of applied mathematics, namely locating a good approximation to the global minimum of a given function in a large search space. It is often used when the search space is discrete (e.g., all tours that visit a given set of cities). For certain problems, simulated annealing may be more effective than exhaustive enumeration — provided that the goal is merely to find an acceptably good solution in a fixed amount of time, rather than the best possible solution.

3.3.1 Using simulated Annealing to solve TSP

The TSP was one of the first problems to which simulated annealing was applied, serving as an example for both Kirkpatrick et al. (1983) and Cerny (1985). Since then the TSP has continued to be a prime test bed for the approach and its variants. Most adaptations have been based on the simple schema presented in Figure below, with

implementations differing as to their methods for generating starting solutions (tours) and for handling temperatures, as well as in their definitions of *equilibrium*, *frozen*, *neighbor*, and *random*. Note that the test in Step *g* is designed so that large uphill moves are unlikely to be taken except at high temperatures *t*. The probability that an uphill move of 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 neighbor* S^* of the current solution.
 - e. Set $\Delta = \text{Length}(S^*) - \text{Length}(S)$.
 - f. If $\Delta \leq 0$ (downhill move):

Set $S = S^*$.

If $\text{Length}(S) < \text{Length}(S^*)$, set $S^* = S$.
 - h. If $\text{length}(S) < \text{length}(S^*)$ (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 Example

Considering Figure 3.1

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

using the parameters;

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

Stop when $T < 0.1$

First Iteration

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

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

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

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

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

Since $\delta < 0$, set $x^0 \leftarrow x^1$

Updating the temperature $T_1 = \alpha T_0 = 0.5(20) = 10$

Second Iteration

$$d(x^0) = 68$$

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

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

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

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

Since $\delta < 0$, set $x^0 \leftarrow x^1$

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

Third Iteration

$$d(x^0) = 65$$

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

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

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

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

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

A random number would be generated from a computer say θ

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

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

This process will continue until the final temperature and the optimal solution are obtained.

3.4 Genetic Algorithm

The genetic algorithm (GA) is an evolutionary algorithm inspired by Darwin (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 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.

The GA have the following simulations of the evolutionary principles;

Evolution	Genetic Algorithm
An <i>individual</i> is a genotype of the species	An <i>individual</i> is a solution of the optimization problem.
<i>Chromosomes</i> defined the structure of an individual.	<i>Chromosomes</i> are used to represent the data structure of the solution.
Chromosomes consists of sequence of cells called <i>genes</i> which contain the	Chromosomes consists of sequence of gene species which are <i>placeholder</i>

structural information.	boxes containing string of data whose unique combination give the solution value.
The genetic information or trait in each gene is called an allele	An allele is an element of data structure stored in a gene placeholder.
Fitness of an individual is an interpretation of how the chromosomes have adopted to competition environment.	Fitness of a solution consists in evaluation of measures of the objective function for the solution and comparing it to the evaluations for other solutions
A population is a collection of species found in a given location.	A population is a set of solution that form domain search space.
A generation is a given number of individuals of the population identified over a period of time.	A population 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 parent for reproduction	Selection is the operation of selecting parents from the generation to produce offsprings
Crossover is mating and breeding of offsprings by chromosomes 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	Mutation is a random operation whereby the allele of a gene in a chromosome of the offspring is changed by a probability

parents are distorted.	pm.
Recombination is a process of nature's survival of the fittest	Recombination 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.

. Table:3.4.0 The relationship between Evolution and Genetic Algorithm

Given a population at t , genetic operators are applied to produce a new population at time $t+1$. A stepwise evolution of the population from the time t to $t+1$ is called generation. The GA for a single generation is based on the general framework of selection, crossover, Mutation and Recombination.

3.4.1 Representation of individuals

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

Suppose we have the search space $x=0,1,2,\dots,10$ then the x values form the individual. The elements of the search space in a binary sequence are encoded by expressing $x=10$ and $x=0$ in 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 placeholder 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.

3.4.2 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 i th city

3.4.3 Initial population

A GA begins with a population of potential solutions. . For function optimization, the variable x in the objective function $f(x)$ will be encoded in a chromosome consisting of a 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 T by the permutation $T=[1,2,3,4,5]$. Each potential solution must be a feasible as well as being a unique solution.

3.4.5 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 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.

3.4.6 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 from the high fitness range . It is used for selecting individuals for crossover and mutation.

3.4.6.1(Elitist) Selection

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

3.4.6.2 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

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

Where f_k is the fitness of k th chromosome

We then find the cumulative fitness (c_j) of the j th 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 i th chromosome is selected.

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

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

$$\text{And } w_i = \frac{F_i}{\sum_{k=1}^n F_k}$$

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.

3.4.6.3 Tournament Selection

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

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

3.4.6.4 Random Selection

Chromosomes may be selected irrespective of their fitness.

3.4.7 Crossover

After the required selection process the crossover is used to divide a pair of selected chromosome into or more parts. Parts of one of the pair 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 i th position and the one in $(i+1)$ th position. For two chromosomes the crossover point are the same and the crossover operation may produce new chromosomes, which are less fit. In this sense that the crossover operation result in a non-improving solution.

3.4.7 .1 Single point crossover

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

3.4.7. 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

3.4..3 Uniform crossover

Single allele in the same positions are considered for swapping. The probability of selecting an allele for swapping is called Mixing Rate. Mixing rates are set for the allele position. Random numbers are then generated and a position satisfying the mixing rate has the allele in the two chromosomes swapped. Crossover operation is an

exploratory operation that allows the GA to take ‘large jumps’ during the search. As convergence is approached the exploratory power of the crossover diminishes.

3.4.8 Mutation

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

3.4.8.1 Random swap mutation

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

3.4.8.2 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 at another randomly chosen locus.

3.4.8.3 Move-and-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.

3.4.8. 4 Uniform mutation

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

3.4.9 Termination Conditions\

The algorithm terminates when a set of conditions are satisfied. At that point the best solution 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) A standard deviation of the generation of the population approaches a given value.
- iv) The average fitness of the generation of the 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 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 individual and call them parents. Remove the parents from the population.
- iv) Use a random process to determine whether to perform crossover. If so, refer 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 individual .Delete one child at random if n is odd. Then replace the old population with the new generations. Return to i

The simple Genetic algorithm can be summarized in the following steps

Step 1: Code the individual of the search space.

Step 2: Initialize the generation counter ($g = 1$).

Step 3 : Choose initial generation of the population(solution).

Step 4: Evaluate the fitness of each individual in the population.

Step 5: Select individuals of the 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 $g=g+1$

3.5 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 Mühlenbein 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.

3.5.1 Codification

The OGA has a population P of p individuals or solutions, as the SGA does. Every individual P_x 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 c_1, c_2, c_3, c_4 and c_5 , the tour defined by the arcs $(c_1, c_4), (c_4, c_3), (c_3, c_2), (c_2, c_5)$ and (c_5, c_1) will be codified with a string containing the visited cities in order, i.e. $[c_1; c_4; c_3; c_2; c_5]$.

3.5.2 Reproduction

The OGA selects randomly two parents (F_1 and F_2) 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 P_x , 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.

3.5.3 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 operators crossover and mutation are done separately. To facilitate the obtaining of offspring who 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 a roulette, where every arc has a weight w or a probability to be chosen. Crossover-Mutation gives a weight w of 1 to each arc $(i; j)$ belonging to set A , i.e. $w_{ij} = 1 \quad \forall (i; j) \in A$. Then, a weight of $O/2$ is added to each arc $(i; j)$ of $F1$, i.e. $w_{ij} = w_{ji} + O/2 \quad \forall (i; j) \in F1$, where Omicron (O) is an input parameter of the OGA. Analogously, a weight of $O/2 \quad \forall j \in N_i$ is added to each arc $(i; j)$ of $F2$. 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 $S1$, 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 $S1$ is

defined by equation (1)**.

$$p_{ij} = \begin{cases} \frac{w_{ij}}{\sum_{\forall h \in N_i} w_{ih}} & \text{if } j \in N_i \\ 0 & \text{otherwise} \end{cases} \quad (1)**$$

3.5.4 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.

3.5.4.1 Reproduction

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

First randomly chosen individual: $P_1 = \{c_1; c_4; c_3; c_2; c_5\}$ with $f_1 = 10$

Second randomly chosen individual: $P_2 = \{c_1; c_3; c_2; c_5; c_4\}$ with $f_2 = 8$

Third randomly chosen individual: $P_3 = \{c_3; c_5; c_1; c_2; c_4\}$ with $f_3 = 1$

Fourth randomly chosen individual: $P_4 = \{c_2; c_5; c_4; c_1; c_3\}$ with $f_4 = 5$

Two parents are randomly selected through roulette, where the weights of the individuals in the roulette are their fitness. It is assumed that individuals P_1 and P_4 are selected to be parents.

$F_1 = \{c_1; c_4; c_3; c_2; c_5\} = \{(c_1; c_4); (c_4; c_3); (c_3; c_2); (c_2; c_5); (c_5; c_1)\}$

$F_2 = \{c_2; c_5; c_4; c_1; c_3\} = \{(c_2; c_5); (c_5; c_4); (c_4; c_1); (c_1; c_3); (c_3; c_2)\}$

3.5.4.2 Crossover-Mutation.

Iteration 1

First, an initial city is randomly chosen to perform Crossover-Mutation. c_4 is assumed as the initial city. Then, N_{c_4} is composed by $[c_1; c_2; c_3; c_5]$, i.e. the set of not yet visited cities. The arc $(c_4; c_2)$ has a weight of 1 in the roulette because it does not belong to any parent. Arcs $\{(c_4; c_3); (c_4; c_5)\}$ have a weight of $1 + O/2 = 3$ in the roulette because they belong to one parent. Finally, the arc $(c_4; c_1)$ has a weight of $1 + O = 5$ in the roulette because it belongs to both parents. It is assumed that the arc $(c_4; c_3)$ is randomly chosen through the roulette.

3.5.4.3 Crossover-Mutation.

Iteration 2

From c_3 we do crossover mutation operation

N_{c_3} is composed by $\{c_1; c_2; c_5\}$. The arc $(c_3; c_5)$ has a weight of 1 in the roulette because it does not belong to any parent. The arc $(c_3; c_1)$ has a weight of $1 + O/2 = 3$ 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.

3.5.4.4 Crossover-Mutation.

Iteration 3

From $c2$ we do crossover mutation operation

$Nc2$ is composed by $[c1; c5]$. The arc $(c2; c1)$ has a weight of 1 in the roulette because it does not belong to any parent. Finally, the arc $(c2; c5)$ has a weight of $1 + O = 5$ in the roulette because it belongs to both parents. It is assumed that the arc $(c2; c1)$ is randomly chosen through the roulette.

3.5.4.5 Crossover-Mutation.

Iteration 4

$Nc1$ is composed by $[c5]$. The arc $(c1; c5)$ has a weight of $1 + O/2 = 3$ in the roulette because it belongs to one parent. The arc $(c1; c5)$ is chosen because it is the unique arc represented in the roulette. The new offspring is $S1 = [c4; c3; c2; c1; c5] = \{(c4; c3); (c3; c2); (c2; c1); (c1; c5); (c5; c4)\}$. Notice that $S1$ has 3 arcs of $F1 \{(c4; c3); (c3; c2); (c1; c5)\}$ and 2 arcs of $F2 \{(c3; c2); (c1; c5)\}$. Also, $S1$ 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.

3.5.4.6 Population Update

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

$P1 = \{c4; c3; c2; c1; c5\}$ with $f(1) = 7$

$P2 = \{c1; c3; c2; c5; c4\}$ with $f(2) = 8$

$P3 = \{c3; c5; c1; c2; c4\}$ with $f(3) = 1$

$P4 = \{c2; c5; c4; c1; c3\}$ with $f(4) = 5$

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

KNUST

3.10 Some Applications of TSP

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

3.10.1 Vehicle Routing Problem (VRP)

With regard to a particular number of vehicles, Vehicle Routing is the problem of determining which customers should be served by which vehicles, and in what order each

vehicle should visit its customers. 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).

The figure below shows an example of Vehicle Routing Problem (VRP) with four routes where the square in the middle denotes the source node

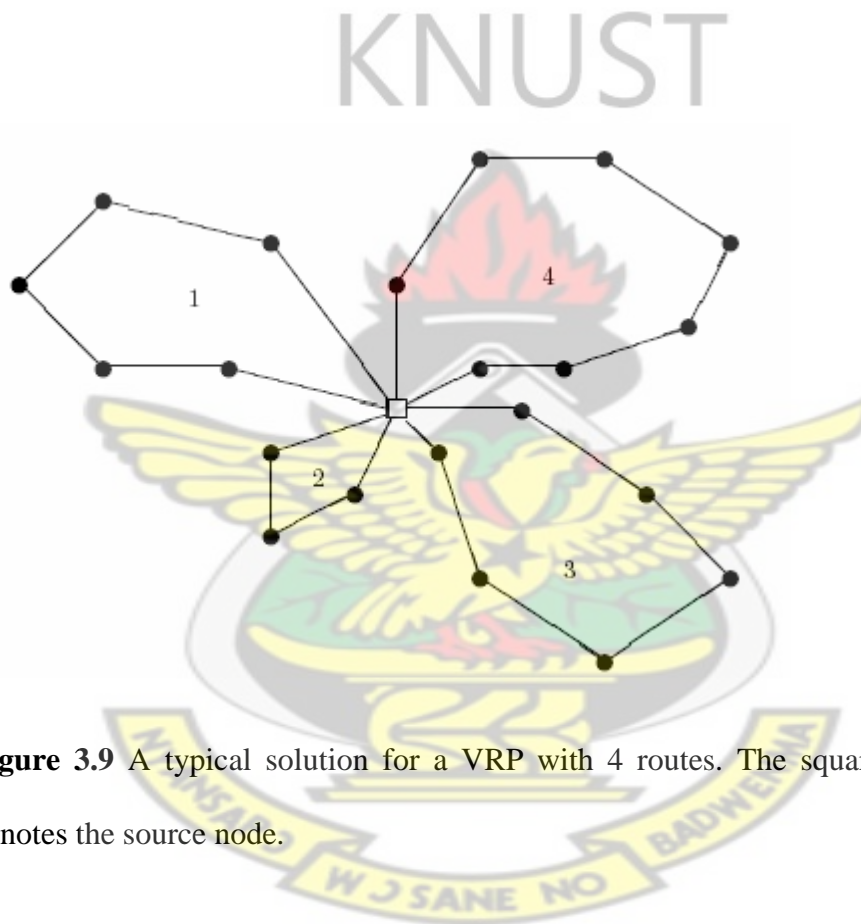


Figure 3.9 A typical solution for a VRP with 4 routes. The square in the middle denotes the source node.

3.10.2 Computer Wiring

This type of problem is common in the design of computers and digital systems. The systems comprise of 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} symbolizes 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).

3.10.3 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.

3.10.4 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 TSP. We try to minimize the total time to process each job. A robot must perform many different operations to complete a process. In this 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).

3.11 Branch and Bound Algorithm

The Branch and Bound(BB) method which was first proposed by A. H Land and A.G Doig in 1960 is a general algorithm for finding optimal solutions of optimization problems such as combinatorial optimization. It consists of systemic enumeration of all candidate solutions, where large sub-sets of fruitless candidates are discarded, using upper and lower estimated bounds of the quantity being optimized.

The steps below are used in the branch and bound algorithm

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

$$\begin{aligned} P_0 : \text{ Maximize } Z &= \sum_{j=1}^n C_j X_j \\ \text{Subject to } \sum_{j=1}^n a_{ij} x_j &\leq b_i, \quad 1 \leq i \leq m \\ X_j &\geq 0 \quad X_j \text{ is integer} \end{aligned}$$

We solve the relaxation problem P_0 by the simplex method.

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

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

has a non-integer value $X_j = K_j$ when it should be an integer. We define $[K_j]$ to be the lower integer part of K_j so that $[K_j] < X_j < [K_j] + 1$. Since X_j must be an integer, it follows that it must obey exactly one of the following constraints.

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

STEP 4: To branch on X_j means solving the following problem. Form two subproblems P_1 and P_2 to replace the current problem P_0 adding a lower bound constraint to one and an upper-bound constraint to the other for the variable selected above in step 3. It then partitions the current subset of solutions into two new subsets of solutions.

We now solve the LP of P_1 such that (iv) $P_1 : P_0 + \{x_j \leq [k_j]\}$ and the problem

$P_2 : P_0 + \{x_j \geq [k_j] + 1\}$ The branching is illustrated in the tree of figure 3.11 below

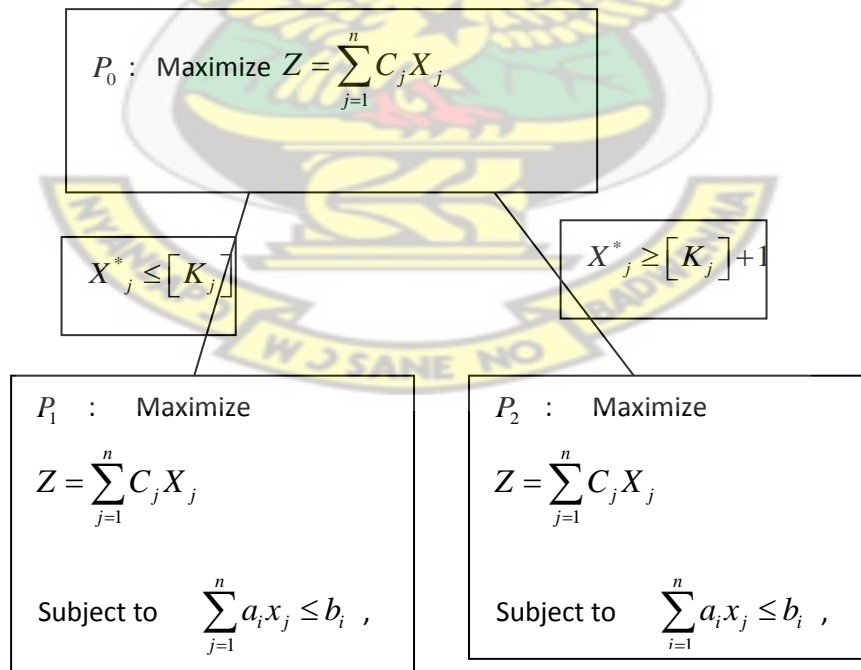


Figure 3.11 solution tree for the method of Daskin.

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

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

3.11.1 Example

Given that

$$\text{Minimize } Z = X_1 + 4X_2 \quad (i)$$

Subject to ;

$$2X_1 + X_2 \leq 8 \quad (ii)$$

$$X_1 + X_2 \geq 6 \quad (iii)$$

$$X_1 \geq 0, X_2 \geq 0 \quad (iv)$$

$$X_1, X_2 \text{ are integers} \quad (v)$$

STEP 1 : The algorithm begins by solving (i) to (iv) as an LP problem. This has the following optimal solutions for P_0 , $X_1^* = \frac{10}{3}$, $X_2^* = \frac{4}{3}$ and $Z^* = \frac{26}{3} = 8\frac{2}{3}$ is the lower bound on the set of all feasible solutions. If this first solution had satisfied (v), it would have been optimal for the integer programming problem and the algorithm would have been terminated. However, as this is not the case, we shall proceed.

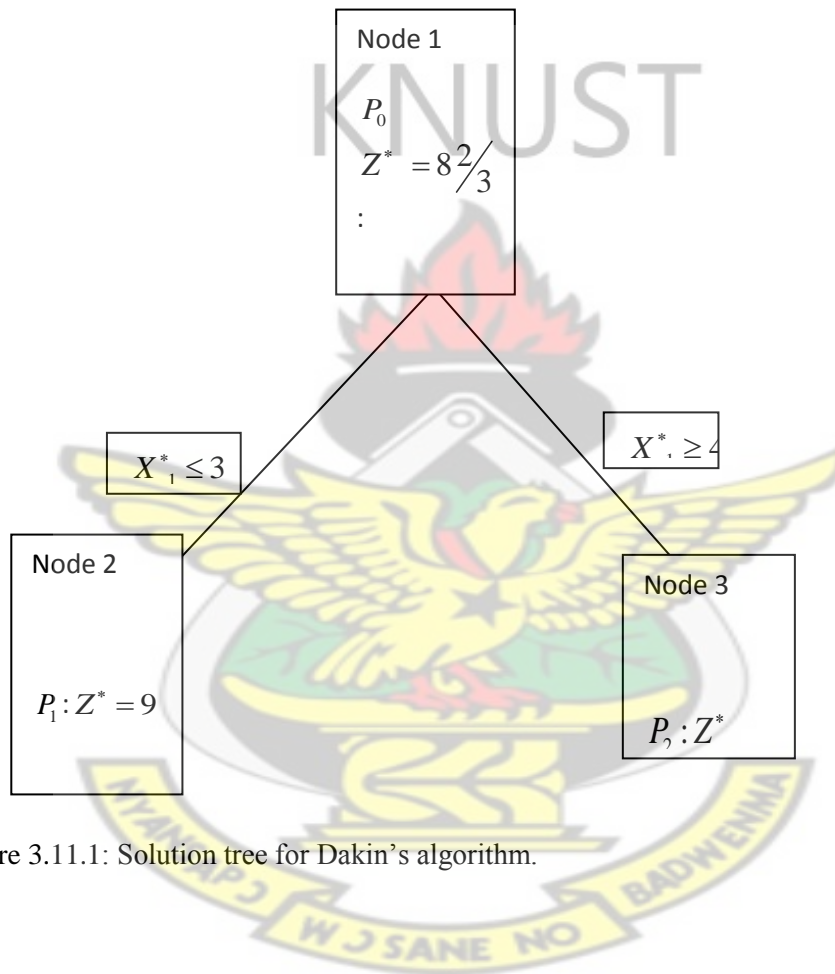


Figure 3.11.1: Solution tree for Dakin's algorithm.

STEP (2): Since X_1^* and X_2^* both have non-integer values in step 1. Arbitrarily select one to branch on. The set of feasible solutions is partitioned into two subsets. One set contains all the feasible solutions with the addition of constraint $X_1^* \leq [\frac{10}{3}] = 3$ and the other contains the set of feasible solutions with the addition of constraint $X_1^* \geq [\frac{10}{3}] + 1 = 4$. This reduces the region of feasible solutions of the LP problem, but leaves the region of feasible solutions of

the integer Linear programming problem unchanged, since there are no integer solution between $3 = \lceil \frac{10}{3} \rceil$ and $4 = \lceil \frac{10}{3} \rceil + 1$. Therefore, iteration 1 begins by partitioning the entire set of solutions into the two subsets below.

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

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

Two LP problems are now created as P_1 and P_2

$$P_1 : \text{Minimize } Z = X_1 + 4X_2$$

$$\text{Subject to } 2X_1 + X_2 \leq 8$$

$$X_1 + 2X_2 \geq 6$$

$$X_1 \leq 3$$

$$X_1 \geq 0, X_2 \geq 0 \quad x_1, x_2 \text{ integers}$$

$$P_2 : \text{Minimize } Z = X_1 + 4X_2$$

$$\text{Subject to } 2X_1 + X_2 \leq 8$$

$$X_1 + 2X_2 \geq 6$$

$$X_1 \geq 4$$

$$X_1 \geq 0, X_2 \geq 0 \quad x_1, x_2 \text{ integers}$$

For problem P_1 , the corresponding LP problem is solved. The solution is

$X_1^* = 3$, $X_2^* = \frac{3}{2}$ and $Z^* = 9$. The solution is still non-feasible for the original problem, but $Z^* = 9$ is the lower bound on the set of all feasible solutions with $X_1^* = 3$, as shown in figure(2). Also, the problem corresponding to P_2 is solved by using the corresponding LP problem. There is no feasible solution for problem P_2 .

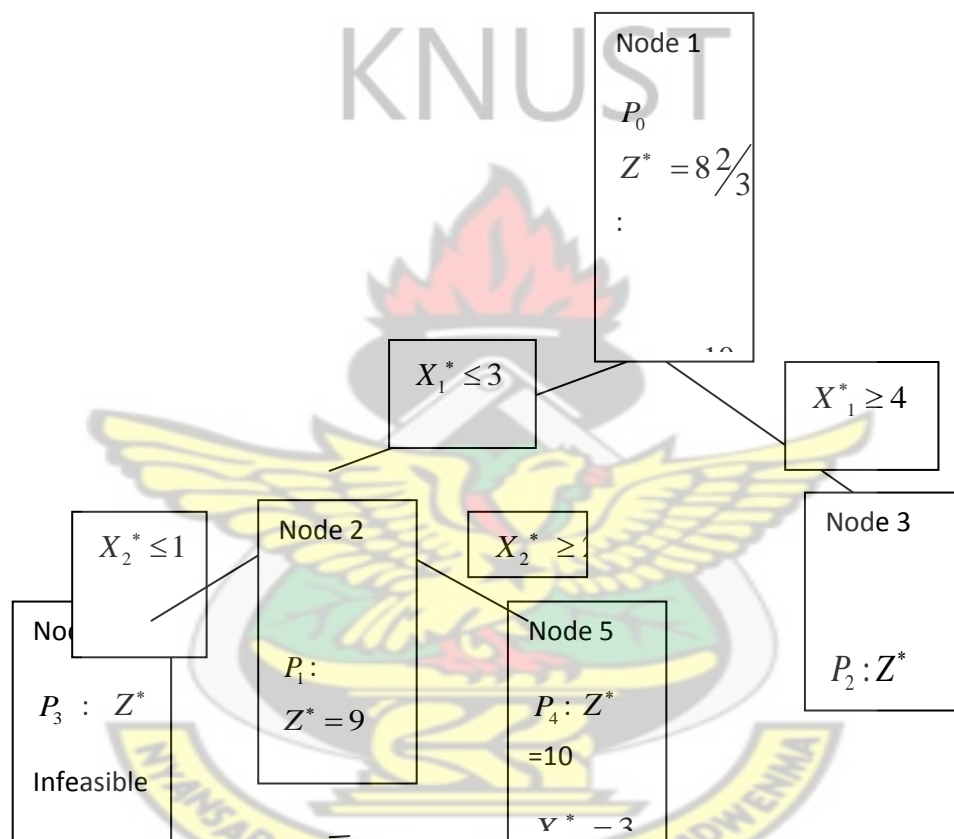


Figure 3.11.2 : The complete solution tree for Daskin's algorithm.

STEP 3 : Node 2 of problem P_1 is the only one for branching. The solutions with $X_2^* \geq \frac{3}{2}$ from problem P_1 is partitioned into two subsets, one with $X_2^* \leq [\frac{3}{2}] = 1$ and the other with $X_2^* \geq 2$. These subsets correspond to Nodes 4 and 5 respectively of problems P_3 and P_4 as shown in figure 2.4.2. Therefore the subproblem to solve at node 4 of problem P_3 is

$$P_3: \text{Minimize } Z = X_1 + 4X_2$$

$$\text{Subject to } 2X_1 + X_2 \leq 8$$

$$X_1 + 2X_2 \geq 6$$

$$X_1 \leq 3$$

$$X_2 \leq 1$$

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

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

$$\text{Minimize } Z = X_1 + 4X_2$$

$$\text{Subject to } 2X_1 + X_2 \leq 8$$

$$X_1 + 2X_2 \geq 6$$

$$X_1 \leq 3$$

$$X_2 \geq 2$$

$$X_1 \geq 0, X_2 \geq 0, X_1 \text{ and } X_2 \text{ are integers.}$$

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

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

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

3.12.1 Cutting Plane Method

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

The relaxations are gradually improved to give better approximations to the integer programming problem, at least in the neighborhood of the optimal solution. For hard instances that cannot be solved to optimality, cutting plane algorithms can produce approximations to the optimal solution in moderate computation times, with guarantees on the distance to optimality.

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

Jünger et al. (1995) contains a survey of applications of cutting plane methods, as well as a guide to the successful implementation of a cutting plane algorithm. Nemhauser and Wolse (1992) provides an excellent and detailed description of cutting plane algorithms as well as other aspects of integer programming. Research by Schrijver 1986 and his article in (Schrijver 1995) are excellent sources of cutting plane applications.

3.12.2 Using the fractional algorithm of cutting plane

In this algorithm all coefficients including the right hand side need to be integer. This condition is necessary as all variables (original, slack and artificial) are supposed to be

integer. The elements of A and b need not be integer although this can be transformed into integers as shown below.

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

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

3.12.3 Procedure for cutting plane algorithm

1. Solve the integer programming problem as a Linear Programming Problem.
2. If the optimal solution is integer stop else go to step 3.
3. Introduce secondary constraints (cut) that will push the solution towards integrality (Return to 1).

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

3.12.3 The construction of the secondary constraints:

Given the integer problem

$$\text{Minimize } Z = C^T X$$

$$\text{Subject to } AX \leq b$$

$$X \geq 0, \text{ integer}$$

X=Vector of decision variable.

C^T =Vector coefficients

A=the given matrix

B=vector coefficient

The optimal tableau of the Linear programming Problem is given in table 2.0 below:

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

$$X_B = (X_1 \dots X_M) \quad \text{and} \quad X_{NB} = (W_1 \dots W_N)$$

Table 3.12.0 Table showing the variables to be considered in the Cutting Plane Method.

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

Consider the i th equation where X_i was required to be integer but found not integer.

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

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

$$\text{Let } \beta_i = [\beta_i] + f_i \text{ and } \alpha_{ij} = [\alpha_{ij}] + g_{ij} \quad (2)$$

then

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

$$f_i - \sum_{j=1}^N (g_{ij} w_j) = X_i - [\beta_i] + \sum_{j=1}^N ([\alpha_{ij}] w_j) \quad (3)$$

KNUST

Where $[a] \leq a$ and $([a])$ is integer part of a ; $0 < f_i < 1$; $0 \leq g_{ij} < 1$

$[\beta] \leq \beta$ and $([\beta])$ is the integer part of β

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

Since all $x_i (i=1, \dots, M)$ and all $w_j (j=1, \dots, N)$ must be integer, the right-hand side is consequently integer and therefore the left-hand side is also integer thus from table 2.0

$$f_i - \sum_{j=1}^N (g_{ij} w_j) \in \mathbb{Z} \quad (\text{Integer}) \quad (4)$$

$g_{ij} \geq 0$ and $w_j \geq 0$ then from equation(3) with $X_i > [\beta_i]$ $f_i - \sum_{j=1}^N (g_{ij} w_j) \geq 0$

Therefore

$$f_i \geq f_i - \sum_{j=1}^N (g_{ij} w_j) \text{ for all } i = 1, \dots, N \quad (5)$$

Since $0 < f_i < 1$ we have $f_i - \sum_{j=1}^N (g_{ij} w_j) < 1$ and using (4) we obtain

$$f_i - \sum_{j=1}^N (g_{ij} W_j) \leq 0 \quad (6)$$

Constraint (6) is the cut and can be expressed as a secondary constraints by adding slack variable:

This gives

$$f_i - \sum_{j=1}^N (g_{ij} W_j) + S_i = 0 \rightarrow S_i = \sum_{j=1}^N (g_{ij} W_j) - f_i \quad (7)$$

for all $i = 1, \dots, M$

Where $S_i \geq 0$ (integer slack variable).

3.12.4 Choice of the cut

Suppose two rows in table 2.0 gives non-integer solutions in X_i and X_k then there will be two cuts based on X_i and X_k having the following conditions:

$$(i) \ f_i \leq \sum_{j=1}^N g_{ij} W_j$$

$$(ii) \ f_k \leq \sum_{j=1}^N g_{kj} W_j$$

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

$$(iii) \ f_i \geq f_k \text{ and } g_{ij} \leq g_{kj} \text{ for all } j$$

With the strict inequality happening at least once.

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

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

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

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

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

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

3.12.5 Prototype Example

$$\text{Maximize } Z = 7x_1 + 9x_2$$

$$\text{Subject to } -x_1 + 3x_2 \leq 6$$

$$7x_1 + x_2 \leq 35$$

$$x_1 \geq 0, x_2 \geq 0, \text{integer}$$

Solution

$$\text{Maximize } Z = 7x_1 + 9x_2 + 0s_1 + 0s_2$$

Subject to

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

$$7x_1 + x_2 + 1s_2 = 35$$

Table 3.12.1 Final Tableau for first iteration

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

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

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

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

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

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

Choice of cut

Taking equations (1) and (2)

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

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

$$\frac{1}{2} - \frac{7}{22}x_3 - \frac{1}{2}x_4 = x_2 + 0x_3 + 0x_4 \text{-----(3) integer.....(1a)}$$

$$\frac{1}{2} - \frac{21}{22}x_3 - \frac{3}{22}x_4 = x_1 - x_3 + 0x_4 \text{-----(4) integer.....(2b)}$$

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

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

Using

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

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

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

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

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

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

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

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

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

Hence (1a) would be considered to be part of the new constraints.

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

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

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

The system of equations becomes;

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

Subject to;

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

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

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

$$S_3 = X_5$$

Table 3.12.2 Final Tableau for the second iteration

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

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

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

Using the fractional algorithm;

$$x_1 + \frac{1}{7}x_4 - \frac{1}{7}x_5 = \frac{32}{7} \text{-----}(1)^*$$

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

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

$$x_3 + \frac{1}{7}x_4 - \frac{22}{7}x_5 = \frac{11}{7} \text{-----}(2)^*$$

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

$$\rightarrow x_3 + 0x_4 - 4x_5 - 1 = \frac{4}{7} - \left(\frac{1}{7}x_4 + \frac{6}{7}x_5\right) \dots\dots\dots \text{integer (2a)*}$$

Choice of Cut

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

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

Using

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

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

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

Therefore

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

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

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

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

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

Tie will be broken arbitrary by choosing equation (2)* as the new constraints to be added.

Where $s_3 = x_5$.

The system of equations becomes;

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

Subject to

$$x_2 = 3$$

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

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

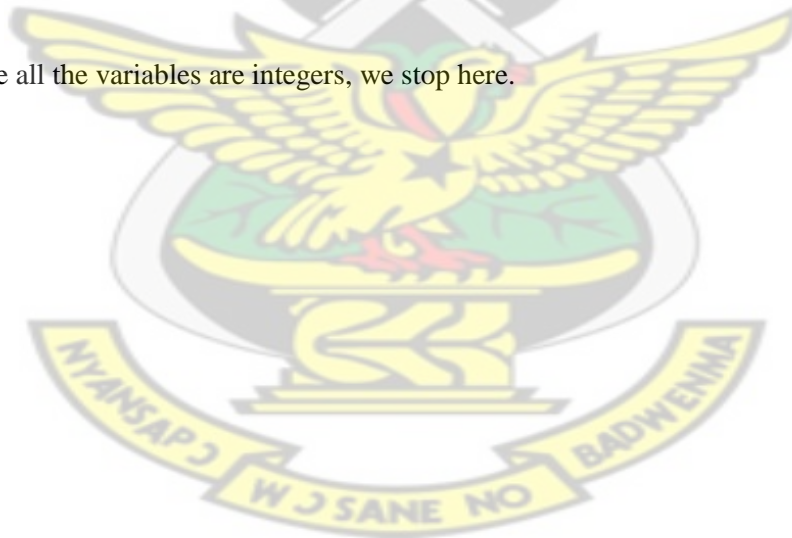
$$-\frac{1}{7}x_4 - \frac{6}{7}x_5 + s_4 = -\frac{4}{7}$$

Table 3.12.3 Final tableau for the last iteration

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

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

Since all the variables are integers, we stop here.



CHAPTER 4

COLLECTION OF DATA, ANALYSIS OF DATA AND RESULTS

4.1 Numerical Representation of Constituency Capitals

For the purpose of this work, numbers have been allocated to the twenty four constituency capitals in the Brong Ahafo Region. This is illustrated in the table below.

Constituency capital	Number Allocated
Sunyani	1
Berekum	2
Wamfie	3
Dormaa-Ahenkro	4
Drobo	5
Sampa	6
Techiman	7
Wenchi	8
Kintampo	9
Jema	10
Nkoranza	11
Atebubu-Amanteng	12
Yeji	13
Bechem	14
Duayaw Nkwanta	15
Goaso	16
Nsokor	17

Kenyase	18
Domase	19
Busuaa	20
Kwame Danso	21
Tuobodom	22
Kukuom	23
Hweddiem	24

Table 4.1: Numbers allocated to constituency capitals in the Brong Ahafo Region

4.2 Distance Matrix for the 24 Constituency Capitals in Brong Ahafo in kilometres(km)

The table below shows the distance matrix obtained from distances between the capitals of the twenty-four constituencies. For cities which have no direct link, the minimum distance along the edges is considered. The cells indicated 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.

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	24
1	0	33	56	79	64	114	64	60	123	105	92.5	263	334	48	33.5	83.7	91	43	7	110	299	71.5	100	90
2	33	0	34	56	32	82	97	93	156	138	125	126	367	81	66	107	66	138	40	143	331	104	124	132
3	56	33.5	0	23	65	116	120	116	179	161	149	319	390	104	90	74	100	105	63	166	354	128	91	98
4	79	56	23	0	88	170	143	138	202	174	172	342	413	127	113	100	122	127	86	189	378	151	113	121
5	64	32	65	88	0	50	128	124	187	169	157	327	398	112	98	139	98	170	71	174	363	136	156	163
6	114	82	115.5	170	50	0	111	80	170	152	139	246	317	162	147	189	50	219	121	157	246	118	205	213
7	64	97	120	143	128	111	0	30	59	41	29	135	206	112	98	148	60	107	71	46	171	7.5	164	154
8	60	93	116	138	124	80	30	0	89	71	59	165	236	108	94	144	30	103	67	76	201	38	160	150
9	123	156	179	202	187	170	59	89	0	18	55	115	186	171	157	207	119	166	130	72	151	67	223	213
10	105	138	161	174	169	152	41	71	18	0	37	133	204	153	139	189	101	148	112	54	168	34	205	195
11	92.5	125	149	172	157	139	29	59	55	37	0	107	178	141	126	176	89	136	100	18	142	36	193	183
12	263	126	319	342	327	246	135	165	115	133	107	0	71	311	297	347	195	306	270	86	35.5	143	363	353
13	334	367	390	413	398	317	206	236	186	204	178	71	0	382	368	418	266	377	341	157	107	214	434	424
14	48	81	104	127	112	162	112	108	171	153	141	311	382	0	15	66	139	91	55	158	347	71.5	82	42
15	33.5	66	90	113	98	147	98	94	157	139	126	297	368	15	0	80	125	76.5	41	144	332	105	116	75.5
16	83.7	107	74	100	139	189	148	144	207	189	176	347	418	66	80	0	175	30.5	91	194	382	155	16.5	23.7
17	91	66	100	122	98	50	60	30	119	101	89	195	266	139	125	175	0	134	98	106	227	70	191	181
18	43	138	105	127	170	219	107	103	166	148	136	306	377	91	76.5	30.5	134	0	50	153	342	115	47	6.5
19	7	40	63	86	71	121	71	67	130	112	100	270	341	155	41	91	98	50	0	117	306	79	107	97
20	110	143	166	189	174	157	46	76	72	54	18	86	157	158	144	194	106	153	117	0	121	54	210	200
21	299	331	354	378	363	246	171	201	151	168	142	35.5	107	347	332	382	227	342	306	121	0	178	399	389
22	71.5	104	128	151	136	118	7.5	38	67	34	36	143	214	71.5	105	155	70	115	79	54	178	0	172	162
23	100	124	91	113	156	205	164	160	223	205	193	363	434	82	116	16.5	191	47	107	210	399	172	0	40.2
24	90	132	98	121	163	213	154	150	213	195	183	353	424	42	75.5	23.7	181	6.5	97	200	389	162	40.2	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).

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

Subject to

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

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

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

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

The problem is an assignment problem with additional restrictions that guarantee the exclusion of subtours in the optimal solution. Recall that a subtour in V is a cycle that does not include all vertices (or cities). Equation (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 to on other city. Constraint (4) eliminates subtours. 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 constraints (2) and (3) we choose the random

initial tour (x^0) = 19 - 2 - 5 - 6 - 17 - 8 - 7 - 22 - 10 - 9 - 11 - 20 - 12 - 21 - 13 - 15 - 14 - 24 - 18 - 19

From objective function (1) the initial distance = $d(x^0) =$

$$d(19,2)+d(2,5)+d(5,6)+d(6,17)+d(17,8)+d(8,7)+d(7,22)+d(22,10)+d(10,9)+d(9,11)+d(11,2) \\ +d(20,12)+d(12,21)+d(21,13)+d(13,15)+d(15,14)+d(14,24)+d(24,18)+d(18,19)= 1113.5\text{km}$$

The initial temperature is taken to be $(T_0) = 4069.00$, $\alpha = 0.99$

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. Toshiba Laptop Computer with processor speed of 2.00GHz was used in finding the solution after 1601 iterations in 382.95 seconds. The execution time varied with the number of iterations.

4.5 Results

After performing 1601 iterations the optimal tour = 19 - 15 - 14 - 24 - 18 - 23 - 16 - 3 - 4 - 2 - 5 - 6 - 17 - 8 - 7 - 11 - 20 - 21 - 12 - 10 - 22 - 1 - 19

Thus,

$$d(19,15)+d(15,14)+d(14,24)+d(24,18)+d(18,23)+d(23,16)+d(16,3)+d(3,4)+d(4,2)+d(2,5)+d(5,6)+d(6,17)+d(17,8)+d(8,7)+d(7,11)+d(11,20)+d(20,21)+d(21,12)+d(12,10)+d(10,22)+d(22,1)+d(1,19) = 980\text{km}$$

The optimal tour was found to be the same after it was run ten times.

The optimal tour is therefore as follows:

Sunyani → Domase → Duayaw Nkwanta → Bechem → Hweddiem → Kenyase → Kukuom → Goaso → Wamfie → Dormaa-Ahenkro → Berekum → Drobo → Sampa → Nsorkor Wenchi → Techiman → Nkoranza → Busuaa → Kwame Danso → Atebubu Jema → Tuobodom → Sunyani

CHAPTER 5

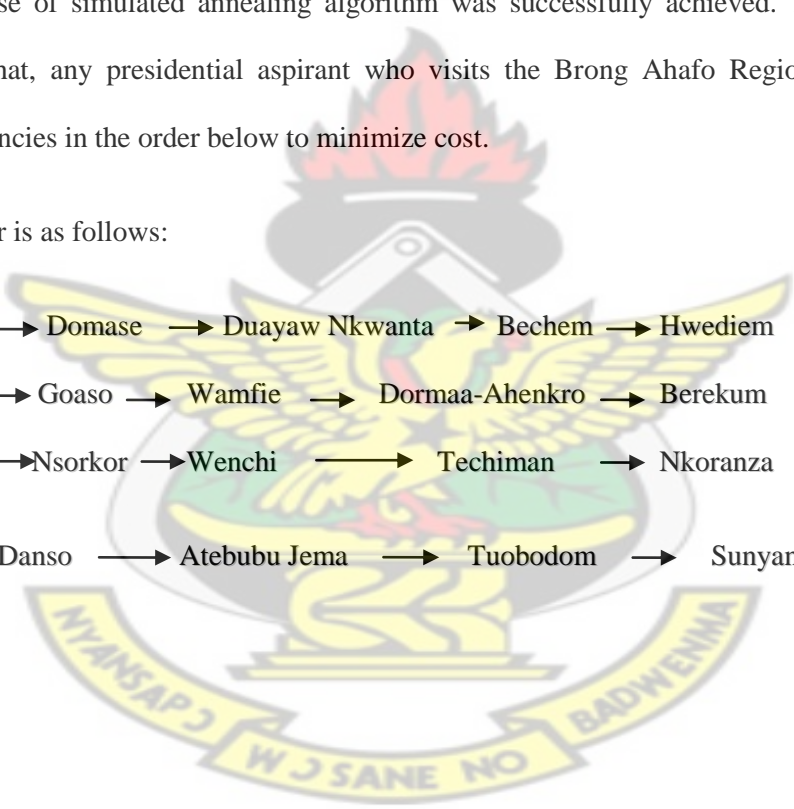
CONCLUSION AND RECOMMENDATION

5.1 Conclusion

The simulated annealing algorithm can be a useful tool to apply to hard combinatorial problems like that of TSP. Using simulated annealing as a method in solving the symmetric TSP model has been 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 Brong Ahafo Region must visit the constituencies in the order below to minimize cost.

The order is as follows:



Sunyani → Domase → Duayaw Nkwanta → Bechem → Hwediem → Kenyase →
Kukuom → Goaso → Wamfie → Dormaa-Ahenkro → Berekum → Drobo →
Sampa → Nsorkor → Wenchi → Techiman → Nkoranza → Busuaa →
Kwame Danso → Atebubu Jema → Tuobodom → Sunyani

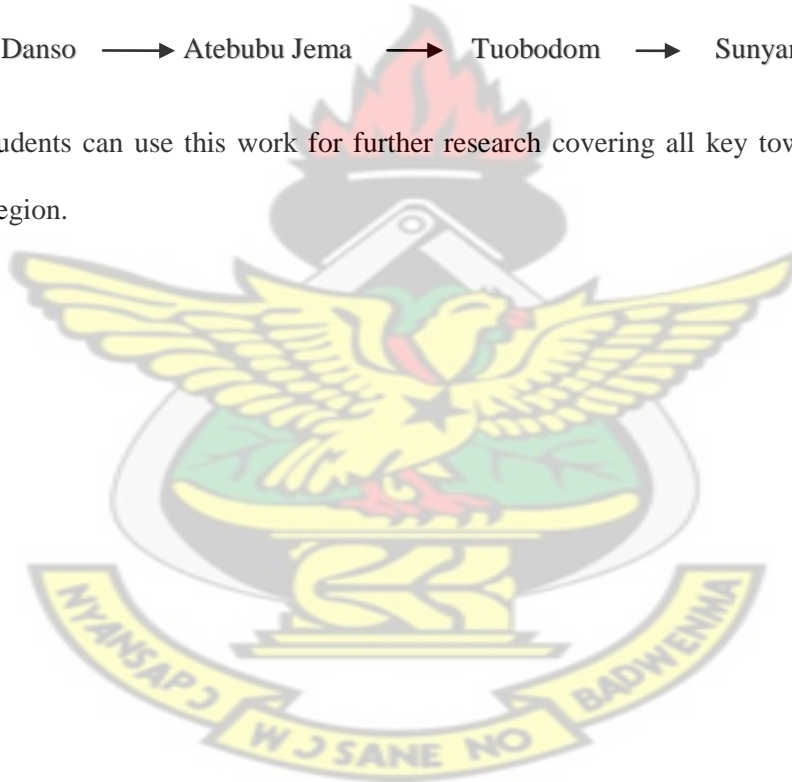
5.2 Recommendations

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

1. Presidential Candidates who visit the Brong-Ahafo Region to Campaign should consider the routes below in order to minimize their cost.

Sunyani → Domase → Duayaw Nkwanta → Bechem → Hweddiem → Kenyase →
Kukuom → Goaso → Wamfie → Dormaa-Ahenkro → Berekum → Drobo →
Sampa → Nsorkor → Wenchi → Techiman → Nkoranza → Busua →
Kwame Danso → Atebubu Jema → Tuobodom → Sunyani

2. Students can use this work for further research covering all key towns in the Brong-Ahafo Region.



REFERENCES

1. 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.
2. Ackoff, R.L., Arnoff, E.L., and Sengupta, S.S. (1961). "Mathematical Programming". In: *Progress in Operations Research*. R.L. Ackoff, editor. John Wiley and Sons: New York: NY. 105-210.
3. 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.
4. 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.
5. Applegate, D., Bixby, R., Chvatal, V., Cook, W., and Helsinghaun, K. (2004). "The Sweden Cities TSP Solution". <http://www.tsp.gatech.edu/sweeden/cities/cities.htm>.
6. 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
7. Applegate, D., Bixby, R., Chv'atal, V. and Cook, W. (2007). *The Traveling Salesman Problem*. Princeton University Press: Princeton, NJ.
8. Amponsah, S .K and F.K Darkwah (2007) .Lecture notes on operation Research ,IDL KNUST 62-67
- 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. Barahona, F., Gr'otschel, M., J'unger, M., and Reinelt, G. (1988): 'An application of combinatorial optimization to statistical physics and circuit layout design', *Operations Research* 36(3) ,493–513
11. 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.
12. Barachet, L.L. (1957). "Graphic Solution of the Traveling-Salesman Problem". *Operations Research*. 5:841-845.
13. Bellman, R. (1960). "Combinatorial Processes and Dynamic Programming". In: *Combinatorial Analysis*. R. Bellman and M. Hall, Jr., eds. American Mathematical Society: Washington, DC. 217-249.
14. Bellman, R. (1960). "Dynamic Programming Treatment of the TSP". *Journal of Association of Computing Machinery*. 9:66.
15. Bellmore, M. and Nemhauser, G.L. (1968). "The Traveling Salesman Problem: A Survey". *Operations Research*. 16:538-558.

16. Bock, F. (1958). "An Algorithm for Solving Traveling-Salesman' and Related Network Optimization Problems". Research Report, Operations Research Society of America Fourteenth National Meeting: St. Louis, MO. Problems". Research Report, Operations Research Society of America Fourteenth National Meeting: St. Louis, MO.
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. Burkard, R.E. (1979). "Traveling Salesman and Assignment Problems: A Survey". In: *Discrete Optimization I*. P.L. Hammer, E.L. Johnson, and B.H. Korte, eds. *Annals of Discrete Mathematics Vol. 4*, North-Holland: Amsterdam. 193-215.
19. Carpaneto, G., Toth, P., (1980). "Some new branching and bounding criteria for the asymmetric traveling salesman problem", *Management Science* 21, pp.736–743.
20. Charles–Owaba, O.E.(2001). "Optimality Conditions to the Accyclic Travelling Salesman Problem". *Operational Research Society of India*. 38:5.
21. 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.
22. Charles–Owaba, O.E. (2002). "Set-Sequencing Algorithm to the Cyclic TSP". *Nigerian Journal of Engineering Management*. 3(2):47-64.
23. 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.
24. Crama, Y., Van de Klundert, J., and Spieksma, F. C.R. (2002). "Production Planning Problems in Printed Circuit Board Assembly". *Discrete Applied Mathematics*. 123:339-361.
25. Croes, G.A. 1958. "A Method for Solving Travelling-Salesman Problems". *Operations Research*. 6:791-812.
26. Crowder, H. and Padberg, M.W. (1980). "Solving Large-Scale Symmetric Travelling Salesman Problems to Optimality". *Management Science*. 26:495-509.
27. Cerny, V. (1985) "Thermodynamical Approach to the Traveling Salesman Problem: An Efficient Simulation Algorithm", *J. Opt. Theory Appl.*, 45, 1, 41-51.
28. 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*, 1st edition (facsimile-1964), Harvard University Press, MA
29. 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
30. Dantzig, G.B., Fulkerson, D.R., and Johnson, S.M. (1954). "Solution of a Large-Scale Traveling-Salesman Problem". *Operations Research*. 2: 393-410.
31. Dawkins, R., (1986). *The Blind Watchmaker*, Penguin, London.
32. David Goldberg, (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning*.

33. Deineko Addison-Wesley, V.G., Hoffmann, M., Okamoto, Y. and Woeginger, G.J.(2006). "The Traveling Salesman Problem with Few Inner Points". *Operations Research Letters*. 34(1):106-110.
34. Eastman, W.L. (1958). "Linear Programming with Pattern Constraints". Ph.D. Dissertation. Harvard University: Cambridge, MA.
35. 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.
36. Flood, M.M.(1956). "The TSP". *Operation Research*. 4:6.
- 37.Foulds, L.R. and Hamacher, H.W. (1993). "Optimal Bin Location and Sequencing in Printed Circuit Board Assembly." *European Journal of Operational Research (Netherlands)*. 66(3):279-290.
38. Goldberg, D.E, (1989). *Genetic Algorithm for Search,Optimization and Machine Learning*, Addison-Wesley.
- 39.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).
- 40.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.
- 41.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.
- 42.Gonzales, R.H. (1962). "Solution to the Traveling Salesman Problem by Dynamic Programming on the Hypercube". Technical Report Number 18, Operations Research Center, Massachusetts Institute of Technology: Boston, MA.
- 43.Grötschel, M., Junger, M., and Reinelt, G. (1984): 'A cutting plane algorithm for the linear ordering problem', *Operations Research* 32, 1195
- 44.Garey M.R. and Johnson D.S. (1979).*Computers and Intractability: A Guide to the Theory of NP Completeness*, W.H. Freeman, San Francisco.Hitchcock F.L., The Distribution of Product from Several Sources to Numerous Localities, *J. Math. Phys.*, 20, No. 2, 1941, 217–224.
- 45.Grötschel, M., Martin, A., and Weismantel, R. (1996),: 'Packing Steiner trees: a cutting plane algorithm and computational results', *Mathematical Programming* 72 ,125–145.
- 46.Grötschel, M. (1980). "On the Symmetric Travelling Salesman Problem: Solution of a 120-City Problem". *Mathematical Programming Study*. 12: 61-77.
- 47.Günther, H.O., Gronalt, M., and Zeller, R. (1998). "Job Sequencing and Component Set-Up on a Surface Mount Placement Machine." *Production Planning & Control*. 9(2):201–211.

48. Gutin, G. and Punnen, J. (2002). *The TSP and its Variations*. Kluwer Academic Publishers.
49. Grötschel M and Padberg M., (1993). "Ulysses 2000: In Search of Optimal Solutions to Hard Combinatorial Problems," Technical Report, New York University Stern School of Business.
50. Gerard Reinelt ,(1994). *The Traveling Salesman: Computational Solutions for TSP Applications*. Springer-Verlag.
51. 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.
52. Golden B.L. and Assad A.A., (1988). *Vehicle Routing: Methods and Studies*, Elsevier Science, Amsterdam
53. Hillier F.S and Lieberman G.J (2005): *Introduction to operations Research*. McGraw-hill companies inc, 1221 Avenue of the Americas, New York ,NY 10020.
53. Hoffman A. J. and Wolfe P. (1985), "History" in *The Traveling Salesman Problem*, Lawler, Lenstra, Rinnooy Kan and Shmoys, eds., Wiley, 1-16
- Holland, J, (1975). *Adaptation in Natural and Artificial Systems*, Michigan University Press
54. Hoffman K.L. and Padberg M., (1991) LP-based Combinatorial Problem Solving, *Annals Operations Research*, 4, 145–194.
55. 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
56. Helbig, H.K. and Krarup, J. (1974). "Improvements of the Held-Karp Algorithm for the Symmetric Traveling-Salesman Problem". *Mathematical Programming*. 7:87-96.
57. 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.
58. Held, M. and Karp, R.M. (1970). "The Traveling-Salesman Problem and Minimum Spanning Trees". *Operations Research*. 18:1138-1162.
59. Held, M. and Karp, R.M. (1971). "The Traveling-Salesman Problem and Minimum Spanning Trees: Part II". *Mathematical Programming*. 1:6-25.
60. Heller, I. (1955). "On the Travelling Salesman's Problem". *Proceedings of the Second Symposium in Linear Programming*: Washington, D.C. Vol. 1.
61. Hong, S. (1972). "A Linear Programming Approach for the Traveling Salesman Problem". Ph.D. Thesis. The Johns Hopkins University: Baltimore, MD.
62. Johnson D.S and Mcgeoch L.A (2002): *Experimental Analysis of Heuristics for STSP*, In *The Traveling Salesman Problem and its Variations* (G. Gutin and A. P. Punnen, eds.), Kluwer.

62. 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 its Variations*. Kluwer Academic Publishers.
63. Kahng, A.B. and Reda, S. (2004). 'Match Twice and Stitch: A new TSP Tour Construction Heuristic'. *Operations Research Letters*. 32(6): 499-509.
64. Karg, R.L. and. Thompson, G.L. (1964). "A Heuristic Approach to Solving Travelling Salesman Problems". *Management Science*. 10:225-248.
65. Keuthen, R. (2003). "Heuristic Approaches for Routing Optimization". PhD thesis at the University of Nottingham: UK.
66. Kolohan, F. and Liang, M. (2000). "Optimization of Hole Making: A Tabusearch Approach". *International Journal of Machine Tools & Manufacture*. 50:1735-1753.
67. Kruskal, 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.
68. Karp, R.M., Steele, J.M., (1990), "Probabilistic Analysis of Heuristics. In: *The Traveling Salesman Problem*", Wiley, New York, pp. 181–205 .
69. Kirkpatrick, S., Gelatt, C.D. Jr., and Vecchi, M.P. (1983). Optimizations by simulated annealing. *Science*, 220, 671-681.
70. Kwon, S., Kim, H., and Kang, M. (2005). "Determination of the Candidate Arc Set for the Asymmetric Traveling Salesman Problem". *Computers and Operations Research*. 32(5): 1045-1057.
71. Kirkpatrick, S., C. D. Gelatt Jr., M. P. Vecchi, (1983) "Optimization by Simulated Annealing", *Science*, 220, 4598, 671-680.
72. Lambert, F. (1960). "The Traveling-Salesman Problem". *Cahiers du Centre de Recherche Opérationnelle*. 2:180-191.
73. Lawler, E.L. and Wood, D.E. (1966). "Branch-and-Bound Methods: A Survey". *Operations Research*. 14:699-719.
74. 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.
75. Lin, S. and Kernighan, B.W. (1973). "An Effective Heuristic Algorithm for the Traveling-Salesman Problem". *Operations Research*. 21:498-516.
76. 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.
77. 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.
78. 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.

79. Oladokun, V.O. (2006). "The Development of a Subtour-Free Set Sequencing Algorithm and the Software for Solving the Machine Set-Up Problem". Ph.D. thesis at the University Of Ibadan: Ibadan, Nigeria.
80. 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.
81. Pinedo, M. (1995). *Scheduling Theory, Algorithm and Systems*. Prentice Hall Pub: New Jersey.
82. Potvin, J.Y. (1996). "The Traveling Salesman Problem: A Neural Network Perspective". *ORSA Journal on Computing*. 5:328-347.
83. Rego C and Glover, F (2002) Local Search and Metaheuristic, in G. Gutin and A.P. Punnen (eds.): *The Traveling Salesman Problem and its Variations* , Kluwer, Dordrecht.
84. Radin, L.R. (1998). *Optimisation in Operations Research*. Prentice Hall Inc. New Jersey.
85. Rajkumar, K. and Narendran, T.T. (1996). "A Heuristic for Sequencing PCB Assembly to Minimize Set-up Times". *Production Planning & Control*. 9(5): 465–476.
86. Raymond, T.C. (1969). "Heuristic Algorithm for the Traveling-Salesman Problem". *IBM Journal of Research and Development*. 13:400-407.
87. Riera-Ledesma, J. and Salazar-González, J.J. (2005). "A Heuristic Approach for The Travelling Purchaser Problem". *European Journal of Operations Research*. 162(1):142-152.
88. Robacker, J.T. (1955). "Some Experiments on the Traveling-Salesman Problem". RAND Research Memorandum.
89. Roberts, S.M. and Flores, B. (1966). "An Engineering Approach to the Traveling Salesman Problem". *Management Science*. 13:269-288.
90. Robinson, B. (1949). "On the Hamiltonian Game (A Traveling-Salesman Problem)". RAND Research Memorandum.
91. Rossman, M.J and Twery, R.J. (1958). "A Solution to the Travelling Salesman Problem". *Operations Research*. 6:687.
92. Shapiro, D. (1966). "Algorithms for the Solution of the Optimal Cost Traveling Salesman Problem". Sc.D. Thesis, Washington University: St. Louis, MO.
93. 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.
94. Turkensteen, M., Ghosh, D., Goldengorin, B., Sierksma, G., (2007), "Tolerance-based Branch and Bound algorithms for the ATSP", *European Journal of Operational Research* 189: 775–788, Available online at www.sciencedirect.com
95. Tian, P. and Yang, S. (1993). An Improved Simulated Annealing Algorithm with Generic Characteristics and Travelling Salesman Problem. *Journal of Information and Optimization Science*. 14(3):241-254.

96. Volgenant, T. and Jonker, R. (1982). A Branch and Bound Algorithm for the Symmetric Traveling Salesman Problem Based on the 1-Tree Relaxation. *European Journal of Operational Research*. 9:83-89.
97. Walshaw, C.A. (2001). Multilevel Lin-Kernighan-Helsgaun Algorithm for the Travelling Salesman Problem. CMS press Centre for Numerical Modelling and process Analysis. University of Greenwich: London, UK.
98. Walshaw, C.A. (2002). Multilevel Approach to the Travelling Salesman Problem. *Operations Research*. 50(5):862-877.
99. Lawler E.L., Lenstra J.K., Rinnooy Kan A.H.G, and Shmoys D.B., (1986).. The Traveling Salesman. JohnWiley and Sons .
100. Rachev and Ruschendorf (1993), constrained transportation Problems, Decision and Control, Proceedings of the 32nd IEEE conference ,Volume 3, Pages 2896-2900
101. Yang, J ,Wu C, Lee H, Liang Y, (2008) .Solving traveling Salesman problems generalized chromosome genetic algorithm ,Progress in Natural Science, Volume 18, Issue 7, 10th July Pages 887-892.
102. Hatfield, D. J. AND J. F. Pierce, (1966). Production Sequencing by Combinatorial Programming, IBM Cambridge Scientific Center, Cambridge, Mass.
103. Lawler and. Wood D. E, (1966) Branch-and-Bound Methods: A Survey, Opns. Res. 14, 69-719
104. Little, J. D. C., K. G. Murty , D. W. Sweeney, AND C. Karel , (1963). An Algorithm for the Traveling Salesman Problem, Opns. Res. 11, 979-989 .
105. Shapiro, D., (1966) Algorithms for the Solution of the Optimal Cost Traveling Salesman Problem, Sc.D. Thesis, Washington University, St. Louis,
106. Gr'otschel, M., and Holland, (1991) O.: 'Solution of large-scale travelling salesman problems', Mathematical Programming 51(2), 141–202.
107. 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.
108. De Simone, C., Diehl, M., Junger, M., Mutzel, P., Reinelt, G., and Rinaldi, G. (1995): 'Exact ground states of Ising spin glasses: New experimental results with a branch and cut algorithm', Journal of Statistical Physics 80 ,487–496.
109. Mitchell, J. E (1997): Computational experience with an interior point cutting plane algorithm, Tech. rep., Mathematical Sciences, Rensselaer Polytechnic Institute, Troy, NY 12180–3590.
110. Nemhauser, G. L., and Sigismondi, G. (1992): A strong cutting plane/branch-and-bound algorithm for node packing, Journal of the Operational Research Society 43 443–457.
111. Nemhauser, G. L., and Wolsey, L. A. (1988): Integer and Combinatorial Optimization, John Wiley, New York.
112. Schrijver, A. (1986): Theory of Linear and Integer Programming, John Wiley, Chichester.

113. Schrijver, A. (1995): 'Polyhedral Combinatorics', Handbook of Combinatorics, in R.L. Graham, M. Grötschel, and L. Lovász (eds.), Vol. 2. Elsevier Science, ch. 30, pp. 1649–1704.
114. Mitchell, J. E., and Borchers, B. (1996): Solving real world linear ordering problems using a primal-dual interior point cutting plane method, *Annals of Operations Research* 62, 253–276.
115. Miller, D. and Pekny, J. (1991), Exact Solution of Large Asymmetric Traveling Salesman Problems, *Science*, 251: 754–761
116. Mitchell, J. E., and Borchers, B. (1997): Solving linear ordering problems with a combined interior point/simplex cutting plane algorithm, Tech. rep., Mathematical Sciences, Rensselaer Polytechnic Institute, Troy, NY 12180–3590, , Accepted for publication in Proceedings of HPOPT97, Rotterdam, The Netherlands.
117. Metropolis, M., Rosenbluth, A., Rosenbluth, M., Teller, A., and Teller, E. (1953). Equation of state calculations by fast computing machines. *Journal of Chemical Physics*, 21, 1087-1092.
118. 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.
119. Notes on Tabu Search Retrieved from <http://itc.ktu.it/itc32/Misev32.pdf>
120. Padberg M. and Rinaldi G. (1991) A branch-and-cut algorithm for the resolution of large-scale traveling salesman problem, *SIAM Review*, 33, 60–100.
121. Papadimitriou C.H. and Steiglitz K. (1982). *Combinatorial Optimization: Algorithms and Complexity*, Prentice-Hall Inc., Englewood Cliffs, New Jersey
125. Wikipedia, the free encyclopedia - Moore's law (2009). Retrieved from http://en.wikipedia.org/wiki/Moore's_law
126. Zhang, W. (2004). Phase Transitions and Backbones of the Asymmetric Traveling Salesman Problem. *Journal of Artificial Intelligence Research*. 21:471-497.

APPENDIX A

Matlab Program

```
%function simanneal()

% *****Read distance (cost) matrix from Table 3.2 *****

clc

d = xlsread('dist.xls');

d_orig = d;

start_time = cputime;

summ=0;

dim1 = size(d,1);

dim12 = size(d);

for i=1:dim1

d(i,i)=10e+06;

end

for i=1:dim1-1

for j=i+1:dim1

d(j,i)=d(i,j);

end

end

%d

% *****Initialise all parameters*****

d1=d;

tour = zeros(dim12);

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

start_time = cputime;

diary output.txt

% *****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:dim1-1

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:dim1-1

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');

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;

accept_ratio =1;

%*****Perform the iteration until one of the criteria is met*****

%1. Max number of iterations reached*****

%2. Acceptance Ratio is less than the threshold

%3. No improvement in last fixed number of iterations

while (iter < Iter_max && accept_ratio > Rf)

iter = iter+1;

trans_tried = 0;

trans_accept = 0;

while(trans_tried < Lmax && trans_accept < ATmax)

trans_tried = trans_tried + 1;

```

```

city1 = round(random('uniform', 1, dim1-1));

city2 = round(random('uniform', 1, dim1-1));

while (city2 == city1)

city2 = round(random('uniform', 1, dim1-1));

end

if (city2>city1)

i=city1;

j=city2;

else

i=city2;

j=city1;

end

nbr(i)=crnt_tour(j);

nbr(j)=crnt_tour(i);

if i==1

if j-i==1

nbr_cost=crnt_tour_cost-d(1,crnt_tour(i))+d(1,crnt_tour(j))-

d(crnt_tour(j),crnt_tour(j+1))+d(crnt_tour(i),crnt_tour(j+1));

else

nbr_cost=crnt_tour_cost-d(1,crnt_tour(i))+d(1,crnt_tour(j))-

d(crnt_tour(j),crnt_tour(j+1))+d(crnt_tour(i),crnt_tour(j+1))-

d(crnt_tour(i),crnt_tour(i+1))+d(crnt_tour(j),crnt_tour(i+1))-d(crnt_tour(j-

1),crnt_tour(j))+d(crnt_tour(j-1),crnt_tour(i));

end

else

if j-i==1

nbr_cost=crnt_tour_cost-d(crnt_tour(i-1),crnt_tour(i))+d(crnt_tour(i-1),crnt_tour(j))-

d(crnt_tour(j),crnt_tour(j+1))+d(crnt_tour(i),crnt_tour(j+1));

else

```

```

nbr_cost=crnt_tour_cost-d(crnt_tour(i-1),crnt_tour(i))+d(crnt_tour(i-1),crnt_tour(j))-
d(crnt_tour(j),crnt_tour(j+1))+d(crnt_tour(i),crnt_tour(j+1))-
d(crnt_tour(i),crnt_tour(i+1))+d(crnt_tour(j),crnt_tour(i+1))-d(crnt_tour(j-
1),crnt_tour(j))+d(crnt_tour(j-1),crnt_tour(i));

end

end

delta = nbr_cost - crnt_tour_cost;

prob1 = exp(-delta/T);

prob2 = random('uniform',0,1);
if(delta < 0 || prob2 < prob1)

summ = summ+delta;

crnt_tour = nbr;

crnt_tour_cost = nbr_cost;

trans_acpt = trans_acpt + 1;

if crnt_tour_cost < best_obj
best_obj = crnt_tour_cost;
best_tour = crnt_tour;
end
else

nbr = crnt_tour;

nbr_cost = crnt_tour_cost;

end

end

accept_ratio = trans_acpt/trans_tried;

fprintf("\niter# = %d\t, T = %2.2f\t, obj = %d\t, accept ratio=%2.2f", iter,T,crnt_tour_cost,accept_ratio);

if crnt_tour_cost == obj_prev

iter_snc_last_chng = iter_snc_last_chng + 1;

else

iter_snc_last_chng = 0;

```

```

end

if iter_snc_last_chng == 10

fprintf('\n No change since last 10 iterations');

break;

end

obj_prev = crnt_tour_cost;

T = alfa*T;

iter = iter + 1;

end

fprintf('\nbest obj = %d', best_obj);

fprintf('\n best tour\n');

best_tour

end_time = cputime;

exec_time = end_time - start_time;

fprintf('\ntime taken = %f\n', exec_time);

diary off

```

