PERFORMANCE COMPARISON OF EXACT AND HEURISTIC ALGORITHMS FOR TRAVELLING SALESMAN PROBLEM

THIDA AYE

M.C.Sc. (THESIS)

JULY, 2016

PERFORMANCE COMPARISON OF EXACT AND HEURISTIC ALGORITHMS FOR TRAVELLING SALESMAN PROBLEM

THIDA AYE

M.C.Sc. (THESIS)

JULY, 2016

PERFORMANCE COMPARISON OF EXACT AND HEURISTIC ALGORITHMS FOR TRAVELLING SALESMAN PROBLEM

By

THIDA AYE

B.C.Sc. (Hons:)

Dissertation submitted in partial fulfillment of the requirements for the degree of

Master of Computer Science

(M.C.Sc.)

Of the

University of Computer Studies, Mandalay

JULY 2016

ACKNOWLEDGEMENT

I wish to dedicate this thesis to my parents and teachers to whom I owe a debt of gratitude.

I would like to thank **Dr. Moe Pwint**, Rector of the University of Computer Studies, Mandalay, for giving valuable advice at seminars and kind permission to develop this thesis.

I would like to thank my deep appreciation to my supervisor, **Dr. San San Mon**, Professor, Head of Faculty of Computing of the University of Computer Studies, Mandalay, for her close supervision, helpful advice, encouragement, numerous invaluable suggestions and comments.

I am greatly thankful to my Dean of Master class, **Dr. Myat Myat Min**, Associate Professor, Head of Faculty of Computer Science of the University of Computer Studies, Mandalay, for her valuable suggestion and priceless guidance.

Especially I would like to thank my teacher, **Daw Thwe Thwe Maw**, Assistant Lecturer of the Department of Linguistics of the University of Computer Studies, Mandalay, for editing my thesis from language point of view.

I also thank all the teachers who have enabled me to obtain this M.C.Sc. degree.

Finally, I would like to express grateful thanks to all the friends of U.C.S.M and many colleagues for their contribution in the making thesis complete.

ABSTRACT

The Travelling Salesman problem is one of the most popular problems from the NP set and it is also one of the hardest too. However even the most common deterministic solutions to the problem are known to possess an exponential time complexity. There have been any efforts to provide time efficient solutions for the problem, both exact and approximate or heuristic algorithm. So this system finds whether which one is more efficient than another between exact and heuristic algorithms. Branch and bound algorithm is one of exact algorithms and genetic algorithm is heuristic. In this system, Branch and bound algorithm and genetic algorithm are used to compare their performance in execution time and their result tours with optimal costs. This system intends for routing of a salesman of company in the whole Shan State of Myanmar to find the shortest path for them by using the above two methods.

CONTENTS

LIST OF FIGURES

LIST OF EQUATIONS

LIST OF TABLES

CHAPTER 1
INTRODUCTION

Travelling Salesman Problem has been of the great interest for many years. It is a problem in combinatorial optimization studied in both, operations research and theoretical computer science. Given a list of cities and their pair wise distances, the task is to find a shortest possible tour that visits each city exactly once and no cities can be skipped. A salesman, starting in one city, wishes to visit each of n -1 other cities and return to the start.

On the basis of the structure of the cost matrix, the TSPs are classified into two groups – symmetric and asymmetric. The TSP is symmetric if $c_{ij} = c_{ji}$, for all i, j and asymmetric otherwise. For an n-city asymmetric TSP, there is $(n\text{-}1)!$, possible solutions, one or more of which gives the minimum cost. For an n-city symmetric TSP, there are $(n - 1)! / 2$ possible solutions along with their reverse cyclic permutations having the same total cost. In either case the number of solutions becomes extremely large for even moderately large n so that an exhaustive search is impracticable.

The travelling salesman problem plays an important role in different solving algorithms. There are different approaches for solving the TSP [15]. Branch and Bound (B&B) algorithm is by far the most widely used tool that solves a large scale *NP*-hard combinatorial optimization problem. This algorithm searches the complete space of solutions for a given problem for the best solution.

Genetic algorithm is one of the best heuristic algorithms that have been used widely to solve the TSP instances which can find exactly optimal solutions to TSP in reasonable time. It is a search heuristic that

mimics the process of natural selection. This heuristic is routinely used to generate useful solutions of optimization and search problems. In hand, branch and bound (B&B) has a simple modeling, on the other hand a genetic algorithm has a difficult solution way.

The travelling salesman problem finds many applications in real life, mainly in transportation and logistics domains: route planning, job scheduling, electronic circuit board drilling, integrated circuit fabrication, etc.

1.1 Objectives of the Thesis

- to know the difference between the exact and heuristic algorithms for travelling salesman problem
- to learn the nature of genetic algorithm and branch and bound algorithm
- to minimize the total distance travelled in a given tour
- to compare genetic algorithm with branch and bound algorithm

1.2 Overview of the Thesis

The system intends to find the shortest path for a given places applying Little's B&B algorithm and Genetic Algorithm (GA). It is also comparison of the CPU execution time and cost between both algorithms.

The user firstly specifies the destination to find the route with minimum cost. The system constructs then the cost matrix as input for both algorithms according to the cities selected by the user. When the system performs the task as finding optimal route for the tour using GA, the user defines the number of iteration, the number of population, crossover rate and mutation rate. GA operates by using these values and the result of GA depends on them.

1.3 Organization of the Thesis

This Chapter describes the introduction of the travelling salesman problems and points to the objectives and overview of the system. Next, Chapter 2 describes about theoretical background. System design and the implementation of the system are shown in Chapter 3. The experiment of the system is described in Chapter 4. Chapter 5 includes the conclusion, the limitation and the further extension of the system.

CHAPTER 2
THEORETICAL BACKGROUND

2.1 History

The origins of the travelling salesman problem are unclear. A handbook for travelling salesmen from 1832 mentions the problem and includes example tours through Germany and Switzerland, but contains no mathematical treatment [16]. The travelling salesman problem was mathematically formulated in the 1800s by the Irish mathematician W. R. Hamilton and by the British mathematician Thomas Kirkman. Hamilton's Icosian Game was a recreational puzzle based on finding a Hamiltonian cycle [6]. The general form of the TSP appears to have been first studied by mathematicians during the 1930s in Vienna and at Harvard, notably by Karl Menger, who defines the problem, considers the obvious brute-force algorithm, and observes the non-optimality of the nearest neighbor heuristic: Hassler Whitney at Princeton University introduced the name *travelling salesman problem* soon after [9].

In the 1950s and 1960s, the problem became increasingly popular in scientific circles in Europe and the USA. Notable contributions were made by George Dantzig, Delbert Ray Fulkerson and Selmer M. Johnson at the RAND Corporation in Santa Monica, who expressed the problem as an integer linear program and developed the cutting plane method for its solution. With these new methods they solved an instance with 49 cities to optimality by constructing a tour and proving that no other tour could be shorter. In the following decades, the problem was studied by many researchers from mathematics, computer science, chemistry, physics, and other sciences.

Richard M. Karp showed in 1972 that the Hamiltonian cycle problem was NP-complete, which implies the NP-hardness of TSP. This supplied a mathematical explanation for the apparent computational difficulty of finding optimal tours. Great progress was made in the late 1970s and 1980, when Grötschel, Padberg, Rinaldi and others managed to exactly solve instances with up to 2392 cities, using cutting planes and branch-and-bound.

In the 1990s, Applegate, Bixby, Chvátal, and Cook developed the program *Concorde* that has been used in many recent record solutions. Gerhard Reinelt published the TSPLIB in 1991, a collection of benchmark instances of varying difficulty, which has been used by many research groups for comparing results. In 2006, Cook and others computed an optimal tour through an 85,900-city instance given by a microchip layout problem, currently the largest solved TSPLIB instance. For many other instances with millions of cities, solutions can be found that are guaranteed to be within 2-3% of an optimal tour [2].

2.2 Travelling Salesman Problem

Traveling Salesman Problem is an extremely important problem in operational research. TSP or Hamiltonian tour is a type of classic and problem which shows one from to solve the more complex ones. Hamiltonian tour starts with a given edge, visits each of the specific groups of edges and then returns to the original point of departure. TSP is the shortest walk in a circuit provided that it passes from all vertices only once.

In other words, TSP of NP-Hard problem class is known as one of the well-known combinatorial optimization problems. This means for TSP, the solution techniques have not been improved in polynomial time. That's why to solve TSP, there are many intuitive techniques. More

precisely, it is complete for the complexity class (FP^{NP}), and the decision problem version is NP-complete. If an efficient algorithm is found for the TSP problem, then efficient algorithms could be found for all other problems in the NP-complete class. Although it has been shown that, theoretically, the Euclidean TSP is equally hard with respect to the general TSP, it is known that there exists a sub exponential time algorithm for it. The most direct solution for a TSP problem would be to calculate the number of different tours through cities V. Given a starting city, it has V-1 choices for the second city, V-2 choices for the third city, etc. Multiplying these together one gets (V-1)! for one city and V! for the cities. Another solution is to try all the permutations (ordered combinations) and see which one is cheapest. At the end, the order is also factorial of the number of cities. Briefly, the solutions which appear in the literature are quite similar. The factorial algorithm's complexity motivated the research in two attack lines: exact algorithms or heuristics algorithms. The exact algorithms search for an optimal solution through the use of branch-and-bound, linear programming or branch-and-bound plus cut based on linear programming techniques. Heuristic solutions are approximation algorithms that reach an approximate solution (close to the optimal) in a time fraction of the exact algorithm. TSP heuristic algorithms might be based on genetic and evolutionary algorithms, simulated annealing, Tabu search, neural network, ant system, among some states in [4].

TSP is categorized in two classes: Symmetric TSP and Asymmetric TSP. In the *symmetric TSP*, the distance between two cities is the same in each opposite direction, forming an undirected graph. This symmetry halves the number of possible solutions. In the *asymmetric TSP*, paths may not exist in both directions or the distances might be different, forming a directed graph. Traffic collisions, one-way streets, and airfares

for cities with different departure and arrival fees are examples of how this symmetry could break down.

2.2.1 Mathematical Model

The TSP can be defined on a complete undirected graph $G = (V, E)$ or on a directed graph $G = (V, A)$. The set V = {1, 2, …, n} is the vertex set, $E = \{(i, j): i, j \in V, i < j\}$ is an edge set and $A = \{(i, j): i, j \in V, i \neq j\}$ is an arc set [12]. A cost matrix $C = (c_{ij})$ is defined on E or on A. The cost matrix satisfies the triangle inequality whenever $c_{ij} \leq c_{ik} + c_{kj}$, for all i, j, k. In a planar problem, the vertices are points $P_i = (X_i Y_i)$ in the plane, and $c_{ij} = \sqrt{(X_i - X_j)^2 + (Y_i - Y_j)^2}$ is the Euclidean distance [5]. The triangle inequality is also satisfied if c_{ij} is the length of the shortest path from i to j on G.

2.2.2 Integer Linear Programming Formulation

TSP can be formulated as an integer linear program. Label the cities with the numbers 0, ..., n and define:

$$x_{ij} = \begin{cases} 1, & \text{the path goes from city } i \text{ to city } j \\ 0, & \text{otherwise} \end{cases} \qquad (2.1)$$

For $i = 1, ..., n$, let μ_i be an artificial variables, and finally take c_{ij} to be the distance from city i to city j. Then TSP can be written as the following integer linear programming problem :

$$\min \sum_{i=0}^{n} \sum_{j \neq i, j=0}^{n} c_{ij} x_{ij} \qquad (2.2)$$

Subject to

$$0 \leq x_{ij} \leq 1 \qquad i, j = 0, \ldots, n$$

$$u_i \in Z \qquad i = 0, \ldots, n$$

$$\sum_{i=0, i\neq j}^{n} x_{ij} = 1 \qquad j = 0, \ldots, n$$

$$\sum_{j=0, i\neq j}^{n} x_{ij} = 1 \qquad i = 0, \ldots, n$$

$$u_i - u_j + n x_{ij} \leq n - 1 \qquad 1 \leq i \neq j \leq n$$

The first set of equalities requires that each city be arrived at from exactly one other city, and the second set of equalities requires that from each city there is a departure to exactly one other city. The last constraints enforce that there is only a single tour covering all cities, and not two or more disjointed tours that only collectively cover all cities. To prove this, it is shown that every feasible solution contains only one closed sequence of cities, and that for every single tour covering all cities, there are values for the dummy variables μ_i that satisfy the constraints.

To prove that every feasible solution contains only one closed sequence of cities, it suffices to show that every subtour in a feasible solution passes through city 0 (noting that the equalities ensure there can only be one such tour). For if we sum all the inequalities corresponding to $x_{ij} = 1$ for any subtour of k steps not passing through city 0, we obtain:

$$nk \leq (n-1)k, \tag{2.3}$$

which is a contradiction.

It now must be shown that for every single tour covering all cities, there are values for the dummy variables u_i that satisfy the constraints. Without loss of generality, define the tour as originating (and ending) at city 0. Choose $\mu_i = t$ if city i is visited in step t ($i, t = 1, 2, \ldots, n$). Then

$$u_i - u_j \leq n - 1, \tag{2.4}$$

since μ_i can be no greater than n and μ_j can be no less than 1; hence the constraints are satisfied whenever $x_{ij} = 0$. For $x_{ij} = 1$, we have:

$$u_i - u_j + nx_{ij} = (t) - (t+1) + n = n - 1, \quad (2.5)$$

satisfying the constraint.

2.3 Computing a Solution for TSP

TSP is the shortest possible route that visits each city exactly once and returns to the origin city. The TSP may be solved into two classes:

- *Exact* algorithms;
- *Approximate* or *heuristic* algorithms.

2.3.1 Exact Algorithms

The *exact* algorithms are guaranteed to find the optimal solution in a bounded number of steps. The most direct solution would be to try all permutations (ordered combinations) and see which one is cheapest (using brute force search). The running time for this approach lies within a polynomial factor of $O(n!)$, the factorial of the number of cities, so this solution becomes impractical even for only 20 cities. One of the earliest applications of dynamic programming is the Held–Karp algorithm that solves the problem in time $O(n^2 2^n)$ [17]. Improving these time bounds seems to be difficult. For example, it has not been determined whether an exact algorithm for TSP that runs in time $O(1.9999^n)$ exists [16]. Other approaches include:

- Various branch-and-bound algorithms, which can be used to process TSPs containing 40–60 cities.
- Progressive improvement algorithms which use techniques reminiscent of linear programming. Works well for up to 200 cities.
- Implementations of branch-and-bound and problem-specific cut generation (branch-and-cut); this is the method of choice for

solving large instances. This approach holds the current record, solving an instance with 85,900 cities; see Applegate et al. (2006).

2.3.2 Heuristic and Approximation Algorithms

The goal of an approximation algorithm is to come as close as possible to the optimum value in a reasonable amount of time which is at most polynomial time [10]. Algorithms that either give nearly the right answer or provide a solution not for all instances of the problem are called heuristic algorithms. This technique does not guarantee the best solution but various heuristics and approximation algorithms, which quickly yield good solutions, have been devised. Modern methods can find solutions for extremely large problems (millions of cities) within a reasonable time.

- Constructive heuristics
- Iterative improvement
- Randomized improvement

2.4 Branch and Bound Algorithm

Branch and bound (BB or B&B) is a general algorithm for finding optimal solutions of various optimization problems, especially in discrete and combinatorial optimization. The basic method will be to break up the set of all tours into smaller and smaller subsets and to calculate for each of them a lower bound on the cost of the best tour there in. The bounds guide the partitioning of the subsets and eventually identify an optimal tour when a subset is found that contains a single tour whose cost is less than or equal to the lower bounds for all subsets, that tour is optimal.

The subsets of tours are conveniently represented as the nodes of a tree and the process of partitioning as a branching of the tree. Hence we have called the method 'branch and bound' [7].

The two basic stages of a general Branch and Bound method:

- Branching :splitting the problem into sub problems
- Bounding :calculating lower and/or upper bounds for the objective function value of the sub problem

Several branching rules have been used in conjunction with the assignment problem relaxation of the TSP. In assessing the advantages and disadvantages of these rules one should keep in mind that the ultimate goal is to solve the TSP by solving as few sub-problems as possible.

Thus a good branching rule is one that (a) generates few successors of a node of the search tree, and (b) generates strongly constrained sub-problems, i.e. excludes many solutions from each sub-problem. Again, these criteria are usually conflicting and the merits of the various rules depend on the tradeoffs.

Consider the root problem to be the problem of finding the shortest route through a set of cities visiting each city once. Split the node into two child problems: the shortest route visiting city first and the shortest route *not* visiting city first. Continue subdividing similarly as the tree grows.

Let S be some subset of solutions.

Let LB(S) = a lower bound on the cost of any solution belonging to S.

Let z_0 = cost of the best solution found so far.

If z_0 <= LB(S), there is no need to explore S because it does not contain any better solution.

If z_0 > LB(S), thus we need to explore S because it may contain a better solution.

A Lower Bound for a TSP, the cost of any tour is

$$\frac{1}{2}\sum_{v\in V}(sum\ of\ the\ costs\ of\ the\ two\ tour\ edges\ adjacent\ to\ v) \quad (2.6)$$

The sum of the two tour edges adjacent to a given vertex v is less than equal to the sum of the two edges of least cost adjacent to v. Therefore, the cost of any tour

$$\frac{1}{2}\sum_{v\in V}(sum\ of\ the\ costs\ of\ the\ two\ least\ cost\ edges\ adjacent\ ot\ v) \quad (2.7)$$

2.4.1 Little's Branch and Bound Algorithm

The costs of travelling salesman problem form a matrix. Let the cities be indexed by i=1,2, …, n. The entry in row i and column j of the matrix is the cost for going from city i to city j. Let

$$C = [c(i,j)] = cost\ matrix.$$

C will start out as the original cost matrix of the problem but will undergo various transformations as the algorithm proceeds. A tour, ***t***, can be represented as a set of ***n*** ordered city pairs, e.g.,

$$t = [(i_1, i_2)(i_2, i_3) \dots (i_{n-1}, i_n)(i_n, i_1)] \quad (2.8)$$

which form a circuit going to each city once and only once. Each (i, j) represents an arc or leg of the trip. The cost of a tour, ***t***, under a matrix, ***C***, is the sum of the matrix elements picked out by ***t*** and will be denoted by ***z(t)***:

$$z(t) = \sum_{(i,j)in\ c} c(i,j). \quad (2.9)$$

Notice that that ***t*** always picks out one and only one cost in each row and in each column. Also, let

$X, Y, \bar{Y}$ = nodes of the tree;

$w(X)$ = a lower bound on the cost of the tours of X, i.e., $z(t) \geq w(X)$ for t a tour of X;

z_0 = the cost of the best tour found so far in the algorithm.

2.4.2 Lower Bounds

A useful concept in constructing lower bounds will be that of reduction. If a constant, h, is subtracted from each element of a row of the cost matrix, the cost of any tour under the new matrix is h less than under the old. This is because every tour must contain one and only one element from that row. The relative costs of all tours are unchanged, however, and so any tour optimal under the new.

The process of subtracting the smallest element of a row from each element in the row will be called reducing the row. A matrix with nonnegative elements and at least one zero in each row and column will be called a reduced matrix and may be obtained, for example, by reducing rows and columns. If z (t) the cost under the matrix afterward, and h the sum of constants used in making the reduction, then

$$z(t) = h + z_1(t). \tag{2.10}$$

Since a reduced matrix contains only nonnegative elements, $\boldsymbol{h}$ constitutes a lower bound on the cost of t under the old matrix.

2.4.3 Branching

The splitting the set of all tours into disjoint subsets will be represented by the branching of the tree, in Figure 2.1. The node containing 'all tours' is self-explanatory. The node containing i, j represents all tours which include the city pair (i, j). The node containing $\overline{i,j}$ represents all tours that do not. At the i, j node there is another branching. The node containing represents all tours that include (i, j) but not (k, l), whereas k, l represents all tours that include both (i, j) and (k, l). In general, by tracing from a node, X, back to the start, we can pick up which city pairs are committed to appear in the tours of X and which are forbidden from appearing. If the branching process is carried far enough, some node will eventually represent a single tour. Notice that at any

stage of the process, the union of the sets represented by the terminal nodes in the set of all tours.

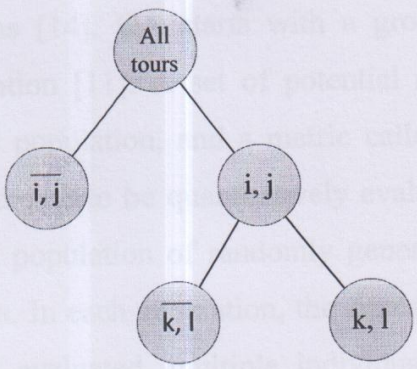


Figure 2.1 Start of the Tree

When a node X branches into two further nodes, the node with the newly committed city pair will frequently be called Y and the node with the newly forbidden city pair $\bar{Y}$.

2.5 Genetic Algorithm

Genetic Algorithm is being applied to a broad range of subjects, from abstract mathematical problems like bin-packing and graph coloring to a tangible engineering issue such as pipeline flow control, pattern recognition and classification, and structural optimization. Today, evolutionary computation is a thriving field, and genetic algorithms are in areas of study as diverse as stock market prediction, portfolio planning, aerospace engineering, microchip design, biochemistry and molecular biology, and scheduling at airports and assembly lines. Genetic Algorithm is applied to many scientific, engineering problems, in business and entertainment, including : Optimization, Automatic Programming, Machine and robot learning, Economic models, Immune system models, Ecological models, Interactions between evolution and learning and Models of social system.

Genetic algorithm (GA) as a computational intelligence method is a search technique to find approximate solutions to combinatorial optimization problems [14]. GA starts with a group of chromosomes known as the population [11]. A set of potential solutions is input to Genetic algorithm as population, and a metric called a fitness function that allows each candidate to be quantitatively evaluated. The evolution usually starts from a population of randomly generated individuals and happens in generation. In each generation, the fitness of every individual in the population is evaluated, multiple individuals are stochastically selected from the current population (based on their fitness), modified (recombined and possibly randomly mutated) to form a new population. The new population is then used in the next iteration of the algorithm. Commonly, the algorithm terminates when either a maximum number of generation has been produced, or a satisfactory fitness level has been reached for the population.

Genetic Algorithm begins with a set of k randomly generated states, called the population. Each state, or individual, is represented as a string over a finite alphabets-most commonly, a string of 0s and 1s [13]. For the representation of individuals from one population was chosen a chromosome, comprising several gens, where each gene contains the information of each city being visited [8].

City1	City2	City3	City4	……..	City n

Figure 2.2 Chromosome representation

The basic genetic algorithm is performed by the following algorithm.

1. **[Start]** Generate random population of *n* chromosomes (suitable solutions for the problem).
2. **[Fitness]** Evaluate the fitness *f(x)* of each chromosome *x* in the population.
3. **[New population]** Create a new population by repeating following steps until the new population is complete.
    (1). **[Selection]** Select two parent chromosomes from a population according to their fitness (the better fitness, the bigger chance to be selected) .
    (2). **[Crossover]** With a crossover probability cross over the parents to form a new offspring (children). If no crossover was performed, offspring is an exact copy of parents.
    (3). **[Mutation]** With a mutation probability mutate new offspring at each position in chromosome.
    (4). **[Accepting]** Place new offspring in a new population.
4. **[Replace]** Use new generated population for a further run of algorithm
5. **[Test]** If the end condition is satisfied, **stop**, and return the best solution in current population .Otherwise Go to **step 2**.

2.5.1 Encoding of Chromosomes

Encoding of chromosomes is the first step of problem solving with genetic algorithm. Encoding very depends on the problem. In this chapter will be introduced some encodings, which have been already used.

2.5.1.1 Binary Encoding

Binary encoding is the most common, mainly because first works about GA used this type of encoding [19]. In binary encoding every chromosome is a string of bits, 0 or 1.

2.5.1.3 Value Encoding

Direct value encoding can be used in problems, where some complicated values, such as real numbers, are used. Use of binary encoding for this type of problems would be very difficult. In value encoding, every chromosome is a string of some values. Values can be anything connected to problem, form numbers, real numbers or chars to some complicated objects.

Chromosome A	1.2324 5.3243 0.4556 2.3293 2.4545
Chromosome B	ABDJEIFJDHDIERJFDLDFLFEGT
Chromosome C	(back), (back), (right), (forward), (left)

Value encoding is very good for some special problems such as finding weights for neural network. On the other hand, for this encoding is often necessary to develop some new crossover and mutation specific for the problem.

In finding weights for neural network, There is some neural network with given architecture. Find weights for inputs of neurons to train the network for wanted output. Real values in chromosomes represent corresponding weights for inputs.

2.5.1.4 Tree Encoding

Tree encoding is used mainly for evolving programs or expressions, for genetic programming. In tree encoding every chromosome is a tree of some objects, such as functions or commands in programming language.

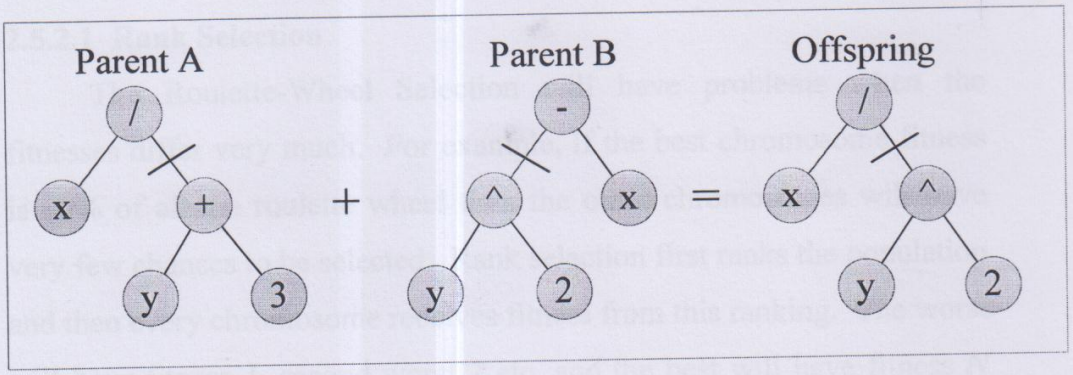


Figure 2.3 Tree Encoding

Tree encoding is good for evolving programs such as finding a function from given values. Programing language LISP is often used to this, because programs in it are represented in this form and can be easily parsed as a tree, so the crossover and mutation can be done relatively easily.

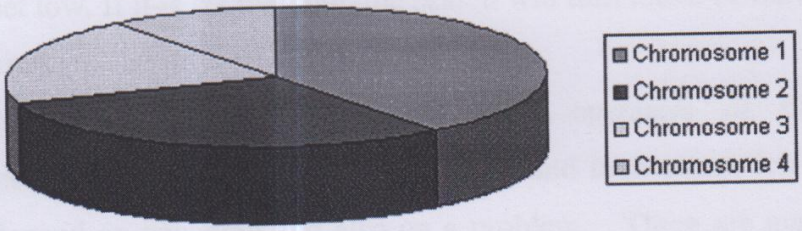
In finding a function from given values. Some input and output values are given. Task is to find a function, which will give the best (closest to wanted) output to all inputs. Chromosomes are functions represented in a tree.

2.5.2 Selection

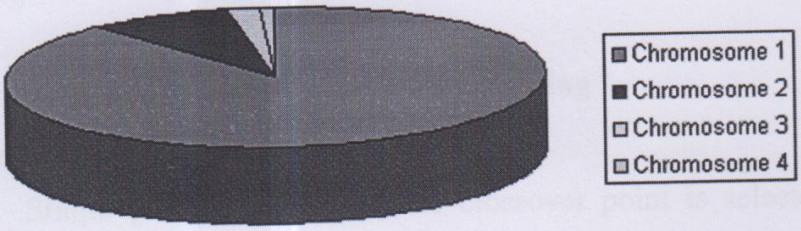
Individuals must be selected from the group to create a new population by crossover and mutation. By theory, good individuals should continue their life and create a new individual. Therefore, the probability of selection is important for the fitness value of the individual. There are many methods how to select the best chromosomes, for example roulette wheel selection, Boltzman selection, tournament selection, rank selection, steady state selection and some others. This chapter introduces only rank selection. Roulette-wheel and Tournament selection are described in section 3.4.3.

2.5.2.1 Rank Selection

The Roulette-Wheel Selection will have problems when the fitnesses differ very much. For example, if the best chromosome fitness is 90% of all the roulette wheel then the other chromosomes will have very few chances to be selected. Rank selection first ranks the population and then every chromosome receives fitness from this ranking. The worst will have fitness *1*, second worst *2* etc. and the best will have fitness *N* (number of chromosomes in population). In following picture, how the situation changes after changing fitness to order number.



Situation before ranking (graph of fitnesses)



Situation after ranking (graph of order numbers)

Figure 2.4 Rank Selection

2.5.3 Crossover and Mutation

Crossover is made in hope that new chromosomes will have good parts of old chromosomes and maybe the new chromosomes will be better. However it is good to leave some part of population survive to next generation. Once a pair of chromosomes has been selected, crossover can take place to produce offspring. Crossover basically

simulates sexual genetic recombination (as in human reproduction) and there are a number of ways that it is usually implemented in GAs.

Mutation is a genetic operator used to maintain genetic diversity from one generation of a population of chromosomes to the next. It is analogous to biological mutation. Mutation alters one or more gene values in a chromosome from its initial state. In mutation, the solution may change entirely from the previous solution. Hence GA can come to better solution by using mutation. Mutation occurs during evolution according to a user-definable mutation probability [15]. This probability should be set low. If it is set too high, the search will turn into a primitive random search.

Crossover and mutation are two main operators of GA. Performance of GA very depend on them. Type and implementation of operators depend on encoding and also on a problem. There are many ways how to do crossover and mutation. In this chapter, there are only some examples and suggestions how to do it for several encoding.

2.5.3.1 Crossover and Mutation on Binary Encoding

- **Crossover**
  - **Single point crossover** - one crossover point is selected, binary string from beginning of chromosome to the crossover point is copied from one parent, and the rest is copied from the second parent [18].

    11001011+11011**111** = **11001111**
  - **Two point crossover** - two crossover point are selected, binary string from beginning of chromosome to the first crossover point is copied from one parent, the part from the first to the second crossover point is copied from the second parent and the rest is copied from the first parent.

11001011 + 11**011111** = **11011111**

- **Uniform crossover** - bits are randomly copied from the first or from the second parent.

  11001011 + **11011101** = 11011111

- **Arithmetic crossover** - some arithmetic operation is performed to make a new offspring.

  11001011 + 11011111 = 11001001 (AND)

- **Mutation**

  Bit inversion - selected bits are inverted.

  11001001 => 10001001

2.5.3.2 Crossover and Mutation on Value Encoding

- **Crossover**

  All crossovers from binary encoding can be used.

- **Mutation**

  Adding a small number (for real value encoding) - to selected values is added (or subtracted) a small number [18].

  (1.29 5.68 **2.86 4.11** 5.55) => (1.29 5.68 **2.73 4.22** 5.55)

2.5.3.3 Crossover and Mutation on Tree Encoding

- **Crossover**

  Tree crossover - in both parent one crossover point is selected, parents are divided in that point and exchange part below crossover point to produce new offspring [18].

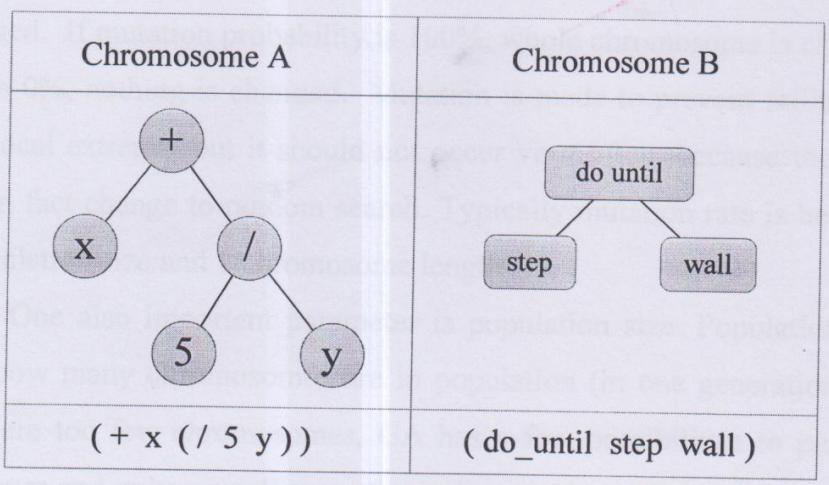


Figure 2.5 Tree Crossover

- **Mutation**

  Changing operator, number - selected nodes are changed.

2.5.4 Parameters of Genetic Algorithm

There are two basic parameters of GA - crossover probability and mutation probability.

Crossover rate says how often will be crossover performed. If there is no crossover, offspring is exact copy of parents. If there is a crossover, offspring is made from parts of parents' chromosome. If crossover probability is 100%, then all offspring is made by crossover. If it is 0%, whole new generation is made from exact copies of chromosomes from old population (but this does not mean that the new generation is the same!). Crossover rate is typically in the range (0.6, 0.9) [1].

Mutation rate (or ratio) is basically a measure of the likeness that random elements of your chromosome will be swapped into something else. If there is no mutation, offspring is taken after crossover (or copy) without any change. If mutation is performed, part of chromosome is

changed. If mutation probability is 100%, whole chromosome is changed, if it is 0%, nothing is changed. Mutation is made to prevent falling GA into local extreme, but it should not occur very often, because then GA will in fact change to random search. Typically mutation rate is between 1/population size and 1/chromosome length [1].

One also important parameter is population size. Population size says how many chromosomes are in population (in one generation). If there are too few chromosomes, GA has a few possibilities to perform crossover and only a small part of search space is explored. On the other hand, if there are too many chromosomes, GA slows down. Good population size is about 20-30. Sometimes sizes 50-100 are as best. Very big population size usually does not improve performance of GA.

2.5.5 Termination of Genetic Algorithm

The process of Genetic algorithm is repeated until a termination condition has been reached. Common terminating conditions are:

- A solution is found that satisfies minimum criteria
- Fixed number of generations reached
- Allocated budget (computation time/money) reached
- The highest ranking solution's fitness is reaching or has reached a plateau such that successive iterations no longer produce better results
- Manual inspection
- Combinations of the above

CHAPTER 3

SYSTEM DESIGN AND IMPLEMENTATION

3.1 System Design

The system applies two algorithms; Little's B&B algorithm and Genetic algorithm to find the optimal route for the salesman to travel in the given region. The workflow of Little's B&B algorithm is described in Section 3.1.2 and the workflow of Genetic algorithm is shown in Section 3.1.3.

3.1.1 System Flow Diagram

This system is designed to implement two algorithms, Little's B&B algorithm and genetic algorithm. Both algorithms run the same task to find optimal solution for the cities where a salesman wants to a tour.

When the user enters the system, he must firstly choose the regions as a tour to find the optimal route. And then the system constructs the cost matrix related to cities in the selected regions applying Euclidean distance formula. Using this cost matrix, Little's B&B algorithm and Genetic Algorithm find the cost of tour with optimal path. In the system, there is a comparison between two costs obtained from both algorithms. The computing times of two algorithms are also compared. Figure 3.1 shows the overview of the system.

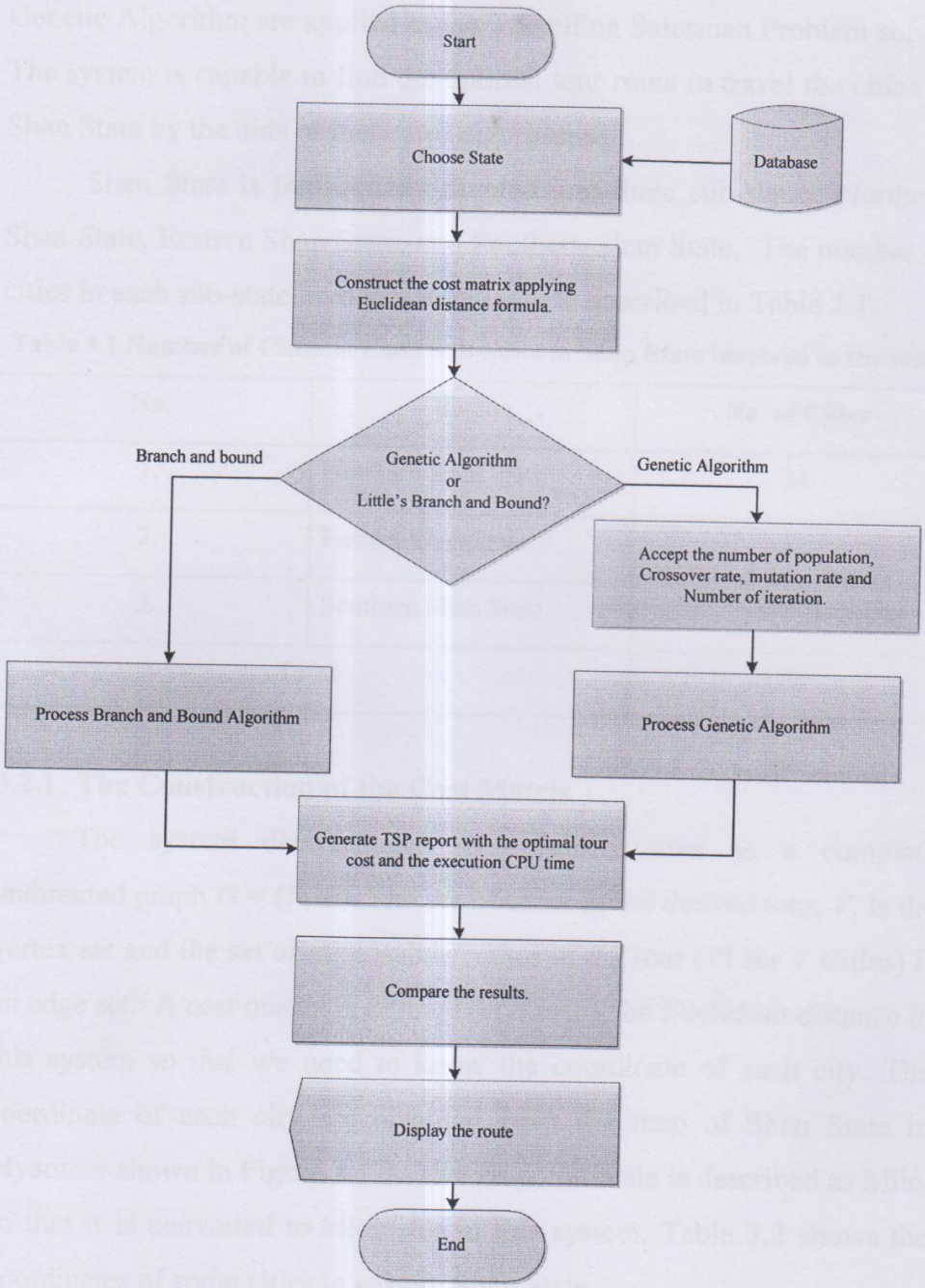


Figure 3.1 System Flow Diagram

3.2 Implementation of the System

The system intends the Travelling Salesman Problem for the cities of Shan State in Myanmar. Little's Branch and Bound algorithm and

Genetic Algorithm are applied as the Travelling Salesman Problem solver. The system is capable to find the optimal tour route to travel the cities in Shan State by the aids of these two algorithms.

Shan State is traditionally divided into three sub-states: Northern Shan State, Eastern Shan State, and Southern Shan State. The number of cities in each sub-state involved in the tour is described in Table 3.1.

Table 3.1 Number of Cities in Each Sub-State in Shan State involved in the tour

No.	Region	No. of Cities
1.	Northern Shan State	24
2.	Eastern Shan State	14
3.	Southern Shan State	25
Total		**63**

3.2.1 The Construction of the Cost Matrix

The system demonstrates the problem area as a complete undirected graph $G = (V, E)$. The set of cities in the desired tour, V, is the vertex set and the set of all possible routes in the tour ($V!$ for V Cities) is an edge set. A cost matrix is defined on E using the Euclidean distance in this system so that we need to know the coordinate of each city. The coordinate of each city are obtained from the map of Shan State in Myanmar shown in Figure A1. In that map, the scale is described as Mile, so that it is converted to kilometer in this system. Table 3.2 shows the coordinates of some cities in eastern Shan State.

The system constructs the cost matrix dynamically according to the coordinates of cities in the selected sub-state in Shan State using the Euclidean distance. The cost matrix obtained from Table 3.2 is shown in Figure 3.3. The illustrative examples of the proposed two algorithms applying this cost matrix are shown in Section 3.3.1 and 3.4.5.

Table 3.2 The Coordinates of six cities in Eastern Shan State

No.	City	X (Km)	Y (Km)	Encoding No.
1	Mong Ton	29.29	64.37	1
2	Mong Hsat	352.45	94.95	2
3	Ta Chi Leik	378.2	88.51	3
4	Mong Hpyak	402.34	135.18	4
5	Kyaing Tung	370.15	177.03	5
6	Mong Ping	318.65	183.47	6

		To					
		1	2	3	4	5	6
From	1	∞	66.94	88.65	130.35	136.6	121.85
	2	66.94	∞	26.54	64.09	83.97	94.75
	3	88.65	26.54	∞	52.54	88.89	112.09
	4	130.35	64.09	52.54	∞	52.8	96.62
	5	136.6	83.97	88.89	52.8	∞	51.9
	6	121.85	94.75	112.09	96.62	51.9	∞

Figure 3.2 Cost Matrix for six cities in Eastern Shan State

3.3 The Workflow of Little's Branch and Bound Algorithm

The working of the algorithm will be explained by tracing through the flow chart of Figure 3.3.

Step 1 starts the calculation by putting the original cost matrix of the problem into C, setting X= 1 to represent the node, 'all tours', and setting the cost of the best tour so far to infinity.

Step 2 reduces the matrix and labels node X with its lower bound $w(X)$.

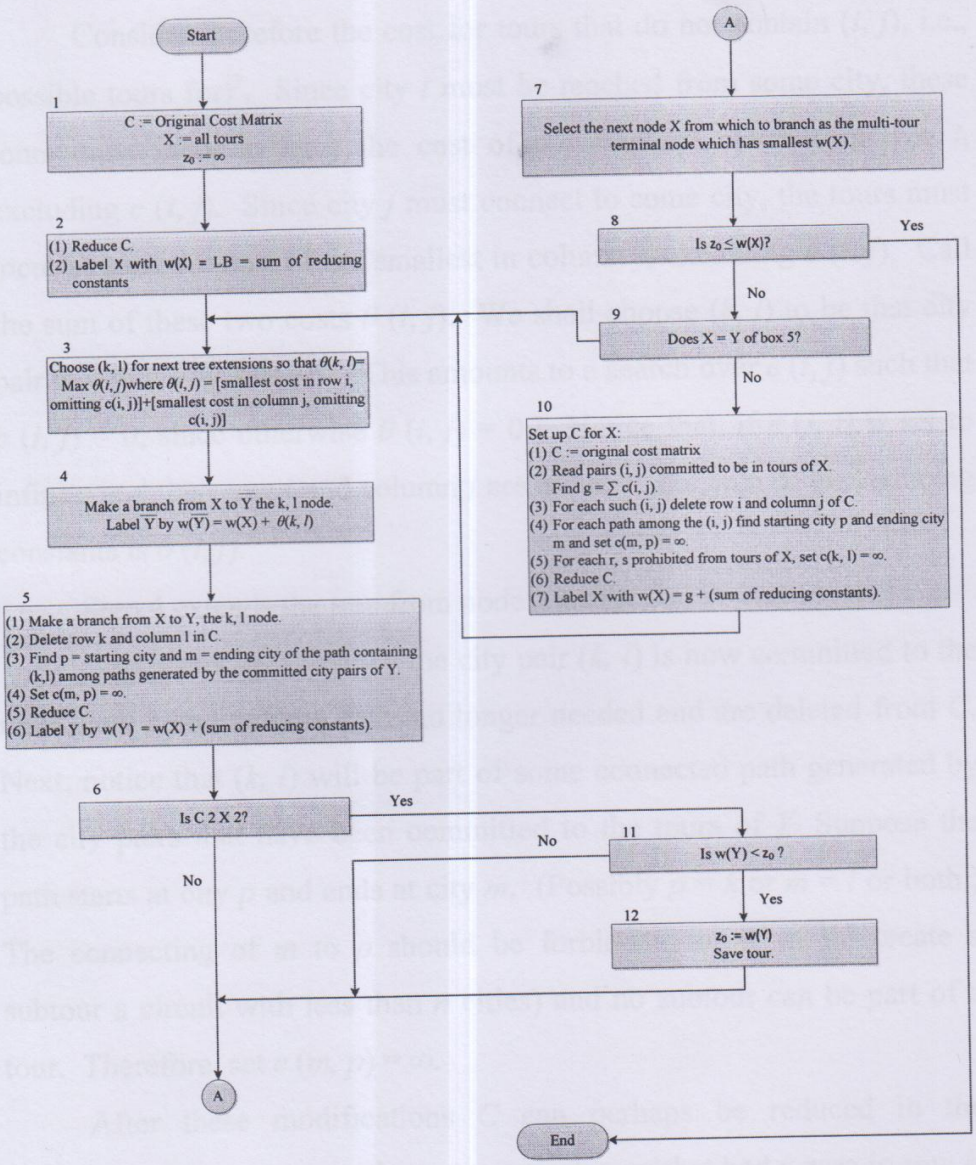


Figure 3.3 Flow Chart of Little's Branch and Bound Algorithm

Step 3 selects (k, l), the city pair on which to base the next branching. The goal in doing this is to split the tours of x into subset (Y) that is quite likely to include the best tour of the node and another (Y) that is quite unlikely to include it. Possible low cost tours to consider for Y are those involving an (i, j) for which $c(i, j) = 0$.

Consider therefore the cost for tours that do not contain (i, j), i.e., possible tours for$\bar{Y}$. Since city i must be reached from some city, these tours must incur at least the cost of the smallest element in row i, excluding $c\ (i, j)$. Since city j must connect to some city, the tours must incur at least the cost of the smallest in column j, excluding $c\ (i, j)$. Call the sum of these two costs $\theta\ (i, j)$. We shall choose (k, l) to be that city pair that gives the largest. [This amounts to a search over $c\ (i, j)$ such that $c\ (i, j) = 0$, since otherwise $\theta\ (i, j) = 0$.] Notice that, if $c\ (i, j)$ is set to infinity and then row i and column j are reduced, the sum of the reducing constants is $\theta\ (i, j)$.

Step 4 extends the tree from node X to $\bar{Y}$.

Step 5 sets up Y. Since the city pair (k, l) is now committed to the tours, row k and column l are no longer needed and are deleted from C. Next, notice that (k, l) will be part of some connected path generated by the city pairs that have been committed to the tours of Y. Suppose the path starts at city p and ends at city m. (Possibly $p = k$ or $m = l$ or both.) The connecting of m to p should be forbidden for it would create a subtour a circuit with less than n cities) and no subtour can be part of a tour. Therefore, set $c\ (m, p) = \infty$.

After these modifications C can perhaps be reduced in the following places: row m, column p, any columns that had a zero in row k, and any rows that had a zero in column l. All other rows and columns contain some zero that cannot have been disturbed. Let h be the sum of the new reducing constants. The lower bound for Y will now be shown to be

$$w(Y) = w(X) + h \tag{3.1}$$

The algorithm operates so that the investigation of each node, X, starts in Step 3 with a matrix C and a lower bound w(X) that stand in a special relation. If t is any tour of X, $z(t)$ its cost under the original matrix,

t_1, the city pairs of t left removing those committed to the tours of X, and $z_1(t_1)$ the cost of t_1 under C, then it will be shown that

$$z(t) = w(X) + z_1(t_1). \tag{3.2}$$

This expression is true for the first node by Equation (3.1). Suppose that from a bound $w(X_1)$ and matrix C_1 of a node X, the algorithm constructs a bound $w(X_2)$ and reduced matrix C_2 for a node X_2. (X_2 will be on some branch out of X_1.) If will be shown that, if Equation (3.2) is true for X_1, Equation (3.2) will also be true for X_2.

The operations on C_1 to get C_2 (shown in Steps 5 and 10) are always of the form: delete row i and column j for each (i, j) committed to the tours of X_2, insert various infinites, reduce. The lower bound is always of the form

$$w(X_2) = w(X_1) + \sum c_1(i,j) + h, \tag{3.3}$$

where the summation is over the city pairs committed in X_2, but not in X_1 and h is the sum of the reducing constants. But consider any t in X_2 (and therefore in X_1). If we let $z_1(t_1)$ be the cost of the uncommitted city pairs of C_1 under X_1 and $z_2(t_2)$ be the cost of the uncommitted city pairs of X_1 under C_2,

$$z_1(t_1) = \sum c_1(i,j) + h + z_2(t_2), \tag{3.4}$$

or using (3.2), assumed true for X_1,

$$z(t) = w(X_1) + \sum c_1(i,j) + h + z_2(t_2) = w(X_2) + z_2(t_2), \tag{3.5}$$

so that (3.2) is true for X_2, as was to be shown.

Equation (3) is used to calculate the lower bounds in Steps 4, 5 and 10. That these lower bounds are valid is established by Equation (3.2) and the nonnegative of the elements of C.

Step 6 checks to see whether a single tour node is near.

Step 7 selects the next node for branching. There are a number of ways choice might be made. The way shown here is to pick the node will the smallest lower bound. This leads to the fewest nodes in the tree.

Step 8 checks to see whether the algorithm is finished whether the best tour so far has a cost less than or equal to the lower bounds on all terminal nodes of the tree.

Step 9 is a time saver. Most branching is from Y nodes, i.e., to the right. Such branching involves crossing out rows and columns and other manipulations that can be done on the matrix left over from the previous branching. When this case occurs, Step 9 detects it and the algorithm returns directly to Step 3.

Step 10 takes up the alternate case of setting up an appropriate lower bound and reduced matrix for an arbitrary X. Starting from the original cost matrix, rows and columns are deleted for city pairs committed to the tours of X, infinites are placed to block subtours and at forbidden city pairs, and the resulting matrix is reduced. The lower bound can be computed from Equation (3.3) by thinking of X_1 in Equation(3.3) as a starting node with $w(X_1) = 0$ and matrix equal the original cost matrix. Since different ways of reducing a matrix may lead to different sums for the reducing constants, the recalculated $w(X)$ is substituted for the former one.

Step 11 and 12 finish up a single tour node. By the time C is a 2×2 matrix, there are only two feasible (i, j) left and they complete a tour. Since the step is entered with a reduced matrix, the costs of the final commitments are zero, and $z = w(Y)$ by Equation (3.2). If $z < z_0$, the new tour is the best yet and is read off the tree to be saved.

3.3.1 Illustrative Example for Little's Branch and Bound

The example is calculated from Table 3.2 and cost matrix from Figure 3.2.

Step 1: C: = original cost matrix, z0:= ∞

Step 2: Reducing C.

	1	2	3	4	5	6	row min
1	∞	66.9	88.7	130	137	122	66.94
2	66.9	∞	26.5	64.1	84	94.8	26.54
3	88.7	26.5	∞	52.5	88.9	112	26.54
4	130	64.1	52.5	∞	52.8	96.6	52.54
5	137	84	88.9	52.8	∞	51.9	51.9
6	122	94.8	112	96.6	51.9	∞	51.9
sum							276.36

(a)

Row reduced cost matrix:

	1	2	3	4	5	6
1	∞	0	21.7	63.4	69.7	54.9
2	40.4	∞	0	37.6	57.4	68.2
3	62.1	0	∞	26	62.4	85.6
4	77.8	11.6	0	∞	0.26	44.1
5	84.7	32.1	37	0.9	∞	0
6	70	42.9	60.2	44.7	0	∞

(b)

	1	2	3	4	5	6	Sum
1	∞	0	21.7	63.4	69.7	54.9	
2	40.4	∞	0	37.6	57.4	68.2	
3	62.1	0	∞	26	62.4	85.6	
4	77.8	11.6	0	∞	0.26	44.1	
5	84.7	32.1	37	0.9	∞	0	
6	70	42.9	60.2	44.7	0	∞	
col min	40.4	0	0	0.9	0	0	41.3

(c)

Column reduced cost matrix:

	1	2	3	4	5	6
1	∞	0	21.7	62.5	69.7	54.9
2	0	∞	0	36.7	57.4	68.2
3	21.7	0	∞	25.1	62.4	85.6
4	37.4	11.6	0	∞	0.26	44.1
5	44.3	32.1	37	0	∞	0
6	29.6	42.9	60.2	43.8	0	∞

(d)

Figure 3.4 Reducing C (a) row minimum, (b) row reduced matrix, (c) column minimum and (d) column reduced matrix

$$Lower\ bound = \sum row\ \min + \sum column\ min \qquad (3.6)$$

The lower bound on all tours of this TSP problem is

L.B. = 276.36+41.3=317.66.

So z_0 is greater than or equal 317.66 for all tours and the starting (root) node of the tree is accordingly marked with 317.66.

Step 3: The $\theta\,(i, j)$ values are written in the right upper corner of the element in the cells of the zeroes of Figure 3.5(b). The largest θ is θ **(5, 6)** = 0 + 44.08 = 44.08 and so (5, 6) will be the first city pair used for branching.

	1	2	3	4	5	6
1	∞	$0^{(21.7+0)}$	21.71	62.51	69.66	54.91
2	$0^{(0+21.71)}$	∞	$0^{(0+0)}$	36.65	57.43	68.21
3	21.71	$0^{(21.71+0)}$	∞	25.1	62.35	85.55
4	37.41	11.55	$0^{(0.26+0)}$	∞	0.26	44.08
5	44.3	32.07	36.99	$0^{(0+25.1)}$	∞	$0^{(0+44.08)}$
6	29.55	42.85	60.19	43.82	$0^{(0.26+29.55)}$	∞

(a)

	1	2	3	4	5	6
1	∞	$0^{(21.71)}$	21.71	62.51	69.66	54.91
2	$0^{(21.71)}$	∞	$0^{(0)}$	36.65	57.43	68.21
3	21.71	$0^{(21.71)}$	∞	25.1	62.35	85.55
4	37.41	11.55	$0^{(0.26)}$	∞	0.26	44.08
5	44.3	32.07	36.99	$0^{(25.1)}$	∞	$0^{(44.08)}$
6	29.55	42.85	60.19	43.82	$0^{(29.81)}$	∞

(b)

Figure 3.5 (a) Finding $\theta(i, j)$ in each element of the zeroes
(b) Cost Matrix with the values of $\theta(i, j)$

Step 4: The system makes a branch from root node to $\bar{Y} = \overline{5,6}$ node as the first branch. So, the system will show below, $w(\bar{Y}) = w(X) + \theta(k, j)$. In the example, $w(\bar{Y}) = 317.66 + 44.08 = 361.74$ and the node is so labeled in Figure (3.6).

Step 5: (1) The system makes a branch from root node to $Y = (5, 6)$ node.

(2) If there is a path from 5 to 6 i.e. 5, 6 = 1, the system *eliminates the respective row 5 and column 6.*

(3) The connected path containing (5, 6) is (5, 6) itself, so $(m, p) = (6, 5)$ and the system must be set $c\ (6, 5) = \infty$.

	1	2	3	4	5
1	∞	*0*	*21.71*	*62.51*	*69.66*
2	0	∞	0	36.65	57.43
3	21.71	0	∞	25.1	62.35
4	37.41	11.55	0	∞	*0.26*
6	29.55	42.85	60.19	43.82	∞

Figure 3.6 Matrix after deletion of row 5 and column 6

(4) Reducing C that is the cost matrix from Figure 3.7 is like Figure 3.4.

(5) $w(Y) = \text{LB} = 317.66 + 29.55 + 14.53 = 361.74$

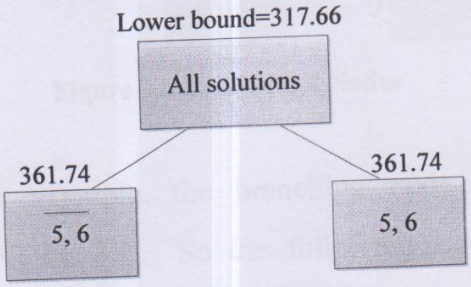


Figure 3.7 Start of Tree with Two Nodes

Step 6: The system checks whether C is a 2 × 2 matrix. Now the matrix is not. So, the system goes to continue to Step 7, and otherwise goes to Step 11.

Step 7: In example, the node (5, 6) as branching node *X*, is chosen for branching because of being right node although the lower bounds of the node $\overline{(5,6)}$ *and* (5,6) are the same.

Step 8: The system checks whether z_0 is less than or equal to $w(X)$ = 361.74. Now z_0 is ∞ so the system goes to Step 9.If not so, system finishes.

Step 9: The selected node 5, 6 is k+2. So, the system goes to Step 3 through Step 6. The calculation is like above until the tree grows in below Figure 3.8. After checking the matrix C is 2 × 2 or not, goes to Step 7. The next node for branching is $(\overline{5,6})$ node with the smallest lower bound seen in Figure 3.8.It is the $(\bar{Y})$ node, so the system calculates the Step 10.

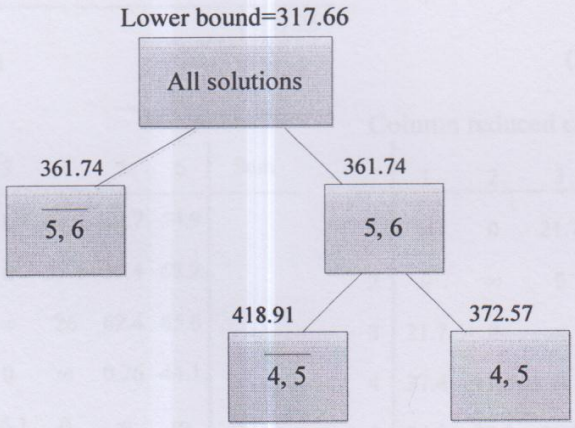


Figure 3.8 Tree with 4 Nodes

Step 10: In this example, the branching node is $\bar{Y}$, $(\overline{5,6})$ node according to the Figure 3.8. So the following steps are needed for making.

(1) Put the original cost matrix to C.

(2) There is no committed to be in tours of $\overline{5,\ 6}$ node, so $g = 0$.

(3) No need to take row and column deletion of C because of no committed tours. Therefore, there is no starting city and ending city.

There is one prohibited tour (5 to 6) from $\overline{5,\ 6}$ node. We set c (5, 6) = ∞.

	1	2	3	4	5	6	row min
1	∞	66.9	88.7	130	137	122	66.94
2	66.9	∞	26.5	64.1	84	94.8	26.54
3	88.7	26.5	∞	52.5	88.9	112	26.54
4	130	64.1	52.5	∞	52.8	96.6	52.54
5	137	84	88.9	52.8	∞	∞	52.8
6	122	94.8	112	96.6	51.9	∞	51.9
sum							277.26

(a)

Row reduced cost matrix:

	1	2	3	4	5	6
1	∞	0	21.7	63.4	69.7	54.9
2	40.4	∞	0	37.6	57.4	68.2
3	62.1	0	∞	26	62.4	85.6
4	77.8	11.6	0	∞	0.26	44.1
5	83.4	31.2	36.1	0	∞	∞
6	70	42.9	60.2	44.7	0	∞

(b)

	1	2	3	4	5	6	Sum
1	∞	0	21.7	63.4	69.7	54.9	
2	40.4	∞	0	37.6	57.4	68.2	
3	62.1	0	∞	26	62.4	85.6	
4	77.8	11.6	0	∞	0.26	44.1	
5	83.4	32.2	36.1	0	∞	∞	
6	70	42.9	60.2	44.7	0	∞	
col min	40.4	0	0	0	0	44.1	84.48

(c)

Column reduced cost matrix:

	1	2	3	4	5	6
1	∞	0	21.7	64.4	69.7	10.8
2	0	∞	0	37.6	57.4	24.1
3	21.7	0	∞	26	62.4	41.5
4	37.4	11.6	0	∞	0.26	0
5	44.3	32.2	36.1	0	∞	∞
6	29.6	42.9	60.2	44.7	0	∞

(d)

Figure 3.9 Reduced Matrix

(4) Reduce *C that* is shown in Figure 3.9.

(5) $w(X) = g + sum\ of\ reducing\ constants = 0 + 361.74 = 361.74$

The recalculated $w(X)$ is substituted for the former one but they are the same. And then calculation processes go on step 3 to step 6. When the cost matrix is 2 × 2 matrix, the system checks Step 11, and otherwise goes to Step 7. The tree grows like in Figure 3.10.

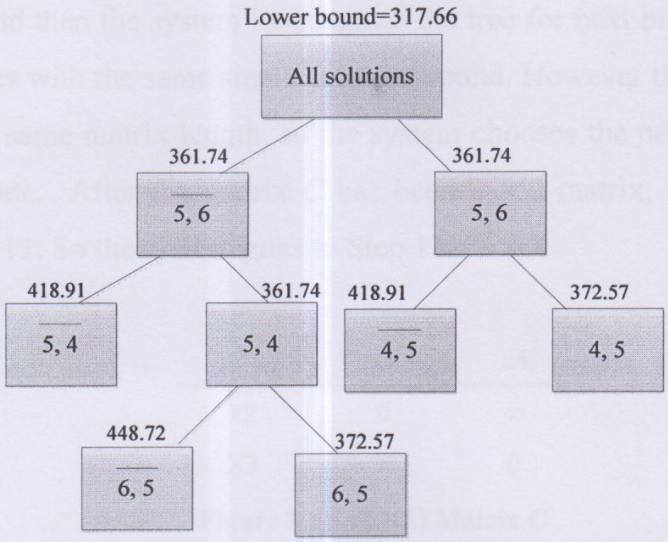


Figure 3.10 Tree with Eight Nodes

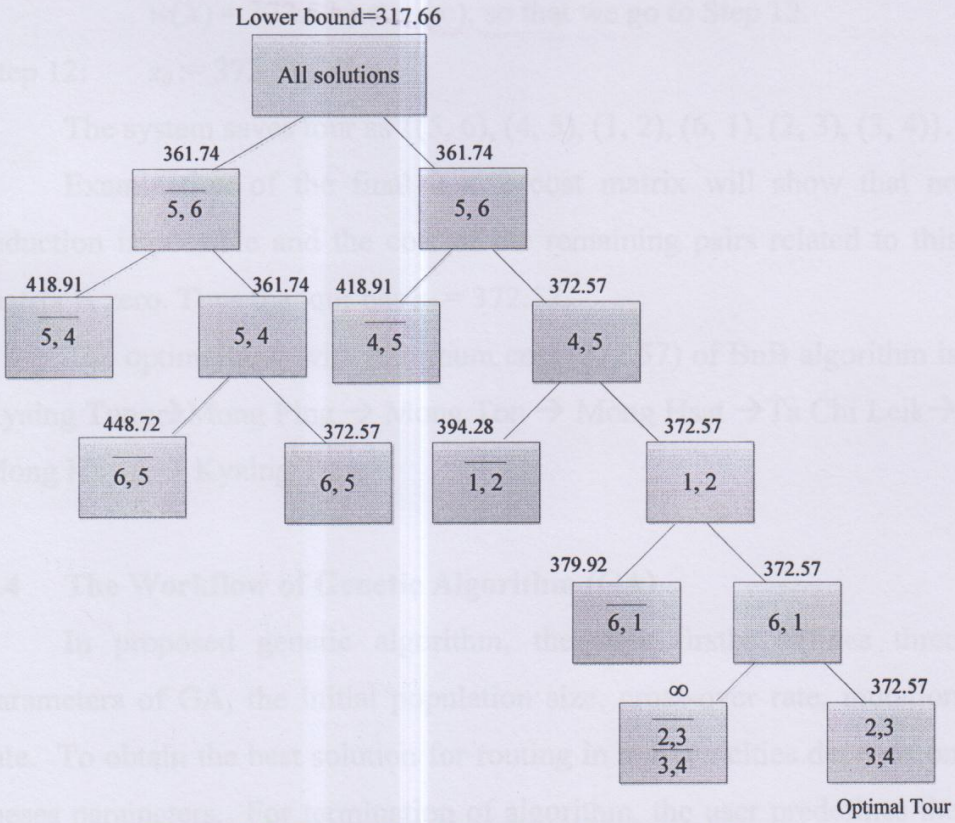


Figure 3.11 Tree with Optimal Solution at Node 14

And then the system investigates the tree for next branch, there are two nodes with the same smallest lower bound. However these tow nodes have the same matrix length, so the system chooses the next node as $Y =$ (4, 5) node. After the matrix C has been 2 × 2 matrix, the tree is like Figure 3.11. So the system goes to Step 11.

	x3	x4
x2	0	∞
X3	∞	0

Figure 3.12 (2 × 2) Matrix *C*

Step 11: If $w(X) = (6, 1) < z_0$, continue Step 12, otherwise Step 7.

$w(X) = 372.57) < z_0 (\infty)$, so that we go to Step 12.

Step 12: $z_0 := 372.57$

The system saves tour as {(5, 6), (4, 5), (1, 2), (6, 1), (2, 3), (3, 4)}.

Examination of the final 2 x 2 cost matrix will show that no reduction is possible and the cost of the remaining pairs related to this matrix is zero. Thus the tour has $z_0 = 372.57$.

The optimal tour with minimum cost (372.57) of BnB algorithm is Kyaing Tung→Mong Ping → Mong Ton → Mong Hsat →Ta Chi Leik→ Mong Hpyak→ Kyaing Tung.

3.4 The Workflow of Genetic Algorithm (GA)

In proposed genetic algorithm, the user firstly defines three parameters of GA, the initial population size, cross-over rate, mutation rate. To obtain the best solution for routing in a given cities depends on theses parameters. For termination of algorithm, the user predefines the number of iteration. In TSP, crossover rate and mutation rate are two

main operators in GA. The work flow of proposed Genetic algorithm is shown in Figure 3.3.

Firstly, the system generates randomly initial population with the predefined population size and then calculates fitness for each individual from population. The system picks up two chromosomes as parents in the population using Tournament or Roulette Selection and combines them to make two children using single point crossover according to the crossover rate. The child tours are mutated according to the mutation rate. The new child tours are saved into the new population list. If the new population size is less than (population size - 2), the system goes to the selection process. Otherwise, if the new population size is equal to (population size - 2), the system picks the two chromosomes with the minimum cost up from old population and saves in the new population list. The size of the new population remains the same. The old population is replaced with the new population.

The system then checks whether the termination criteria reaches. When the optimal cost from the new population is compared the optimal cost of the Branch and Bound, the system terminates the algorithm and returns the best tour with the optimal cost if the cost of GA is less than Branch and Bound's. Else if the system repeats the algorithm until the desired number of iteration is reached.

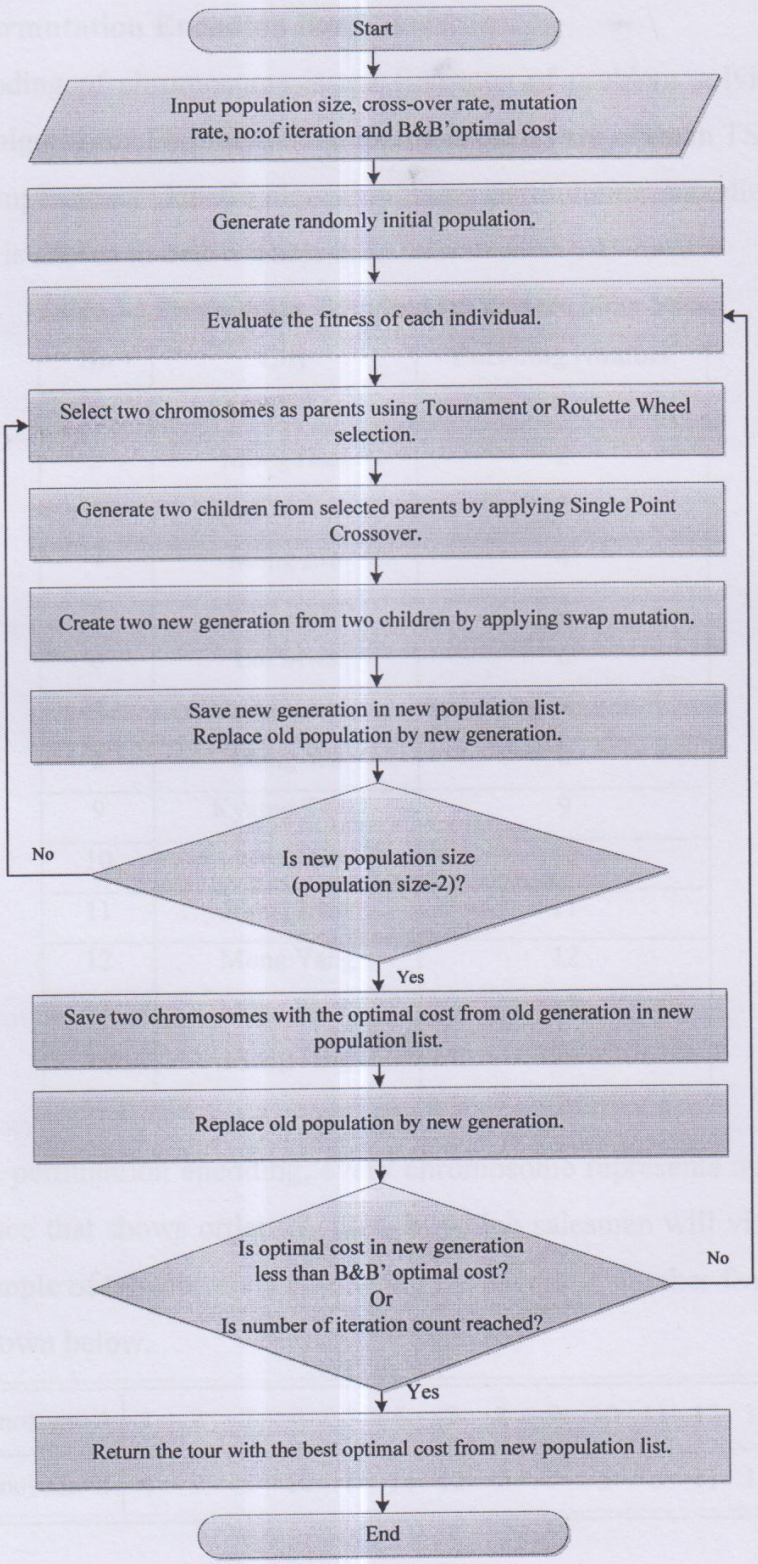


Figure 3.13 Flow Diagram of Genetic Algorithm

3.4.1 Permutation Encoding For GA

Coding of chromosome is the first step of problem solving with Genetic algorithm. Permutation encoding is used very often in TSP. This system implements Genetic algorithm using permutation encoding. The example is shown in below Table 3.3.

Table 3.3 Permutation Encoding for Eastern Shan State

No	City	Encoding Number
1	Mong Ton	1
2	Mong Hsat	2
3	Ta Chi Leik	3
4	Mong Lin	4
5	Mong Hpyak	5
6	Loi Mwe	6
7	Mong Yawng	7
8	Mong Wa	8
9	Kyaing Tung	9
10	Mong La	10
11	Mong Hkat	11
12	Mong Yang	12
13	Mong Pawk	13
14	Mong Ping	14

In permutation encoding, every chromosome represents number in a sequence that shows order of cities, in which salesman will visit them. The example of two chromosomes using the encoding number from Table 3.3 is shown below.

Chromosome A	1	2	3	4	5	6	7	8	9	10	11	12	13	14
Chromosome B	4	9	5	10	1	14	12	2	7	3	6	11	13	8

Figure 3.14 Example of Chromosomes

3.4.2 Fitness Function

The GA is used for maximization problem. For the maximization problem the fitness function is same as the objective function. But, for minimization problem, one way of defining a 'fitness function' is as $F(x) = \frac{1}{f(x)}$, where $f(x)$ is the objective function. Since, TSP is a minimization problem; this fitness function is used where $f(x)$ calculates cost (or value) of the tour represented by a chromosome.

3.4.3 Selection

In this system the Tournament and Roulette-Wheel selection are implemented.

3.4.3.1 Tournament Selection

Tournament selection is a method of selecting an individual from a population of individuals in a genetic algorithm. k individuals are chosen randomly from the population and the ones with the best fitness are picked [6]. Selection pressure is easily adjusted by changing the tournament size. If the tournament size is larger, weak individuals have a smaller chance to be selected.

3.4.3.2 Roulette-Wheel Selection

Parents are selected according to their fitness. The better the chromosomes are, the more chances to be selected they have. The system imagines a roulette wheel where are placed all chromosomes in the population, every chromosome has its place big accordingly to its fitness function, like on the following picture. Chromosome with bigger fitness will be selected more times. This can be simulated by following algorithm.

[Sum] : S = sum of all chromosome fitnesses in population

[Select] : r = random number from interval *(0, S)*

[Loop] : s = Go through the population and sum fitnesses from *0*

When the sum *s* is greater than *r*, stop and return the chromosome where you are.

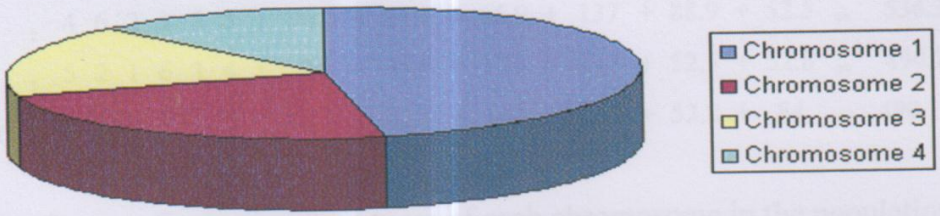


Figure 3.15 Roulette-Wheel Selection

3.4.4 Cross over and Mutation on Permutation Encoding

Crossover

Single point crossover - one crossover point is selected, till this point of permutation is copied from the first parent, then the second parent is scanned and if the number is not yet in the offspring, it is added.

Note: there are more ways how to produce the rest after crossover point.

(**1 2 3 4 5 6** 7 8 9 10 11 12 12 14) + (**4 9 5 10 1 14** 12 2 7 3 6 11 13 8)

= (**1 2 3 4 5 6** 9 10 14 12 7 111 13 8)

Mutation

Order changing - two numbers are selected and exchanged

(1 2 3 4 5 6 9 **10** 14 12 7 11 13 **8**) => (1 2 3 4 5 6 9 **8** 14 12 7 111 13 **10**)

3.4.5 Illustrative Example for Genetic Algorithm

The example of Genetic Algorithm is calculated from Table 3.2 and cost matrix from Figure 3.2.

Step 1: Generate 10 initial individual populations randomly.

T1	:	1	3	5	2	6	4	:	88.7	+	88.9	+	84	+	94.8	+	96.6	+	130	=	**583.23**
T2	:	3	4	1	6	5	2	:	52.5	+	130	+	122	+	51.9	+	84	+	26.5	=	**467.15**
T3	:	2	4	1	5	6	3	:	64.1	+	130	+	137	+	51.9	+	113	+	26.5	=	**522.33**
T4	:	3	6	2	1	4	5	:	112	+	94.8	+	66.9	+	130	+	52.8	+	88.9	=	**545.77**
T5	:	5	1	4	3	6	2	:	137	+	130	+	52.5	+	112	+	94.8	+	84	=	**610.3**
T6	:	2	3	1	5	6	4	:	26.5	+	88.7	+	137	+	51.9	+	96.6	+	64.1	=	**464.4**
T7	:	1	2	3	4	5	6	:	66.9	+	26.5	+	52.5	+	52.8	+	51.9	+	122	=	**372.57**
T8	:	4	6	2	1	5	3	:	96.6	+	94.8	+	66.9	+	137	+	88.9	+	52.5	=	**536.3**
T9	:	5	2	1	6	3	4	:	84	+	66.9	+	122	+	112	+	52.5	+	52.8	=	**490.2**
T10	:	2	1	6	3	4	5	:	66.9	+	122	+	113	+	52.5	+	52.8	+	84	=	**490.11**

Step 2: Evaluation the fitness of each chromosome in the population

F(T1)	:	1/583.23	=	0.0017
F(T2)	:	1/467.15	=	0.0021
F(T3)	:	1/522.33	=	0.0019
F(T4)	:	1/545.77	=	0.0018
F(T5)	:	1/610.3	=	0.0016
F(T6)	:	1/464.3	=	0.0021
F(T7)	:	1/372.57	=	0.0026
F(T8)	:	1/536.3	=	0.0018
F(T9)	:	1/490.2	=	0.002
F(T10)	:	1/490.11	=	0.002

Step 3: Creating new population

3-1 Selection

- Selection using Roulette Wheel

S = 0.0017+ 0.0021+ 0.0019 + 0.0018 + 0.0016 + 0.0021 + 0.0026 +

0.0018 + 0.0020 + 0.0020 = 0.0196

Choose random number (r) between 0 and 0.0916.

r = 0.009,	chromosomes -1	0.0017
	chromosomes -2	0.0021
		0.0038
	chromosomes -3	0.0019
		0.0057
	chromosomes -4	0.0018
		0.0075
	chromosomes -5	0.0016
		0.091 > 0.009

So the system chooses chromosomes-4 as parent. And then choose another random number (r) for another parent. The system finds two parents from random-1 and random-2:

r = 0.0065	chromosomes -1	0.0017
	chromosomes -2	0.0021
		0.0038
	chromosomes -3	0.0019
		0.0057
	chromosomes -4	0.0018
		0.0075 > 0.0065

So the system chooses chromosomes-3 as parent. After choosing two parents, the system creates new two offspring.

- Selection using Tournament Selection

1. Select 5 individuals (T1, T3, T5, T7 and T9) from populations randomly.
2. T7 with the best fitness is picked up as parent 1.

   Another parent 2 is also selected like the above.

3-2 Crossover

*

Parent 1	**5**	**1**	**4**	**3**	**6**	**2**
Parent 2	**3**	**6**	**2**	**1**	**4**	**5**

5	1	4	3	6	2	**c1**
3	6	2	5	1	4	**c2**

Figure 3.16 Example of Crossover

3-3 Mutation

c1

Before	5	1	4	3	6	2
After	5	6	4	3	1	2

c2

Before	3	6	2	5	1	4
After	3	1	2	5	6	4

Figure 3.17 Example of Mutation

3-4 After mutation, these two offspring are placed in a new population. The system loops step 3 until new population size is equal to old population size.

Step 4: Use new population for next iteration.

Step 5: When the defined number of loop count reaches, stop, and return the best solution in the current population. Otherwise, the system goes to step 2. In this example, when number of loop count reaches 3 times, the new population is below.

T1 : 5 2 4 1 6 3 : 84 + 64.1 + 130 + 122 + 112 + 88.9 = 601.24

T2 : 5 3 1 6 4 2 : 88.9 + 88.7 + 122 + 96.6 + 64.1 + 84 = 544.07

T3 : 3 6 1 5 2 4 : 112 + 122 + 137 + 84 + 64.1 + 52.5 = 571.14

T4 : 2 4 1 6 3 5 : 64.1 + 130 + 122 + 112 + 88.9 + 84 = 601.24

T5 : 1 5 3 4 6 2 : 137 + 88.9 + 52.5 + 96.6 + 94.8 + 66.9 = 536.34

T6 : 1 3 5 6 4 2 : 88.7 + 88.9 + 51.9 + 96.6 + 64.1 + 66.9 = 457.09

T7 : 2 3 6 1 5 4 : 26.5 + 112 + 122 + 137 + 52.8 + 64.1 = 505.93

T8 : 1 6 3 5 2 4 : 122 + 112 + 88.9 + 84 + 64.1 + 130 = 601.24

T9 : 4 3 2 1 6 5 : 52.5 + 26.5 + 66.9 + 122 + 51.9 + 52.8 = 372.57

T10 : 1 2 3 4 5 6 : 66.9 + 26.5 + 52.5 + 52.8 + 51.9 + 122 = **372.57**

At the third iteration the best tour is cost 372.57. If it continues other next iterations, it can get the best tour with the better minimum cost than 372.57.

The optimal tour with minimum cost (372.57) of GA algorithm is Mong Ton → Mong Hsat → Ta Chi Leik → Mong Hpyak→ Kyaing Tung→Mong Ping → Monsg Ton.

CHAPTER 4
EXPERIMENTS OF THE SYSTEM

In this chapter the experimental results of algorithms, Genetic Algorithm and Little's Brand and Bound are described. The experimentation of this system is carried out by using Intel Core i3 Processor and 4 GB of main memory.

To measure the performance of both Little's B&B and GA, the system uses second for CPU time and kilometer the unit of tour cost.

4.1 Experiments with Little's Branch and Bound Algorithm

If the branching of tree is always from the right node, the execution time is less. Otherwise, if the branching is from the left node, the execution time is more because the examining nodes are more. The more the number of the cities, the longer the execution time is. Table 4.1 shows the optimal tour's costs with their CPU execution time of Little's B&B algorithm.

Table 4.1 CPU Time and Tour Cost for Little's B&B

No.	Sub-State	No of Cities	Little's B&B	
			Time in seconds	Cost (Km)
1	Eastern	14	0.53	671.90
2	Northern	24	3.432	1238.79
3	Southern	25	3.837	1106.77
4	Northern & Eastern	38	9.641	2058.54
5	Southern & Eastern	39	13.088	1940.18
6	Southern & Northern	49	27.799	2265.29
7	The whole Shan State	63	58.459	3254.45

4.2 Experiments with Genetic Algorithm

According to the experiment results, the GA gets better results using the tournament selection rather than using Roulette-Wheel selection. So only the results of GA applying the tournament selection are discussed.

Table 4.2 CPU Time and Tour Cost for Genetic Algorithm with Crossover rate = 0.6, mutation rate = 0.05

No.	No of City	Population size = 50		Population size = 100		Population size = 1000		Population size=10000	
		time	Cost	Time	Cost	Time	Cost	Time	Cost
1	14	13.104	694.82	25.241	**675.84**	246.31	694.82	251.2	**675.84**
2	24	4.493	**1215.74**	28.954	1312.61	6.879	1222.16	68.343	1237.88
3	25	14.57	1130.08	15.78	**1080.09**	19.747	1087.07	79.263	1097.61
4	38	19.251	2734	39.452	2273.59	352.78	2313.28	274.55	**2047.8**
5	39	19.719	2122.79	8.985	1930.63	41.995	1933.2	417.64	**1922.14**
6	49	23.198	3263.86	46.987	**2767.71**	435.81	2977.51	470.89	3024.91
7	63	2.918	6307.28	6.25	6497.75	530.20	**3788.95**	581.09	5413.34

The above Table 4.2 highlights the best cost showed various Time and Cost results depending on the different number of population using crossover rate = 0.6 and mutation rate = 0.05 experimented by GA. The optimal tour costs are obtained at population size = 50, 100, 1000 and 10000. The more the number of Population, the longer the Execution time is.

Table 4.3 CPU Time and Tour Cost for Genetic Algorithm with Population size = 100, Mutation rate = 0.05 and Iteration=1000

No.	No of City	Crossover rate=0.6		Crossover rate=0.7		Crossover rate=0.8		Crossover rate=0.9	
		time	Cost	Time	Cost	Time	Cost	Time	Cost
1	14	24.789	675.84	24.632	726.53	24.172	675.84	2.34	**671.9**
2	24	21.68	1227.48	28.673	1305.91	29.156	1253.44	1.888	**1221.71**
3	25	25.6	1126.94	25.162	1302.37	24.321	1113.83	0.936	**1105.91**
4	38	32.619	2135.37	30.327	2049.01	31.621	2233.94	31.7	**2049.01**
5	39	30.623	2107.43	29.828	2092.31	30.763	2305.72	30.233	**1922.14**
6	49	34.304	2896.95	33.509	2823.33	34.85	3183.47	36.722	**2823.21**
7	63	44.757	4784.74	41.028	4470.39	43.15	4644.46	220.49	**3301.45**

In this system, single point crossover is used for Genetic Algorithm. The effects of making Crossover may get the new better chromosome. S0, the various crossover rates are needed to test.

Generally the best crossover rate is 0.9 rather than 0.6, 0.7 and 0.8 with Population size = 100 and mutation rate = 0.05 according to the results shown in Table 4.3.

Mutation is made to prevent failing GA into local extreme. Mutation rate should be very low because it should not occur very often. Our experimental results shows the best tour costs that obtained at mutation rate = 0.05 with Population size = 100 and crossover rate 0.8 in Table 4.4.

Table 4.4 CPU Time and Tour Cost for Genetic Algorithm with Population size = 100, Crossover rate = 0.8 and Iteration=1000

No.	No of City	Mutation rate=0.05		Mutation rate=0.06		Mutation rate=0.07		Mutation rate=0.08	
		time	Cost	Time	Cost	Time	Cost	Time	Cost
1	14	20.779	675.84	19.921	675.84	19.267	694.82	1.747	**671.9**
2	24	24.866	1362.79	11.623	1218.87	18.221	**1201.7**	24.476	1254.09
3	25	23.992	1162.99	21.149	1141.3	14.539	**1090.35**	5.616	1098.02
4	38	30.218	2337.26	31.293	2519.18	29.297	2560.53	29.746	2355.86
5	39	29.874	**1958.53**	31.075	2237.39	30.794	2329.94	30.061	1967.49
6	49	49.077	3192.23	49.498	3177.54	38.158	**2915.23**	34.289	3519.14
7	63	60.497	4733.83	60.201	4576.28	60.17	**4693.5**	60.357	5146.85

In this system, the number of repetition process of the algorithm is used for the termination of Genetic Algorithm. So this value is needed to adjust to get the best result. The different results of GA depending on this value show in Table 4.5. The optimal tour costs are obtained at Iteration= 1000 and 5000.

If the number of cities increase, the better results are get at the greater number of iteration except the number of cities = 38. Although the

greater number of iteration is good, in our experiments, the GA results are not better with iteration = 10000 compared with iteration = 5000.

Table 4.5 CPU Time and Tour Cost for Genetic Algorithm with Population size = 100, Crossover rate = 0.8 and Mutation rate=0.05

No.	No of City	Iteration=1000		Iteration=3000		Iteration=5000		Iteration=10000	
		time	Cost	Time	Cost	Time	Cost	Time	Cost
1	14	21.512	675.84	107.77	675.84	63.851	**675.84**	214.41	675.84
2	24	28.137	1357.1	84.646	1292.85	140.76	**1250.37**	306.02	1330.93
3	25	18.891	1100.46	22.854	1106.08	18.58	**1084.19**	6.78	1106.22
4	38	44.897	**2109.37**	132.93	2244.86	196.39	2428.18	390.20	2492.42
5	39	39.842	2179.71	118.34	2176.54	115.52	**1936.94**	325.2	2730.08
6	49	36.645	2983.35	109.12	3010.95	184.87	**2887.7**	364.62	2640.81
7	63	42.666	4666.51	131.54	4205.96	220.49	**3301.45**	440.58	3828.55

4.3 Comparison CPU Execution Time and Tour Cost GA with Little' B&B

The same problems by Little's B&B algorithm are experimented. Table 4.5 shows the Time and Cost of Little's B&B on different number of cities and the best results of GA with Population size = 100, crossover rate = 0.9, mutation rate = 0.05 and Iteration=1000 obtained from Table 4.3.

Table 4.6 shows the optimal tour's cost for each TSP problem with both algorithms, Little's B&B and Genetic Algorithm. After comparing B&B with GA, the optimal tour cost of GA is the same B&B's for Eastern Shan state, but for the others, GA is better results than B&B except for Southern & Northern Shan state, and the whole Shan state.

Although the greater the number of cities, the longer the execution time is in Little's B&B, the execution time for GA depends on the termination criteria such as the number of population, the number of iteration. The GA algorithm exits from loop if the best cost is obtained less than B&B's optimal result, so the execution time may be less. Otherwise, the GA algorithm executes iteratively until the desired number

of iteration is reached, so the CPU time increases in proportion with the number of iteration.

Table 4.6 The Comparison of Little's B&B's and GA's Results

Sub-State	No of Cities	Little's B&B		GA	
		Time in seconds	Cost(Km)	Time in seconds	Cost(Km)
Eastern	14	0.53	**671.90**	2.34	**671.90**
Northern	24	3.432	1238.79	1.888	**1221.71**
Southern	25	3.837	1106.77	0.936	**1105.91**
Northern & Eastern	38	9.641	2058.54	31.7	**2049.01**
Southern & Eastern	39	13.088	1940.18	30.233	**1922.14**
Southern & Northern	49	27.799	**2265.29**	36.722	2823.21
The whole Shan State	63	58.459	**3254.45**	220.49	3301.45

The system terminates GA algorithm if the optimal cost of GA is obtained comparing Little's B&B result before the iteration count does not reach. So the execution time for GA may be less.

According to the observation Table 4.6, the CPU time for Little's B&B is less than GA. Although the same result is obtained with both algorithms, GA is longer almost 2 times more than B&B. The CPU time increases in proportion with the number of cities.

CHAPTER 5

CONCLUSION, LIMITATION AND FURTHER EXTENSION

5.1 Conclusion

In this system, symmetric matrices are considered to solve TSP. The system is designed and developed to find optimal route with the minimum cost and to compare the costs and CPU Time between Genetic Algorithm and Little's B&B Algorithm.

Genetic Algorithm is depended on the main operators: population size, crossover rate and mutation rate to get the best solution. To obtain the best solution using Genetic Algorithm, there are needed to adjust these three operators' values. According to the experimental results, crossover rate 90% (0.9) and mutation rate 5% (0.05) are the best rates to get the optimal tour in this system. The more the population size, the longer the execution time is. And then, if the number of iteration increases, the calculation time may take longer. If the number of cities increases, the execution times are longer to get the better results.

Branch and bound is a well-known, all-purpose optimization strategy. One may wish to stop branching when the gap between the upper and lower bounds becomes smaller than a certain threshold. This is used when the solution is "good enough for practical purposes" and can greatly reduce the computations required. In Little's B&B algorithm, the execution time is less because branching is kept to the right unless its lower bound exceeds or equals the costs of a known tour. As a result a few extra nodes may be examined, but usually there will be substantial reduction in the number of time consuming setups of Step 10.

5.2 Limitation

This system is implemented only the cities in Shan State of Myanmar. In Genetic algorithm, many parameters must be specified. The user has to choose the population size, the crossover rate between 0 and 1and the mutation rate between 0 and 1. And then the number of iteration is also chosen. In Branch and Bound Algorithm, it is extremely time consuming.

5.3 Further Extension

The system can be extended for Genetic Algorithm with other several operators such as Partially Edge Recombination Crossover (ERX), Greedy Crossover (GX), Matched Crossover (PMX), Uniform mutation, Creep mutation, etc. And then the system can also be implemented by using branch and cut algorithms instead of branch and bound.

REFERENCES

[1] A.E Eiben and J.E. Smith, "Introduction to Evolutionary Computing" Springer, 2003

[2] Alexander Schrijver, "On the History of Combinatorial Optimization".

[3] B. Kaur and U. Mittal, "Optimization OF TSP Using Genetic Algorithm".

[4] Balas, E. and Toth, P. (1983), "Branch and Bound Methods for the Travelling Salesman Problem" Management Science Research Report.

[5] Hulya Demez, "Combinatiorial Optimization: Soultion Methods of Travelling Salesman Problem" Eastern Mediterranean University, January 2013.

[6] Jin Ouyang, Thomas Weise, Alexandre Devert and Raymond Chiong "A Developmental Approach for Traveling Salesman Problems".

[7] John D.C Little, Katta G. Murty, Dura W. Sweeney, Caroline Karel "An Algorithm of Travelling Salesman Problem".

[8] Jorge Nunes, Luis Matos and Antonio Trigo, "Taxi- Pick-Ups Route Optimization using Genetic Algorithms", Coimbra Institute

of Engineering, Rua Pedro Nunes- Quinta da Nora, 3030-199,Coimbra , Portugal.

[9] K. Aardal, G.L. Nemhauser, R. Weismantel, eds., Elsevier, Amsterdam. "Handbook of Discrete Optimization" 2005, pp. 1–68.

[10] Otman Abound, Jaafar Abouchabaka and Chakir Tajani," Analyzing the Performance of Mutation Operators to solve the Travelling Salesman Problem" Larit Laboratory, Faculty of sciences, Ibn Tofail University, Kenitra, Morocco.otman.fsk@gmail.com, abouch06-univ@yahoo.fr, chakir_tajani@hotmail.fr.

[11] Randy L. Haupt Sue Ellen Haupt, "Practical Genetic Algorithms", Second Edition, A John Wiley & Sons, Inc., Publication.

[12] Rego, César; Gamboa, Dorabela; Glover, Fred; Osterman, Colin (2011) "Travelling Salesman Problem Heuristics: Leading Methods, Implementation and Latest Advances", European Journal of Operational Research 211 (3): 427–441.

[13] Stuart Russell. Peter Norving, "Artificial Intelligence – A Modern Approach". ISBN-81-203-2382-3.

[14] Varunika Arya1, Amit Goyal2 and Vibhuti Jaiswal, "An Optimal Solution to Multiple Travelling Salesman Problem Using Modified Genetic Algorithm", Department of Electronics and

Communication Engineering, Maharishi Markandeshwar University, Sadopur, Ambala, Haryana.

[15] https://en.wikipedia.org/wiki/Mutation_(genetic_algorithm).

[16] https://en.wikipedia.org/wiki/Travelling_salesman_problem.

[17] http://gebweb.net/blogpost/2011/06/24/the –dynamic - programming-algorithm-for-the travelling-salesman-problem/.

[18] http://www.obitko.com/tutorials/genetic-algorithms/crossover-mutation.php.

[19] http://www.obitko.com/tutorials/genetic-algorithms/ encoding.php.

APPENDIXES

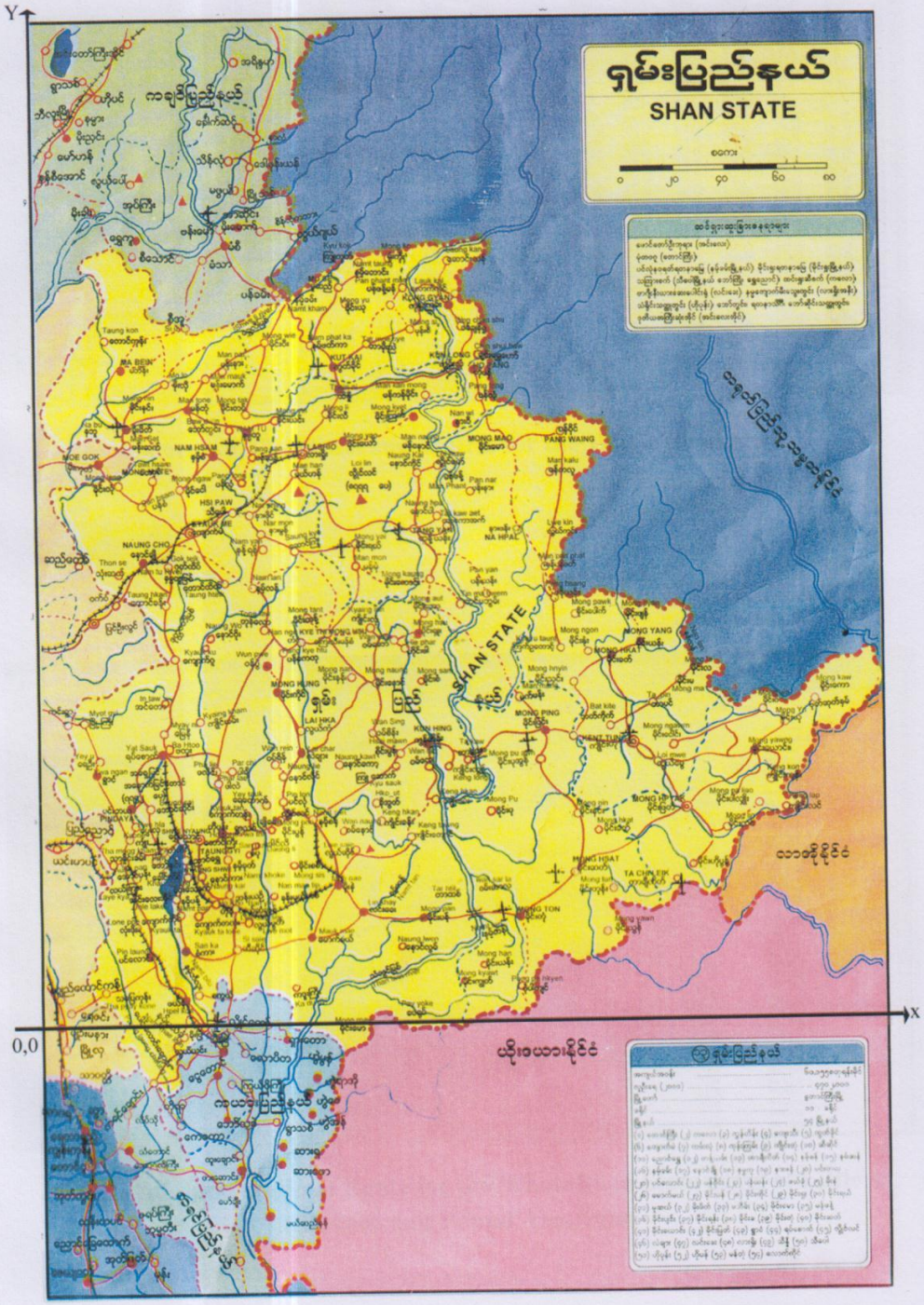


Figure A1. Map of Shan State in Myanmar

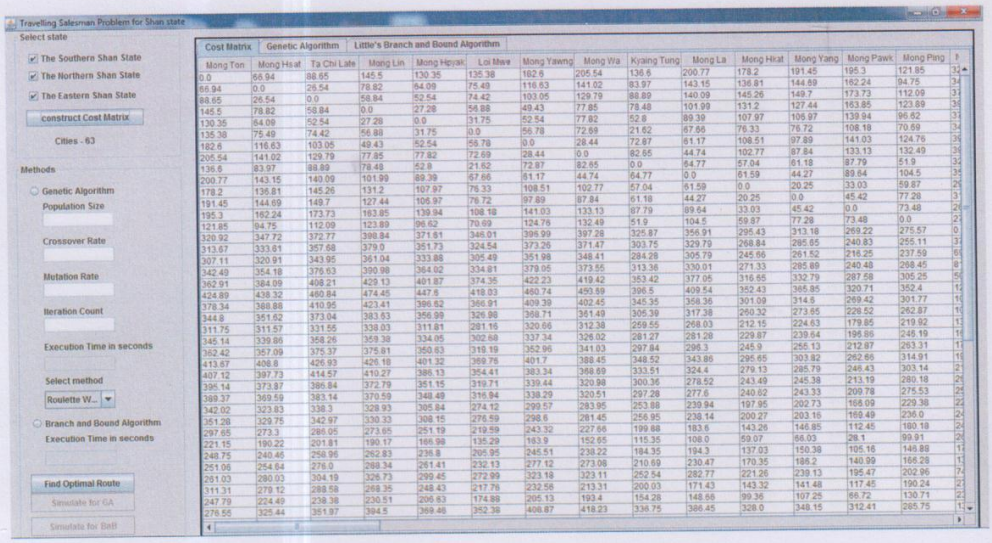


Figure A2. Construct cost matrix for the Whole Shan State

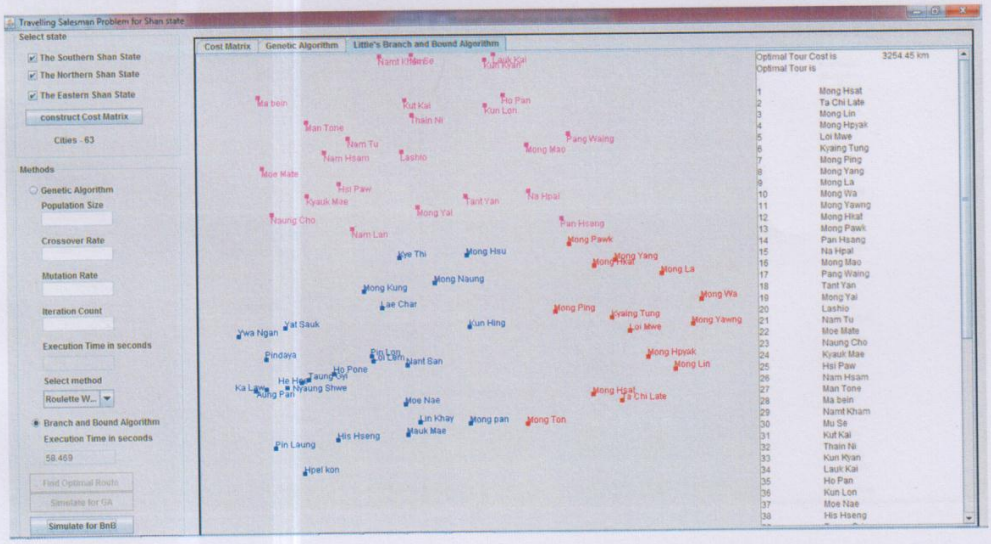


Figure A3. Find Optimal Route for the Whole Shan State using of Little's Branch and Bound Algorithm

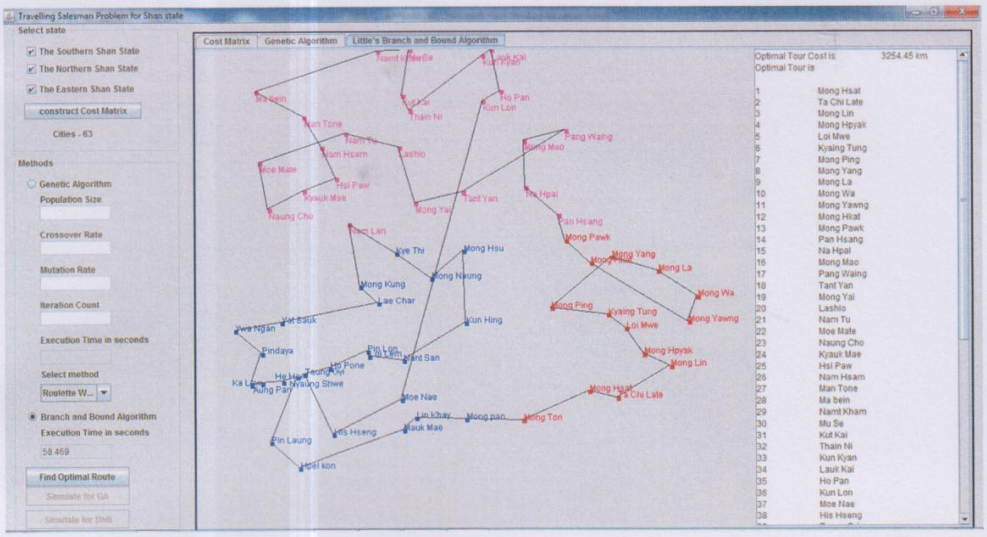


Figure A4. Result of Little's Branch and Bound Algorithm for the whole Shan State

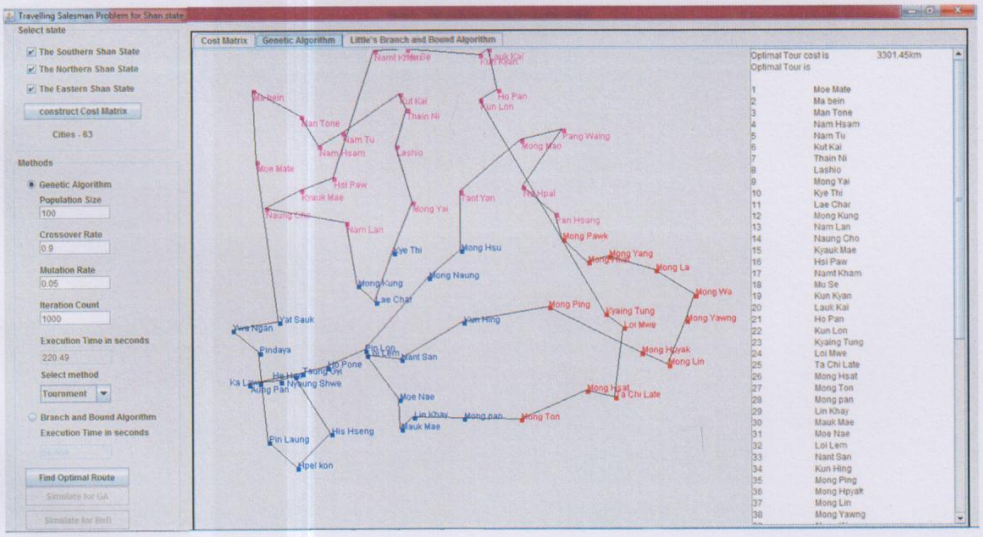


Figure A5. Result of Genetic Algorithm for the whole Shan State