BRANDS USING BANZHAF POWER AND SHAPLEY-SHUBIK POWER INDEX

HNIN INZALI

M.C.Sc. (THESIS) JANUARY, 2016

SOLVING SYMMETRIC TRAVELING SALESMAN PROBLEM (sTSP) USING ANT COLONY SYSTEM (ACS)

NYEIN NYEIN SOE

M.C.Sc. (THESIS) **JANUARY, 2016**

SOLVING SYMMETRIC TRAVELING SALESMAN PROBLEM (sTSP) USING ANT COLONY SYSTEM (ACS)

BY

NYEIN NYEIN SOE

B.C.Sc. (Hons:)

Dissertation submitted in partial fulfillment of the requirements

for

the degree

of

Master of Computer Science

(M.C.Sc.)

COMPUTER UNIVERSITY (MANDALAY)

JANUARY, 2016

ACKNOWLEDGEMENTS

I would like to express my deepest gratitude and sincere thanks to all persons who contributed directly or indirectly towards the success of this thesis.

I owe my respectful thanks to **Dr. Win Aye**, the Rector, Computer University (Mandalay), for her kind permission to make this thesis in this university.

I am deeply thankful to **Dr. Thi Thi Soe**, Associate Professor, Head of Research and Development Department, Computer University (Mandalay), for her continuous, sincere, and kind guidance during the period of completing this thesis.

I am greatly indebted to my supervisor, **Daw Thin Thin Naing**, Associate Professor, Head of Computational Mathematics Department, Computer University (Mandalay), for her valuable supervision, good advice, detailed guidance, and helpful suggestions throughout the terms of finishing the thesis.

I would like to extend my deepest gratitude to **U Thaung Kyaw**, Associate Professor, Head of Department of English at Computer University (Mandalay), for editing my thesis.

Finally, I also thank all the members of Computer University (Mandalay), who attended the seminars on this thesis, for their support, valuable suggestions, helpful hints, and fair criticisms.

Moreover, I also thank all my **mentors and friends**, for their invaluable suggestions and helpful contributions to success of this thesis;

Especially, my deepest thanks go to **my parents** for giving me moral and material support, sympathy and empathy, care and kindness throughout my studies and my efforts to complete this thesis without any trouble.

ABSTRACT

Symmetric Traveling Salesman Problem (sTSP) is a complex problem to find the shortest tour for a salesperson that must start his current company location and go to all cities exactly once in his customer cities list and return started location. sTSP can be illustrated by a fully connected (undirected) graph where vertex (v) defined as cities, edge (e) defined as routes between cities and weights defined as the distance between the pairs of cities. Many methods can be used to find the shortest tour. However, this system uses Ant Colony System (ACS). In this system, a set of cooperating agents are called persons which cooperate to find good solutions for Symmetric Traveling Salesman Problem (sTSP). This system needs datasets and parameter such as α, β, ρ, etc. Number of calculations depends on the parameters. If system parameters are not optimal, this system calculates many occasions to find the shortest tour. Therefore, optimal parameters are very important. Especially, the ACS algorithm parameters $\alpha = 0.1$ and numbers of persons = 10 provide the optimal result for sTSP. Today, traveling salesman problem occurs in real world but it has not satisfied solutions. This system solves on sTSP problems from TSPLIB datasets. This system is implemented by Dot Net Frame work 3.5 with C# programming language based on TSPLIB datasets.

CONTENTS

LIST OF FIGURES

LIST OF EQUATIONS

LIST OF TABLES

CHAPTER 1

INTRODUCTION

Ants exhibit complex social behavior that has long since attracted the attention of human beings. Probably one of the most noticeable behaviors visible to us is the formation of so-called ant streets. When we were young, several of us may have stepped on such an ant highway or may have placed some obstacle in its way just to see how the ants would react to such disturbances. We may have also wondered where these ant highways lead to or even how they are formed. This type of question may become less urgent for most of us as we grow older and go to university, studying other subjects like computer science, mathematics, and so on. However, there are a considerable number of researchers, mainly biologists, who study the behavior of ants in detail. One of the most surprising behavioral patterns exhibited by ants is the ability of certain ant species to find what computer scientists call shortest paths.

In the early 1990s, M.Dorigo and colleagues introduced ant colony optimization (ACO) for the solution of hard combinatorial optimization (CO) problems. Ant colony optimization is a metaheuristics approach, which are approximate algorithms used to obtain good enough solutions to hard combinatorial optimization problems in a reasonable amount of computation time. The inspiring source of ant colony optimization is the pheromone trail laying and following behavior of real ants, which use pheromones as a communication medium. In analogy to the biological example, ant colony optimization is based on the indirect communication of a colony of simple agents, called (artificial) ants, mediated by artificial pheromone trails. The pheromone trails in ant colony optimization serve as distributed, numerical information, which the ants use to construct solutions to solve the problem. The first method of ant colony

optimization is ant system. Ant colony system has been applied to combinatorial optimization problems such as traveling salesman problem (TSP) and the quadratic assignment problem, vehicle routing problem and sequential ordering problem.

Ant colony system (ACS) is the latest method of ant colony optimization. Ant colony system is invented to build on the previous ant system in the direction of improving efficiency when applied to symmetric and asymmetric traveling Salesman Problem. This system applies on symmetric traveling salesman problem (sTSP). The main idea is of the ACS which has a set of agents, called ants, search in parallel for good solutions to the traveling salesman problem, and cooperate through pheromone-mediated indirect and global communication. Informally, each ant constructs a symmetric Traveling Salesman problem (sTSP) solution in an iterative way: it adds new cities to a partial solution by exploiting both information gained from past experience and a nearest neighbor heuristic.

1.1 Motivation of the Thesis

Ant colony optimization (ACO) is a paradigm for designing metaheuristic algorithms for combinatorial optimization problems. The main idea of ACS is to use repeated and often recurrent simulations of artificial ants to generate new solutions to the problem at hand. The ants use information collected during past simulations to direct their search and their information is available and modified through the environment. Ant colony system (ACS) is an extension of ACO and any modifications aimed at improving the performance of ACO can also be applied to ant colony system (ACS) to improve its performance as well. So, this system uses ant colony system (ACS) method.

1.2 Objectives of the Thesis

The objectives of the thesis are:

- To study the Ant Colony Optimization
- To understand how to implement Ant Colony System in step by step
- To implement Symmetric Traveling Salesman Problem using Ant Colony System Algorithm
- To find the shortest possible tour in all cities for a traveling salesperson in order to save traveling time and distance

1.3 Organization of the Thesis

The thesis is organized as follows: chapter 2 is the theory background giving a brief introduction to combinatorial optimization problem, metaheuristic and its methods, ant colony optimization (ACO) and its variants methods, and the applications of ant colony optimization. Chapter 3 is described about the ant colony system for the symmetric traveling salesman problem in detail. Chapter 4 presents the methodology, design and implementation of the system. Chapter 5 gives the conclusion and explores possible directions of future work.

CHAPTER 2

THEORY BACKGROUND

2.1 Combinatorial Optimization Problem

Combinatorial optimization problems are intriguing because they are often easy to state but very difficult to solve [1, 10]. Many of the problems arising in real applications are strongly believed that they cannot be solved to optimally within polynomially bounded computation time. Heuristic algorithm is a algorithm to solve large application has to use approximate method which returns near optimal solution in a relatively short time.

2.2 Metaheuristic

A metaheuristic is a set of algorithmic concepts that can be used to define heuristic methods applicable to a wide set of different problems [1, 8]. In other words, a meta-heuristic can be seen as a general-purpose heuristic method designed to guide an underlying problem-specific heuristic. Some of the best known and most widely applied metaheuristics are simulated annealing, tabu search, iterated local search, evolutionary computation and ant colony optimization. The use of metaheuristics has significantly increased the ability of finding very high-quality, practically relevant combinatorial optimization problems in a reasonable time.

2.2.1 Simulated Annealing

Simulated annealing (SA) is inspired by an analogy between the physical annealing of solids (crystals) and combinatorial optimization problems [1]. In the physical annealing process a solid is first melted and then cooled very slowly, spending a long time at low temperatures, to

obtain a perfect lattice structure corresponding to a minimum energy state. SA transfers this process to local search algorithms for combinatorial optimization problems. SA is a local search strategy which tries to avoid local minima by accepting worse solutions with some probability.

SA has been applied to a wide variety of problems with mixed success. To guarantee convergence to the optimal solution, an impractically slow annealing schedule has to be used and theoretically an infinite number of states have to be visited by the algorithm.

2.2.2 Tabu Search

Tabu search (TS) relies on the systematic use of memory to guide the search process [1]. It is common to distinguish between short-term memory and long-term memory. TS uses a local search that, at every step, makes the best possible move from s to a neighbor solutions s' even if the new solution is worse than the current one; the move that least worsens the objective function is chosen. The use of a short-term memory in the search process is probably the most widely applied feature of TS. TS algorithms that only rely on the use of short-term memory are called simple TS algorithms. To increase the efficiency of simple TS, long-term memory strategies can be used to intensify or diversify the search. Many long-term memory strategies in the context of TS are based on the memorization of the frequency of solution attributes.

Faigle and Kern presented a convergence proof for probabilistic TS; Hanafi and Glover proved that several deterministic variants of TS implicitly enumerate the search space and are also guaranteed to find the optimal solution in finite time.

2.2.3 Iterated Local Search

Iterated local search (ILS) is a simple and powerful metaheuristic; it is starting from an initial solution s; a local search is applied [1]. Once the local search is stuck, the locally optimal solution $\hat{s}$ is perturbed by a move in a neighborhood different from the one used by the local search. This perturbed solution s' is the new starting solution for the local search that takes it to the new local optimal $\hat{s}'$.Finally, an acceptance criterion decides which of the two locally optimal solutions to select as a starting point for the next perturbation step. The main motivation for ILS is to build a randomized walk in a search space of the local optima with respect to some local search algorithm.

Some of the first iterated local search implementations have shown that the approach is very promising and current iterated local search algorithms are among the best-performing approximation methods for combinatorial optimization problems like the traveling salesman problem and several scheduling problems.

2.2.4 Evolutionary Computation

Evolutionary Computation (EC) has become a standard term to indicate problem-solving techniques which use design principles inspired from models of the natural evolution of species [1]. There are three main algorithmic developments within the field of EC: evolution strategies, evolutionary programming and genetic algorithms. Common to these approaches is that they are population-based algorithms that use operators inspired by population genetics to explore the search space. Each individual in the algorithm represents directly or indirectly (through a decoding scheme) a solution to the problem under consideration. The reproduction operator refers to the process of selecting the individuals

that will survive and be part of the next generation. This operator typically uses a bias toward good-quality individuals: The better the objective function values of an individual, the higher the probability that the individual will be selected and therefore be part of the next generation. The recombination operator (often also called crossover) combines parts of two or more individuals and generates new individuals, also called offspring. The mutation operator is a unary operator that introduces random modifications to one individual.

Evolutionary computation algorithm, called population-based incremental learning (PBIL) is mentioned here because of its similarities to ant colony optimization. PBIL maintains a vector of probabilities called the generating vector. Starting from this vector, a population of binary strings representing solutions to the problem under consideration is randomly generated: each string in the population has the i-th bit set to 1 with a probability given by the i-th value on the generating vector. Once a population of solutions is created, the generated solutions are evaluated and this evaluation is used to increase (or decrease) the probabilities of each separate component in the generating vector with the hope that good (bad) solutions in future generations will be produced with higher (lower) probability.

2.2.5 Ant Colony Optimization

Marco Dorigo and colleagues introduced the first ant colony optimization algorithms in the early 1990's. An artificial ant in ant colony optimization is a stochastic constructive procedure that incrementally builds a solution by adding opportunely defined solution components to a partial solution under construction. Therefore, the ant colony optimization metaheuristic can be applied to any combinatorial optimization problem for which a constructive heuristic can be defined. Cooperation is a key

design component of ant colony optimization algorithms: The choice is to allocate the computational resources to a set of relatively simple agents (artificial ants) that communicate indirectly by stigmergy, that is, by indirect communication mediated by the environment .Good solutions are an emergent property of the agents' cooperative interaction. Ant colony optimization algorithms can be used to solve both static and dynamic combinatorial optimization problems. Static problems are those in which the characteristics of the problem are given once and for all when the problem is defined, and do not change while the problem is being solved. An example of such problems is the traveling salesman problem, in which city locations and their relative distances are part of the problem definition and do not change at run time. On the contrary, dynamic problems are defined as a function of some quantities whose value is set by the dynamics of an underlying system. The problem instance changes therefore at run time and the optimization algorithm must be capable of adapting online to the changing environment. Example of dynamics problems is network routing problems in which the data traffic and the network topology can vary in time.

2.2.5.1 Ants' Behavior

Ant colony optimization is a metaheuristic in which a colony of artificial ants cooperates in finding good solutions to difficult discrete optimization problems. The development of these algorithms was inspired by the observation of ant colonies. Ants are social insects. They live in colonies and their behavior is governed by the goal of colony survival rather than being focused on the survival of individuals. The behavior that provided the inspiration for ant colony optimization is the ants' foraging behavior, and in particular, how ants can find shortest paths between food sources and their nest. When searching for food, ants

initially explore the area surrounding their nest in a random manner. While moving, ants leave a chemical pheromone trail on the ground.

Ants can smell pheromone. When choosing their way, they tend to choose, in probability, paths marked by strong pheromone concentrations. As soon as an ant finds a food source, it evaluates the quantity and the quality of the food and carries some of it back to the nest. During the return trip, the quantity of pheromone that an ant leaves on the ground may depend on the quantity and quality of the food. The pheromone trails will guide other ants to the food source. It has been shown in that the indirect communication between the ants via pheromone trails—known as stigmergy enables them to find shortest paths between their nest and food sources.

In most applications, ants construct feasible solutions. However, sometimes it may be necessary or beneficial to also let them construction feasible solutions. It is important that ants act concurrently and independently and that although each ant is complex enough to find a (probably poor) solution to the problem under consideration, good-quality solutions can only emerge as the result of the collective interaction among the ants. This is obtained via indirect communication mediated by the information ants read or write in the variables storing pheromone trail values. In a way, this is a distributed learning process in which the single agents, the ants, are not adaptive themselves but, on the contrary, adaptively modify the way the problem is represented and perceived by other ants.

2.2.5.2 Ant Colony Optimization Metaheuristic

In Figure 2.1, the ant colony optimization metaheuristic is described in pseudo-code [2].

```
Procedure: ACO metaheuristic ScheduleActivities
    ConstructAntSolutions ()
    EvaporatePheromone ()
    DaemonActions () {Optional}
    end ScheduleActivities
end ACO metaheuristic
```

Figure 2.1 Ant Colony Optimization Metaheuristic in Pseudo-Code

The ScheduleActivities construct does not specify how these three activities are scheduled and synchronized. In other words, it does not say whether they should be executed in a completely parallel and independent way, or if some kind of synchronization among them is necessary. The designer is therefore free to specify the way these three procedures should interact, taking into account the characteristics of the considered problem.

ConstructAntsSolutions manages a colony of ants that concurrently and asynchronously visit adjacent states of the considered problem by moving through neighbor nodes of the problem's construction graph. They move by applying a stochastic local decision policy that makes use of pheromone trails and heuristic information.

In this way, ants incrementally build solutions to the optimization problem .Once an ant has built a solution, or while the solution is being built, the ant evaluates the (partial) solution that will be used by the UpdatePheromones procedure to decide how much pheromone to deposit.

EvaporatePheromone is the process by which the pheromone trails are modified. The trails value can either increase, as ants deposit pheromone on the components or connections they use, or decrease, due to pheromone evaporation. From a practical point of view, the deposit of new pheromone increases the probability that those components or connections that were either used by many ants or that were used by at

least one ant and which produced a very good solution will be used again by future ants. Differently, pheromone evaporation implements a useful form of forgetting: it avoids a too rapid convergence of the algorithm toward a suboptimal region, therefore favoring the exploration of new areas of the search space.

DaemonActions is used to implement centralized actions which cannot be performed by single ants. Examples of daemon actions are the activation of a local optimization procedure, or the collection of global information that can be used to decide whether it is useful or not to deposit additional pheromone to bias the search process from a nonlocal perspective. As a practical example, the daemon can observe the path found by each ant in the colony and select one or a few ants (e.g., those that built the best solutions in the algorithm iteration) which are then allowed to deposit additional pheromone on the components/connections they used. The procedure DaemonActions is optional and refers to centralized actions executed by a daemon possessing global knowledge.

Nowadays numerous successful implementations of the ant colony optimization metaheuristic are available and they have been applied to many different combinatorial optimization problems.

2.3 Ant Colony Optimization for Traveling Salesman Problem

Ant colony optimization can be applied to the traveling salesman problem in a straightforward way, the construction graph G = (C, L), where the set L fully connects the components C, is identical to the problem graph, that is, C = N and L = A; the set of states of the problem corresponds to the set of all possible partial tours; and the constraints Ω enforce that the ants construct only feasible tours that correspond to permutations of the city indices [1]. This is always possible, because the

construction graph is a complete graph and any close path that visits all the nodes without repeating any node corresponds to a feasible tour.

In all available ant colony optimization algorithms for the traveling salesman problem, the pheromone trails are associated with arcs and therefore τ_{ij} refers to the desirability of visiting city j directly after city i. The heuristic information is chosen as $\eta_{ij} = 1/d_{ij}$, that is, the heuristic desirability of going from city i directly to city j is inversely proportional to the distance between the two cities. In case d_{ij} for some arc (i, j), the corresponding η_{ij} is set to a very small value. For implementation purposes, pheromone trails are collected into a pheromone matrix whose elements are the τ_{ij}'s. This can be done analogously for the heuristic information.

Tours are constructed by applying the following simple constructive procedure to each ant: (1) choose, according to some criterion, a start city at which the ant is positioned; (2) use pheromone and heuristic values to probabilistically construct a tour by iteratively adding cities that the ant has not visited yet (see Figure 2.1), until all cities have been visited; and (3) go back to the initial city. After all ants have completed their tour, they may deposit pheromone on the tours they have followed. Before adding pheromone, the tours constructed by the ants may be improved by the application of a local search procedure. This high-level description applies to most of the published ant colony optimization algorithms for the traveling salesman problem, one notable exception being ant colony system in which pheromone evaporation is interleaved with tour construction.

This algorithm's scheme is shown in Figure 2.2; after initializing the parameters and the pheromone trails, these ant colony optimization algorithms iterate through a main loop, in which first all of the ants' tours

are constructed, then an optional phase takes place in which the ants' tours are improved by the application of some local search algorithm, and finally the pheromone trails are updated. This last step involves pheromone evaporation and the update of the pheromone trails by the ants to reflect their search experience. In Figure 2.2, the DaemonActions procedure of Figure 2.1 is replaced by the ApplyLocalSearchprocedure , and by a routine (not shown in the figure and most often integrated in the UpdatePheromones procedure to facilitate implementation) that helps selecting the ants that should be allowed to deposit pheromone.

```
procedure ACOMetaheuristicStatic
    Set parameters, initialize pheromone trails
    while (termination condition not met) do
        ConstructAntsSolutions
        ApplyLocalSearch %optional
        UpdatePheromones
    end
end-procedure
```

Figure 2.2 Algorithmic Skeleton for Ant Colony Optimization Algorithm Applied to "Static" Combinatorial Optimization Problems

As already mentioned, the first ant colony optimization, ant system algorithm was introduced using the traveling salesman problem as an example application [1, 7, 11]. Ant system achieved encouraging initial results, but was found to be inferior to state-of-the-art algorithms for the traveling salesman problem. The importance of ant system therefore mainly lies in the inspiration it provided for a number of extensions that significantly improved performance and are currently among the most successful ant colony optimization algorithms. In fact, most of these extensions are direct extensions of ant system in the sense that they keep

the same solution construction procedure as well as the same pheromone evaporation procedure. These extensions [1, 7] include:

- Elitist Ant System,
- Rank-based Ant System,
- MAX–MIN Ant System,
- Ant-Q,
- Ant Colony System (ACS),
- ANTS algorithm and
- Hyper-cube Ant System.

In these extensions, ANTS algorithm and Hyper-cube ant system do not solve the traveling salesman problem. So, we presented about ant system (AS), elitist ant system, rank-based ant system, Min-Max ant system and ant colony system in this chapter.

2.3.1 Ant System and Its Direct Successors

In this section we present ant system and those ant colony optimization algorithms that are largely similar to ant system [2]. We do not consider the use of the optional local search phase. Initially, three different versions of ant system were proposed which are explained below:

- The ant-cycle is the approach discussed so far
  - Information is updated at the end of each tour as such function tour length
- The ant-density is an approach where in the pheromone quantity Q is deposited once the segment is traversed
  - Pretty much a greedy approach (local information) and not really providing relative information

- The ant-quantity is an approach where in the pheromone quantity Q/d_{ij} is deposited once the segment is traversed
  - Also a greedy approach but providing some relative information by scaling Q by the length of the segment

Nowadays, when referring to ant system, one actually refers to ant-cycle since the two other variants abandoned because of their inferior performance. Difference between ant system and these extensions are the way the pheromone update is performed, as well as some additional details in the management of the pheromone trails.

2.3.2 Ant System

Ant system was the first ant colony optimization algorithm to be proposed. Its main characteristic is that the pheromone values are updated by all the ants that have completed the tour [1, 6]. This is done by first lowering the pheromone value on all arcs by a constant factor, and then adding pheromone on the arcs the ants have crossed in their tours. Pheromone evaporation is implemented by

$$\tau_{ij} \leftarrow (1-\rho)\,\tau_{ij},\ \forall (i,j) \in L, \tag{2.1}$$

After evaporation, all ants deposit pheromone on the arcs they have crossed in their tour:

$$\tau_{ij} \leftarrow \tau_{ij} + \sum_{k=1}^{m} \Delta\tau_{ij}^{k},\quad \forall (i,j) \in L, \tag{2.2}$$

where ρ is the evaporation rate, m is the number of ants, and $\Delta\tau_{ij}^{k}$ is the quality of pheromone per unit length laid on edge (i , j) by the k^{th} ant:

$$\Delta\tau_{ij}^{k} = \begin{cases} \frac{1}{L_k}, & \text{if arc (i, j) belongs to } T^k; \\ 0, & \text{otherwise;} \end{cases} \tag{2.3}$$

$$\Delta\tau_{i,j}^{bs}(t) = \begin{cases} 1/C^{bs}(t), & \text{if arc}(i,j) \in T^{bs} \\ 0, & \text{otherwise} \end{cases} \quad (2.6)$$

Note that in elitist ant system, as well as in all other algorithms, pheromone evaporation is implemented as in ant system. Computational results presented in suggest that the use of the elitist strategy with an appropriate value for parameter e allows ant system both finding better tours and finding them in a lower number of iterations.

2.3.4 Rank-Based Ant System

Another improvement over AS is the rank-based version of ant system (AS_{rank}) proposed by Bullnheimer [1, 2]. In AS_{rank} each ant deposits an amount of pheromone that decreases with its rank. Additionally, as in Elitist Ant System, the best-so-far ant always deposits the largest amount of pheromone in each iterations.

Before updating the pheromone trails, the ants are sorted by increasing tour length and the quantity of pheromone an ant deposits is weighted according to the rank r of the ant. This can be solved randomly. In each iteration only (w – 1) best ranked ants and the ant that produced the best-so-far tour (this ant does not necessarily belong to the set of ants of the current algorithm iteration) are allowed to deposit pheromone. The best-so-far tour gives the strongest feedback, with weight w (i.e., its contribution $1/C^{bs}$ is multiplied by w); the r^{th} best ant of the current iteration contributes to pheromone updating with the value $1/C^{r}$ multiplied by a weight given by max {0, w – r}. Thus, the AS_{rank} pheromone update rule is

$$\tau_{ij}(t+1) = (1-\rho)\,\tau_{ij}(t) + \sum_{r=1}^{w-1}(w-r)\Delta\tau_{ij}^{r}(t) + w\Delta\tau_{ij}^{gb}(t) \quad (2.7)$$

where $\Delta\tau_{ij}^{r} = 1/C^{r}$ and $\Delta\tau_{ij}^{gb} = 1/C^{bs}$. The results of an experimental evaluation suggest that AS_{rank} performs slightly better than elitist ant system and significantly better than ant system.

2.3.5 MAX-MIN Ant System

MAX-MIN Ant system is an improvement over the original ant system idea [1,2]. MAX-MIN ant system was proposed by Stutzle and Hoos, who introduced a number of changes of which the most important are the following:

- only the best ant can update the pheromone trails, and
- the minimum and maximum values of the pheromones are limited.

Equation (2.1) takes the following new form:

$$\tau_{ij} \leftarrow (1\text{-}\rho) \cdot \tau_{ij} + \Delta\tau_{ij}^{best} \tag{2.8}$$

where $\Delta\tau_{ij}^{best}$ is the pheromone update value defined by

$$\Delta\tau_{ij}^{best} = \begin{cases} \frac{1}{L_{best}}, & \text{if the best ant used edge (i,j) in its tour} \\ 0, & \text{otherwise} \end{cases} \tag{2.9}$$

L_{best} is the length of the tour of the best ant. This may be either the best tour found in the current iteration best, L_{ib}- or the best solution found since the start of the algorithm best-so-far, L_{bs}- or a combination of both.

Concerning the limits on the minimal and maximal pheromone values allowed, respectively τ_{min} and τ_{max} , Stutzle and Hoos suggest that they should be chosen experimentally based on the problem. The maximum value τ_{max} may be calculated analytically provided that the optimum ant tour length is known. In the case of TSP, τ_{max} is given by

$$\tau_{max} = \frac{1}{\rho} \cdot \frac{1}{L^{*}} \tag{2.10}$$

where L^* is the length of the optimal tour. If L^* is not known, it can be approximated by L_{bs}.Stutzle and Hoos present an analytical approach to finding this value based on the probability ρ_{best} that an ant constructs the best tour found. First, it is assumed that at each construction step an ant has a constant number k of options available. The probability that an ant makes the right decision at each of n steps is given by $\rho_{dec} = \sqrt[n-1]{\rho_{best}}$ .The minimum pheromone value τ_{min} should be chosen from

$$\tau_{min} = \frac{\tau_{max} \cdot (1-\rho_{dec})}{k.\rho_{dec}} \tag{2.11}$$

The process of pheromone update in MAX-MIN Ant system is concluded by verifying that all pheromone values are within the imposed limits:

$$\tau_{ij} = \begin{cases} \tau_{min} \ if \ \tau_{ij} < \tau_{min} \\ \tau_{ij} \ \ if \ \tau_{min} \leq \tau_{ij} \leq \tau_{max} \\ \tau_{max} \ if \ \tau_{ij} > \tau_{max} \end{cases} \tag{2.12}$$

MAX-MIN ant system provided a significant improvement over the basic ant system performance. The first implementations focused on the traveling salesman problem, the generalized assignment problem and the set-covering problem.

2.4 Applications of Ant Colony Optimization

The best way of illustrating how the ant colony optimization metaheuristic operates is by describing how it has been applied to combinatorial optimization problems. This is done with a full and detailed description of most of the current applications of ant colony optimization [1, 9]. A brief description of the main points to consider when applying ant colony optimization algorithms to a few examples of

problems representative of important classes of optimization problems are as follows.

2.4.1 Application to Routing Problem

The traveling salesman problems (TSP) is the first problem that the ant algorithms were applied due to the high degree of similarity between ants finding the shortest path to a food source and artificial ants finding the shortest Hamiltonian cycle in a graph.

The traveling salesman problem is the problem faced by a salesman who, starting from his hometown, wants to find a shortest possible trip through a given set of customer cities, visiting each city once before finally returning home. The traveling salesman problems can be represented by a complete weighted graph G= (N, A) with N being the set of n= | N | nodes (cities). A be the set of arcs fully connecting the nodes. Each arc (i , j) ϵ A is assigned a weight d_{ij} which represents the distance between cities i and j. The TSP is the problem of finding a minimum length Hamiltonian circuit of the graph, where a Hamiltonian circuit is a closed walk (a tour) visiting each node of G exactly once. Consequently, each major Ant Colony Optimization algorithm has been applied to either the symmetric Traveling Salesman Problems or asymmetric traveling salesman problems (ATSP). Symmetric traveling salesman problems is the distances between the cities are independent of the direction of traveling the arcs, that is $d_{ij} = d_{ji}$ for every pair of nodes, and the symmetric Traveling Salesman Problems (STSP), where at least for one pair of nodes i , j has $d_{ij} \neq d_{ji}$.

Construction graph: The construction graph is identical to the problem graph: the set of components to the set of nodes (i.e., C = N), the connections correspond to the set of arcs (i.e., L = A), and each connection has a weight which corresponds to the distance d_{ij} between

nodes i and j; the states of the problem are the set of all possible partial tours.

- **Constraints**: The only constraint in the TSP is that all cities have to be visited and that each city is visited at most once. This constraint is enforced that ants have to choose the next city only among those it has not visited yet at each construction step.
- **Pheromone trails and heuristic information**: The pheromone tails π_{ij} in the Traveling Salesman Problems refer to the desirability of visiting city j directly after i. The heuristic η_{ij} is typically inversely proportional to the distance between cities i and j, a straightforward choice being $\eta_{ij} = 1/d_{ij}$. In fact, this is also the heuristic information used in most Ant Colony optimization algorithms for the Traveling Salesman Problems.
- **Solution Construction**: Each ant is initially put on a randomly chosen start city and at each step iteratively adds one still unvisited city to its partial tour. The solution construction terminates once all cities have been visited.

Another problem like traveling salesman problems is the vehicle routing problem (VRP), which consists of delivering commodities of different weights to a number of customer/ cities using vehicles with specific capacities: minimize the number of vehicles used, the total distance travelled. An added complication is that vehicles must start at and return to a warehouse. Each vehicle also has a maximum distance/ time for its route. Bullnheimer and colleagues developed a version of their rank-based ant system algorithm for this problem. They note that once customers have been assigned to vehicles, the Vehicle Routing Problem reduces to a number to traveling salesman problems. Hence, their algorithm is applied to the vehicle routing problem in the same way

as ant colony optimization algorithms for the traveling salesman problems, with the modification. When the current route makes it impossible to select another customer without violating a vehicle's capacity, or would exceed the vehicle's maximum route length, the next component chosen is the component representing the warehouse.

Another one is the sequential ordering problem (SOP) for production planning. Sequential ordering problem finds a minimum Hamiltonian path in directed graph with weight on the edges and on the nodes subject to the precedence constraints among nodes. With some additional precedence constraints, sequential ordering problem can be modeled as an asymmetric traveling salesman. By applying this, hybrid ant system for the sequential ordering problem (HAS-SOP) was published by Gambardella and Dorigo. HAS-SOP uses a local search heuristic, a relatively novel feature at the time of their proposed, to improve the solutions.

2.4.2 Application to Scheduling Problem

A scheduling problem is a problem for solving the optimal schedule under various objectives, different machine environment, and characteristic of the various jobs. There is a myriad of terms related to this problem. We discuss some of this type of problems.

In 1994, an ant colony optimization algorithm for the job shop problem (JSP) was developed by Colorni, Dorigo, Maniezzo and Trubian. The construction graph they define is made up of the operations to be scheduled plus an additional node representing the empty sequence from which ants start. This extra node is required as pheromone is associated with the edges of the graph and, unlike the TSP, it is important to know which operation appears first in the schedule. Again in 1999, Vander Zwaan and Marques describe an ant colony optimization algorithm

similar to above colonic and colleagues'; with only minor changes to the way pheromone is updated. Neither of these approaches was considered highly effective.

The majority of scheduling ant colony optimization algorithm associates pheromone with the absolute position of an operation in the permutation. This innovative approach removes the need for an artificial start node and appears better suited to describing permutation than associating pheromone with pairs of adjacent components. This approach has been used in ant colony optimization algorithm algorithms for the flow shop problem (FSP), and single machine total tardiness problem (SMTTP).

2.4.3 Application to Assignment Problems

Assignment problems involve the allocation of resources to items subject to a number of constraints. The distinguishing characteristics of these problems are that they contain two distinct types of entity "items and resources", and that while all items must be assigned a resource, the number of items assigned to one resource can very form zero to many depending on the problem.

In a general framework for solving assignment problems, two pheromones are used in their system; one is used to select the next item to be assigned and another is used to determine which resource to assign to (GAP) has taken this second one approach, and associate pheromone with the assignments made. The addition of a local search procedure improved results for the algorithm significantly.

The quadratic assignment problem (QAP) is a facility layout problem, with applications in building and office layout, keyboard design and scheduling. It consists of assigning n facilities (i.e., items) to n locations (i.e., resources), with exactly one facility assigned to each

location. The aim is to minimize the total cost of flows between facilities. Many ant colony optimization algorithm algorithms have been developed for this problem. Many of these Ant Colony Optimization algorithms produced results competitive with alternative solution approaches.

2.4.4 Application to Knowledge Discovery Problems

Knowledge discovery problems (KPD) involved the extraction or mining knowledge from large amounts of data. There are many other terms carrying a similar or slightly different meaning to knowledge discovery problems, such as knowledge mining from databases, knowledge extraction, data/pattern analysis, data archaeology, data dredging, and data mining. It involves an integration of techniques from multiple disciplines such as database technology, statistic, machine learning, high-performance computing, pattern recognition, neutral networks, data visualization, information retrieval, image and signal processing, and spatial data analysis. We adopt a database perspective in our presentation of knowledge discovery problems in this dissertation. That is, emphasis is placed on efficient and scalable knowledge discovery techniques for large database.

Knowledge discovery problems may appear that it does not have many properties in common. However, recent studies suggest that they can be used together for several real world data mining problems especially when other methods would be too expensive or difficult to implement. Some of the past and ongoing works of ant colony optimization algorithm in knowledge discovery problems are presented.

CHAPTER 3

ANT COLONY SYSTEM FOR THE TRAVELING SALESMAN PROBLEM

3.1. Ant Colony System

Ant colony system differs from ant system in three main points [4, 5]. First, it exploits the search experience accumulated by the ants more strongly than ant system does through the use of a more aggressive action choice rule. Second, pheromone evaporation and pheromone deposit take place only on the arcs belonging to the best-so-far tour. Third, each time an ant uses an arc (i, j) to move from city i to city j, it removes some pheromone from the arc to increase the exploration of alternative paths. In the following, these innovations are presented in more detail.

3.1.1 Tour Construction

In ant colony system, when located at city i, ant k moves to a city j chosen according to the so-called pseudorandom proportional rule [1, 3], given by

$$j = \begin{cases} \text{argmax}_{l \in N_i^k}\{\tau_{il}[\eta_{il}]^\beta\}, & \text{if } q \leq q_0 \\ J, & \text{otherwise} \end{cases} \qquad (3.1)$$

where q is an random variable uniformly distributed in (0,1), $q_0 (0 \leq q_0 \leq 1)$ is a parameter, and J is a random variable selected according to the probability distribution given by equation (2.4) (with $\alpha = 1$).

In other words, with probability q_0 the ant makes the best possible move as indicated by the learned pheromone trails and the heuristic information, while with probability $(1 - q_0)$ it performs a biased exploration of the arcs. Tuning the parameter q_0 allows modulation of the

degree of exploration and the choice of whether to concentrate the search of the system around the best-so-far solution or to explore other tours.

3.1.2 Local Pheromone Trail Update

In ant colony system, the ants use a local pheromone update rule that they apply immediately after having crossed an arc (i, j) during the tour construction:

$$\tau_{ij} \leftarrow (1 - \rho)\,\tau_{ij} + \rho\tau_0 \qquad (3.2)$$

where ρ, $0 < \rho < 1$, and τ_0 are two parameters [1, 3]. The value of τ_0 is set to be the same as the initial value for the pheromone trails. Experimentally, a good value for ξ was found to be 0.1, while a good value for τ_0 was found to be $1/nC_{nn}$, where n is the number of cities in the traveling salesman problem instance and C_{nn} is the length of a nearest-neighbor tour. The effect of the local updating rule is that each time an ant uses an arc (i, j) its pheromone trail τ_{ij} is reduced, so that the arc becomes less desirable for the following ants. This allows an increase in the exploration of arcs that have not been visited yet and has the effect that the algorithm does not show a stagnation behavior. It is important that, while for the previously discussed ant system variants it does not matter whether the ants construct the tours in parallel or sequentially, this makes a difference in ant colony system because of the local pheromone update rule. In most ant colony system implementations the choice has been to let all the ants move in parallel, although there is, at the moment, no experimental evidence in favor of one choice or the other.

3.1.3 Global Pheromone Trail Update

In ant colony system only one ant (the best-so-far ant) is allowed to add pheromone values after calculating once iteration. Thus, the update in ant colony system is implemented by the following equation:

$$\tau_{ij} \leftarrow (1-\alpha)\,\tau_{ij} + \alpha\Delta\tau_{ij}^{bs},\ \forall(i,j) \in T^{bs}, \qquad (3.3)$$

where $\Delta\tau_{ij}^{bs} = 1/\,C^{bs}$ [1, 3]. It is important that in ant colony system the pheromone trail update, both evaporation and new pheromone deposit, only applies to the arcs of T^{bs}, not to all the arcs as in ant system. This is important, because in this way the computational complexity of the pheromone update at each iteration is reduced from O(n^2) to O(n),where n is the size of the instance being solved. As usual, the parameter ρ represents pheromone evaporation; unlike ant system's equations (2.1) and (2.2), in equation (3.3) the deposited pheromone is discounted by a factor ρ ; the result in the new pheromone trail being a weighted average between the old pheromone value and the amount of pheromone deposited.

In this system, the symbols define as follows:

- $\tau(r,s)$: amount of pheromone trail on route (r,s)
- $\eta(r,s)$: 1/distance (r,s) is the inverse of the distance on route(r,s)
- $J_k(r)$: the set of cities that remain to be visited by person k positioned on cities r
- A : parameter which weighs the relative importance of heuristic visibility
- β : a parameter which weighs the relative importance of pheromone trail and of closeness

- q : a value chosen randomly with uniform between 0 and 1
- q_0 : a parameter that decides the probability to make random choices or to exploit the routes with higher pheromone
- τ_0 : a very small constant value calculated by $1/ nC_{nn}$
- C_{nn} : length the nearest neighbor tour
- n : number of cities
- ρ : a value chosen randomly with uniform between 0 and 1
- L_{gb} : the length of the global best tour
- m : the numbers of person

In initial experiments, the use of the iteration-best tour was also considered for the pheromone updates. Although for small traveling salesman problem instances the differences in the final tour quality obtained by updating the pheromones using the best-so-far or the iteration-best tour was found to be minimal, for instances with more than 100 cities the use of the best-so-far tour gave far better results.

3.2 Behavior of Ant Colony System

Ant colony system generates tours that differ only in a relatively small number of arcs from the best-so-far tour T^{bs} [1, 2]. This is achieved by choosing a large value for q_0 in the pseudorandom proportional action choice rule [equation (3.1)], which leads to tours that have many arcs in common with the best-so-far tour. An interesting aspect of ant colony system is that while arcs are traversed by ants, their associated pheromone is diminished, making them less attractive, and therefore favoring the exploration of still unvisited arcs. Local updating has the effect of lowering the pheromone on visited arcs so that they will be

chosen with a lower probability by the other ants in their remaining steps for completing a tour.

3.3 The Ant Colony System Algorithm

1. /* Initialization phase */
   For each pair (r,s) $\tau(r,s) := \tau_0$ **End-for**
   For k : = 1 to m **do**
   Let r_{k1} be the starting city for agent k
   $J_k(r_{k1}) := \{1, \ldots., n\} - r_{k1}$
   /* $J_k(r_{k1})$ is the set of yet to be visited cities for agent k in city r_{k1} */
   $r_k := r_{k1}$ /* r_k is the city where agent k is located*/
   End-for
2. /* This is the phase in which agents build their tours. The tour of agent k is stored in $Tour_k$.*/
   For i : =1 to n **do**
   If i < n
   Then
   For k := 1 to m **do**
   Choose the next city s_k according to equation (3.4) and equation (3.1)
   If i < n-1 **Then** $J_k(s_k) := J_k(r_k) - s_k$
   If i = n-1**Then** $J_k(s_k) := J_k(r_k) - s_k + r_{k1}$
   $Tour_k(i) := (r_k, s_k)$
   End-for
   Else
   For k := 1 to m **do**
   /* In this cycle all the agents go back to the initial city r_{k1} */
   $s_k := r_{k1}$
   $Tour_k(i) := (r_k, s_k)$
   End-for
   /* In this phase local updating is computed and τ-values are updated */
   For k := 1 to m **do**
   $\tau(r_k, s_k) \leftarrow (1-\rho)\,\tau(r_k, s_k) + \rho\tau_0$

$r_k := s_k$ /* New city for agent k */

End-for

End-for

3. /* In this Phase delayed reinforcement is computed and τ-values are updated */

For k := 1 to m **do**

Compute L_k /* L_k is the length of the tour done by agent k */

End-for

Compute $L_{best-iter}$

/* Update edges belonging to $L_{best-iter}$ */

For each edge (r , s)

$\tau(r_k, s_k) \leftarrow (1-\alpha)\,\tau(r_k, s_k) + \alpha(L_{best-iter})^{-1}$

End-for

4. **If** (End-condition ==true)

then Print shortest of L_k

else goto Phase 2

Figure 3.1 Ant Colony System Algorithm

The stage 1 of the algorithm initializes the algorithm parameters and stage 2 chooses the next city apply on the state transition rule or random proportional rule. Then, it uses the local updating rule. Stage 3 computes the total distance for each person and updates the pheromone value by choosing the person who found the shortest tour apply on global updating rule. Stage 4 checks the termination condition.

The good value of the ACS algorithm parameters are shows in table 3.1. It should be clear that each individual instance, different parameters produce different result. However, good parameters can give the optimal result for sTSP instances [1, 3].

Table 3.1 The Good Value of the ACS Algorithm Parameters

Parameter	Value
β	2 to 5
q	0 to1
ρ	0.1
τ_0	a very small constant value from 1/nC_{nn}
q_0	0.9
n	number of cities
m	10
α	0.1

3.4 The Algorithm Analysis

The main tasks to be considered in an ant colony system algorithm [1] are the data initialization, solution construction and termination condition. In addition, the data structures and parameters need to be initialized and some statistics about the run need to be maintained.

3.4.1 Data Initialization

In the data initialization, (1) the instance has to be read; (2) the distance matrix has to be computed; (3) the nearest-neighbor lists for all cities have to be computed; (4) the pheromone matrix and the choice-info matrix have to be initialized; (5) the ants have to be initialized; (6) the algorithm's parameters must be initialized; and (7) some variables that keep track of statistical information, such as the used CPU time, the number of iterations, or the best solution found so far, have to be initialized.

```
Procedure InitializeData
    ReadInstance
    ComputeDistances
    ComputeNearestNeighborLists
    ComputeChoiceInformation
    InitializeAnts
    InitializeParameters
    InitializeStatistics
end-procedure
```

Figure 3.2 Procedure to Initialize the Ant Colony System Algorithm

3.4.2 Solution Construction

The tour construction is managed by the procedure constructSolutions. The solution construction requires the following phases.

- First, the ants' memory must be emptied. This is done in lines 1 to 5 of procedure constructSolutions (Figure 3.3) by marking all cities as unvisited, that is, by setting all the entries of the array ants.visited to false for all the ants.
- Second, each ant has to be assigned an initial city. One possibility is to assign each ant a random initial city. This is accomplished in lines 6 to 11 of the procedure (Figure 3.3). The function random returns a random number chosen according to a uniform distribution over the set {1,...,n}.
- Next, each ant constructs a complete tour. At each construction step (see the procedure in Figure 3.3) the ants apply the ant colony system pseudorandom proportional action choice rule [equation (3.1)]. The procedure ant colony system pseudorandom proportional action choice rule implements the

action choice rule and takes as parameters the ant identifier and the current construction step index.

- Finally, in lines 18 to 21(see the procedure in Figure 3.3), the ants move back to the initial city and the tour length of each ant's tour is computed. Remember that, in the tour representation we repeat the identifier of the first city at position n+1; this is done in line 19.

The all states described in above, the solution construction of all of the ants is synchronized in such away that the ants build solutions in parallel. The same behavior can be obtained, by ants that construct solutions sequentially, because the ants do not change the pheromone trails at construction time (this is not the case for ant colony system, in which case the sequential and parallel implementations give different results).

```
procedure ConstructSolutions
1     for k = 1 to m do
2      for i = 1 to n do
3         Ant [k].visited[i] ← false
4      end-for
5     End-for
6     Step ← 1
7     for k = 1 to m do
8       r ← random {1,…,n}
9       ant [k].tour[step] ← r
10      ant [k].visited[r] ← true
11    end-for
12    while (step < n) do
13      step ← step + 1
14      For k = 1 to m do
15      ACSpseudorandomproportionalactionchoicerule(k,step)
```

```
    end-for
  end-while
  for k = 1 to m do
    ant [k].tour[n+1] ← ant [k].tour[1]
    Ant[k].tour_length ← ComputeTourLength(k)
  end-for
end-procedure
```

Figure 3.3 Pseudo-Code of the Solution Construction Procedure for ACS

3.4.3 Termination Condition

The program stops if at least one termination condition applies. Possible termination conditions are: (1) the algorithm has found a solution within a predefined distance from a lower bound on the optimal solution quality (2) a maximum number of tour constructions or a maximum number of algorithm iterations has been reached (3) a maximum CPU time has been spent (4) the algorithm shows stagnation behavior. A maximum number of tour constructions is used or a maximum number of algorithm iterations has been reached.

3.5 Symmetric Traveling Salesman Problem

3.5.1 Data Structures

As a first step, the basic data structures have to be defined [1]. These must allow storing the data about the traveling salesman problem instance and the pheromone trails, and representing artificial ants.

3.5.2 Problem Representation

- **Intercity Distance:** Often a **Symmetric Traveling Salesman Problem** instance is given as the coordinates of a number of n points. In this case, one possibility would be to store the x and y coordinates of the cities in two arrays and then computes the fly

distance between the cities as needs [13]. To find the distance between two cities for symmetric traveling Salesman Problem by calculating the formula as below:

$$\text{distance}_{ij}\,(d_{ij}) = \sqrt{(x_j - x_i)^2 + (y_j - y_i)^2} \qquad (3.4)$$

However, this leads to a significant computational overhead, it is more reasonable to pre-compute all intercity distances and to store them in a symmetric distance matrix with n^2 entries. In fact, we need to store n(n-1)/2 distinct distances for symmetric traveling salesman problem, it is more efficient to use an n^2 matrix to avoid performing additional operations. Fortunately, in these cases there exist some intermediate possibilities, such as storing the distances between a city and the cities of its nearest-neighbor list, that greatly reduce the necessary amount of computation. It is also important to know that, for historical reasons, in almost all the traveling salesman problem literature, the distances are stored as integers. In fact, in old computers integer operations used to be much faster than operations on real numbers, so that by setting distances to be integers, much more efficient code could be obtained.

- **Nearest-Neighbor Lists:** In addition to the distance matrix, it is convenient to store for each city a list of its nearest neighbors. Let d_{ij} be the list of the distances from a city i to all cities j, with j = 1,…,n and $i \neq j$. The nearest-neighbor list of a city i is obtained by sorting the list d_i according to non-decreasing distances. Complexity of Nearest-neighbor lists for all cities is $O(n^2 \text{ Log } n)$. It should be noted that the use of truncated nearest-neighbor lists can make it impossible to find the optimal solution.

- **Pheromone Trails:** In addition to the instance-related information, we also have to store for each connection (i,j) a number τ_{ij} corresponding to the pheromone trail associated with that connection. For symmetric TSPs this requires storing n(n-1)/2 distinct pheromone values, because we assume that $\tau_{ij} = \tau_{ji} \forall(i,j)$.In the case for the distance matrix, it is more convenient to use some redundancy and to store the pheromones in a symmetric n^2 matrix.
- **Combining Pheromone and Heuristic Information:** When constructing a tour, an ant located on city i chooses the next city j with a probability which is proportional to the value of $\tau_{il}^{\alpha} \cdot \eta_{il}^{\beta}$. Because these very same values need to be computed by each of the m ants, computation times may be significantly reduced.
- **Representing Ants:** An ant is a simple computational agent which constructs a solution to the problem at hand, and may deposit an amount of pheromone τ_0 on the arcs it has traversed. To do so, an ant must be able to (1) store the partial solution it has constructed so far, (2) determine the feasible neighborhood at each city, and (3) compute and store the objective function value of the solutions it generates. The knowledge of the partial tour at each step is sufficient to allow the ant to determine whether a city j is in its feasible neighborhood: it is enough to scan the partial tour for the occurrence of city j. If city j has not been visited yet, then it is member of the feasible neighborhood; otherwise it is not. Unfortunately, this simple way of determining the feasible neighborhood involves an operation of worst-case complexity O(n) for each city i, resulting in a high computational overhead.

For each sTSP problem, this system performed maximum evaluations number (generated tours) given by the following formula [3]:

evaluations =100 * problem_size * problems_type (3.5)

where problems_type = 100 for sTSP .In Ant Colony System, the maximum evaluations number base on the person count.

evaluations = evaluations/ numbers of person (3.6)

where numbers of person (m) =10 according to ACS good parameters.

3.6 Nearest Neighbor Heuristic Algorithm

Nearest neighbor heuristic is called the greedy heuristic algorithm [12]. The greedy heuristic gradually constructs a tour by repeatedly selecting the shortest edge and adding it to the tour as long as it doesn't create a cycle with less than N edges, or increases the degree of any node to more than 2. We must not add the same edge twice of course. Complexity of the greedy heuristic is O ($n^2 \text{Log}_2(n)$). Steps of Greedy approach are:

- Sort all edges.
- Select the shortest edge and add it to our tour if it doesn't violate any of the above constraints.
- Do we have N edges in our tour? If no, go to step 2.

3.7 Case Study of the System

- Calculating the distance between two cities
  - Example for Ulysses 22 datasets using equation 3.4
  - $d_{1,2} = \sqrt{(39.57 - 38.24)^2 + (26.15 - 20.42)^2} = 5.882$
  - $d_{1,3} = \sqrt{(40.56 - 38.24)^2 + (25.32 - 20.42)^2} = 5.421$
  - $d_{1,4} = \sqrt{(36.26 - 38.24)^2 + (23.12 - 20.42)^2} = 3.348$

- $d_{1,5} = \sqrt{(33.48 - 38.24)^2 + (10.54 - 20.42)^2} = 10.967$
- $d_{1,6} = \sqrt{(37.56 - 38.24)^2 + (12.19 - 20.42)^2} = 8.258$
- $d_{1,7} = \sqrt{(38.42 - 38.24)^2 + (13.11 - 20.42)^2} = 7.321$

- Calculating pheromone value (τ_0) between two cities
  - Example for city 1 in Ulysses 22 datasets using greedy heuristic
  - $l_{1,1} = d_{1,8} + d_{8,16} + d_{16,22} + d_{22,4} + d_{4,18} + d_{18,17} + d_{17,2} + d_{2,3} + d_{3,13} + d_{13,12} + d_{12,14} + d_{14,15} + d_{15,6} + d_{6,7} + d_{7,19} + d_{19,20} + d_{20,21} + d_{21,10} + d_{10,9} + d_{9,5} + d_{5,11} + d_{11,1} =$ 0.720 + 2.040+ 3.493+ 1.425+ 0.208+ 2.419+ 2.323+ 1.292+ 10.257+ 0.388+ 0.961+ 2.192+ 2.970+ 1.259+ 4.897+ 0.590+ 0.089+ 1.426+ 3.950+ 7.883+ 15.963+ 25.72 = 89.641
  - $\tau_0 = (22 * 89.641)^{-1}$ =0.0005070732

- Choosing the next city for each person
  - When the parameter values of q = 0.1, q_0 = 0.9 and α = 0.1, example for Ulysses 22 datasets using equation 3.1
  - next city for person 1 = $argmax_{8 \in J_1(1)}$ {0.00045781420 * $(1/0.720)^2$} = 0.0008831292
  - next city for person 2 = $argmax_{16 \in J_1(1)}$ {0.00045781420 $(1/1.412)^2$} = 0.0002296254
  - next city for person 3 = $argmax_{22 \in J_1(1)}$ {0.00045781420 $(1/2.242)^2$} = 0.000091079
  - next city for person 7 = $argmax_{13 \in J_1(1)}$ {0.00045781420 * $(1/5.071)^2$} = 0.0000178034

- Updating the pheromone value after choosing the next city
  - Using equation 3.2, the local pheromone value update for person no: =1
  - $\tau(1,8) \leftarrow (1 - 0.1) * 0.00045781420 + 0.1 * 0.00045781420 = 0.00045781420$
  - $\tau(8,16) \leftarrow (1 - 0.1) * 0.00045781420 + 0.1 * 0.00045781420 = 0.00045781420$
  - $\tau(16,22) \leftarrow (1 - 0.1) * 0.00045781420 + 0.1 * 0.00045781420 = 0.00045781420$
  - $\tau(22,4) \leftarrow (1 - 0.1) * 0.00045781420 + 0.1 * 0.00045781420 = 0.00045781420$
  - $\tau(4,18) \leftarrow (1 - 0.1) * 0.00045781420 + 0.1 * 0.00045781420 = 0.00045781420$
- Updating the pheromone value for person who gets global best tour
  - Example for Ulysses 22 datasets, person no: = 3 is person who finds the global best tour
  - Using equation 3.3
  - $\Delta\tau(r,s) = (L_{gb})^{-1} = (87.542)^{-1} = 0.0114230883$
  - $\tau(1,22) \leftarrow (1 - 0.1) * 0.00045781420 + 0.1 * 0.0114230833 = 0.0015543416$
  - $\tau(22,4) \leftarrow (1 - 0.1) * 0.00045781420 + 0.1 * 0.0114230833 = 0.0015543416$
  - $\tau(4,18) \leftarrow (1 - 0.1) * 0.00045781420 + 0.1 * 0.0114230833 = 0.0015543416$
  - $\tau(18,17) \leftarrow (1 - 0.1) * 0.00045781420 + 0.1 * 0.0114230833 = 0.0015543416$
  - $\tau(17,2) \leftarrow (1 - 0.1) * 0.00045781420 + 0.1 * 0.0114230833 = 0.0015543416$

- Calculating maximum evaluation numbers for sTSP datasets
  - Using equation 3.5
  - the maximum evaluations numbers of Burma (14) = 140,000
  - the maximum evaluations numbers of Ulysses (16) = 160,000
  - the maximum evaluations numbers of Ulysses (22) = 220,000
  - the maximum evaluations numbers of Eli (51) = 510,000
- Calculating maximum evaluation numbers for sTSP datasets on ACS
  - Using equation 3.6, example for sTSP datasets when the numbers of person count = 10
  - the maximum evaluations numbers of Burma (14) = 14,000
  - the maximum evaluations numbers of Ulysses (16) = 16,000
  - the maximum evaluations numbers of Ulysses (22) = 22,000
  - the maximum evaluations numbers of Eli (51) = 51,000

CHAPTER 4

DESIGN AND IMPLEMENTATION

4.1 System Design

"Solving Symmetric Traveling Salesman Problem using Ant Colony System Algorithm" is implemented by applying ant colony system algorithm with the help of C# Programming language.

In this work, the symmetric TSP datasets from the TSPLIB datasets are used to compute the performance of ant colony system. User-defined parameters are as follows:

- Number of persons,
- maximum numbers of program iterations or maximum numbers of tour constructions,
- α which weigh the relative importance of heuristic visibility,
- β which weigh the relative importance of pheromone trail and of closeness,
- q_0 and q which values chosen randomly with uniform probability in (0,1)

This system also generates τ_0 using nearest neighbor heuristic and then the shortest possible route will be found by computing ant colony system algorithm. Finally, the system provides user-friendly information as the result.

4.1.1 System Flow

The preprocessing steps of the system are calculation distance between cities by using Euclidean distance formula, the initial pheromone value by using greedy heuristic method, the numbers of person and the

ACS algorithm parameters. In ACS algorithm, this system must initialize the start city for all persons.

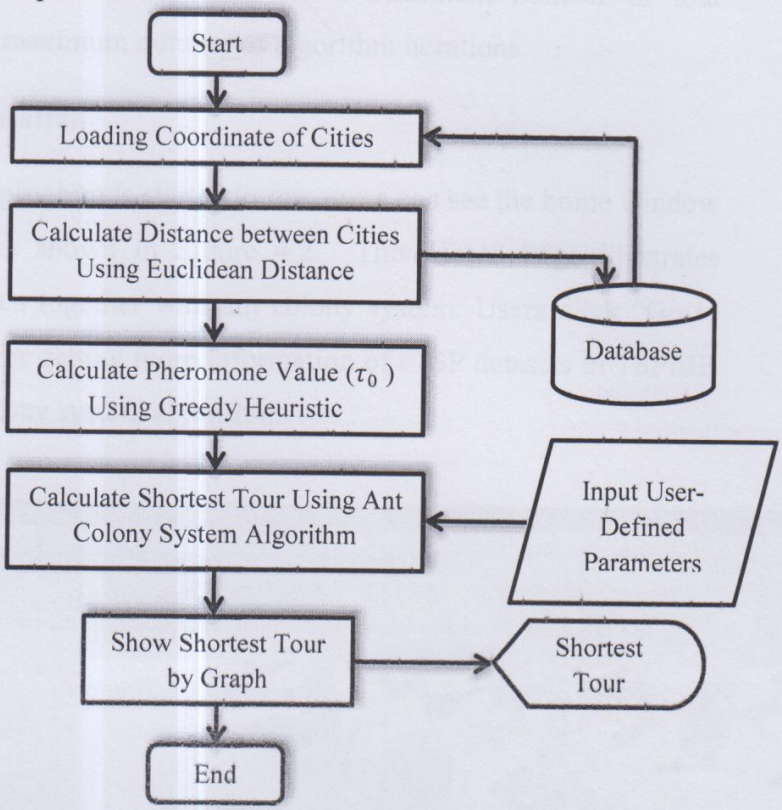

Figure 4.1 System Flow

This system uses the city 1 as start city. If the start city for all persons is different, the optimal result will be changed. All persons have to move to next city by applying State transition rule or random proportional rule depending on parameter value of. Local updating rule to update pheromone value after each person had found the next city in time. Each person visits all cities and then moves to the start city. Finally, this system calculates the total distance for all persons. When all persons have constructed their tour, the global updating rule has been used to update all pheromone value in the shortest tour. If the termination condition is true, the system generates the output result of the ant colony system algorithm. This system constructs tour for all persons. This system

shows the optimal result with graph. In this system, only one termination condition must be used by means of a maximum number of tour constructions or maximum number of algorithm iterations.

4.2 Implementation

When the program is started to run, users can see the home window of the system as shown in Figure 4.2. This Home Page illustrates traveling salesmen together with ant colony system. Users click "Go to System" button for getting more information of sTSP datasets in TSPLIB and about ant colony system algorithm.

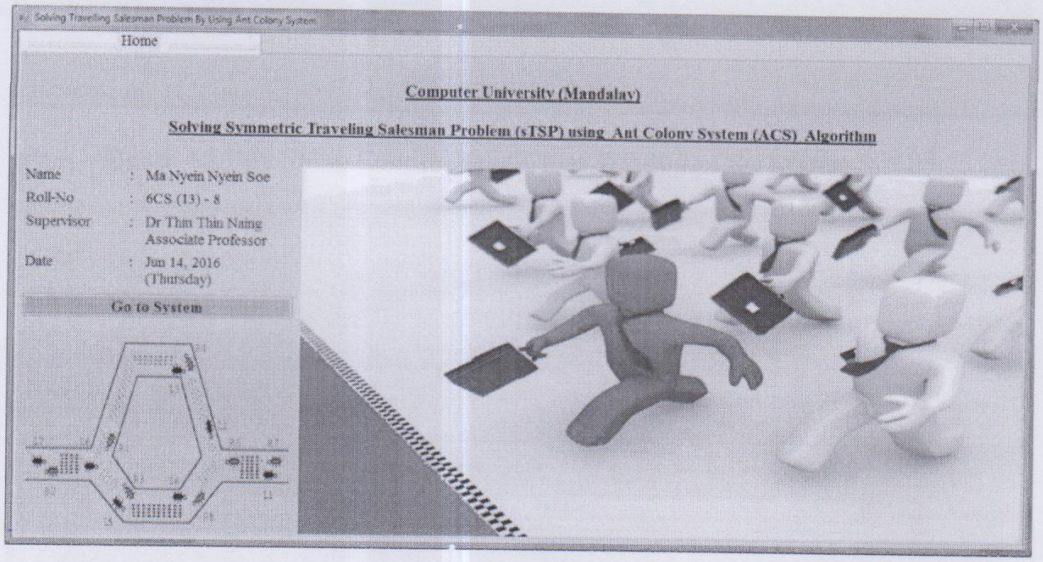

Figure 4.2 Home Window of the System

In figure 4.3, users can choose any sTSP datasets in this system database by pressing the combo box of "Datasets Name" such as Burma 14, Ulysses 16, Ulysses 22, Eli 51, Pr 76 and so on. This form shows detailed information of sTSP datasets that are chosen by user. These datasets are coordinates of cities dataset. In this system, this dataset is applied on ant colony system algorithm. This form contains one button

that is "Calculation Distance between Cities". This button works to find the distance between cities.

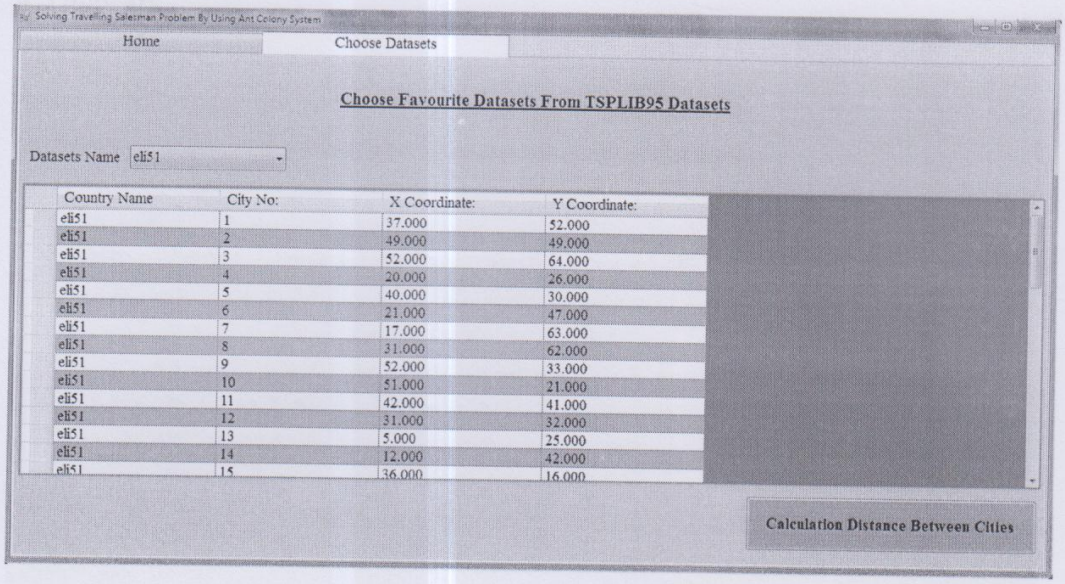

Figure 4.3 Describing Coordinate of Cities Window of the System

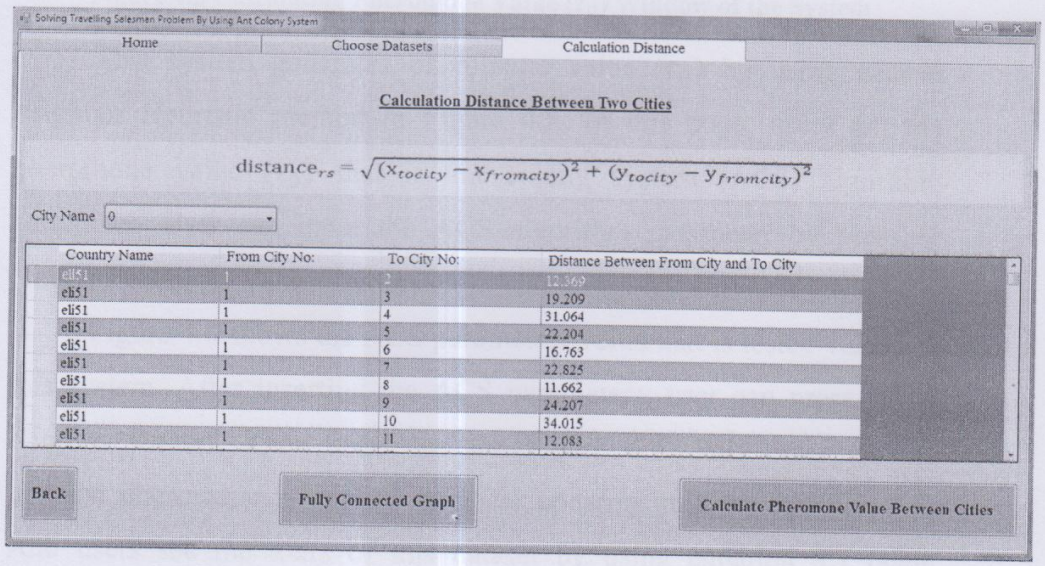

Figure 4.4 Distances between Cities Window of the System

Figure 4.4 shows the distance between cities by using euclidean distance formula. Users can test each city number by pressing "City Name" combo box. This form contains three buttons. "Fully Connected

Graph" button describes the fully connected graph of each dataset. When users click "Calculate Pheromone Value Between Cities" button, Figure 4.5 appears.

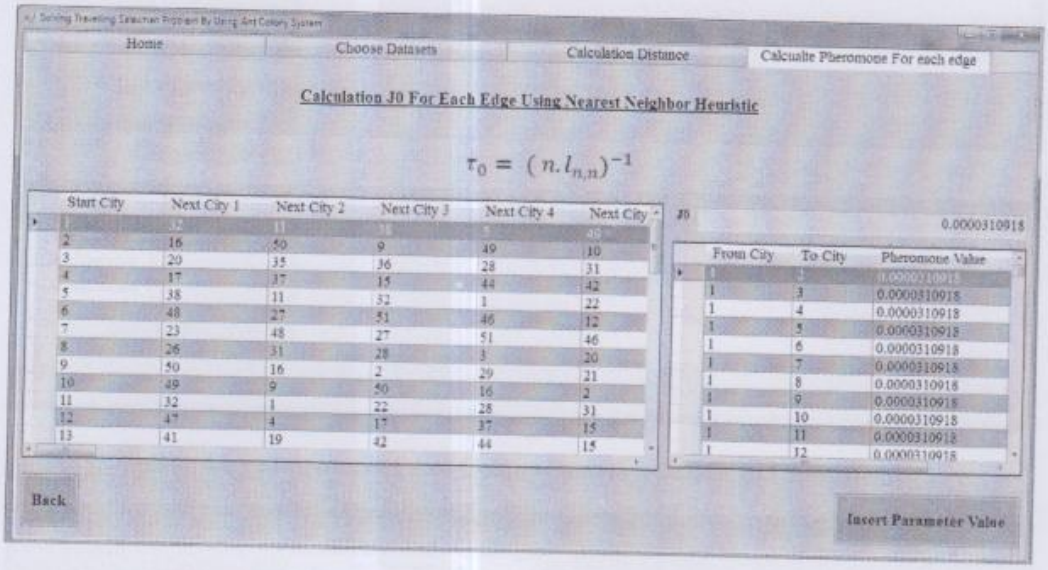

Figure 4.5 Calculating Pheromone Value (τ_0) Window of the System

This system generates pheromone value (τ_0) by using nearest neighbor Heuristic method in Figure 4.5. In this page, users get the pheromone value (τ_0) that is ant colony system (ACS) algorithm parameter. User can input the ACS algorithm parameter by pressing "Insert Parameter Value" button.

Figure 4.6 shows the ACS parameters value that is tested values of this system. After inserting the ACS parameters, user can press "Find Shortest Route". Then users see the Figure 4.7. This describes the updated pheromone is applied on local updating rule on equation 3.2. And users see the tours of this dataset by using equation 3.1 (state transition rule) in Figure 4.8. When users click "Global Step" button, they see the updated pheromone value for shortest tour depending on equation 3.3 (global updating rule) as shown in Figure 4.9. Finally, users click the "Shortest Tour" button to see Figure 4.10.

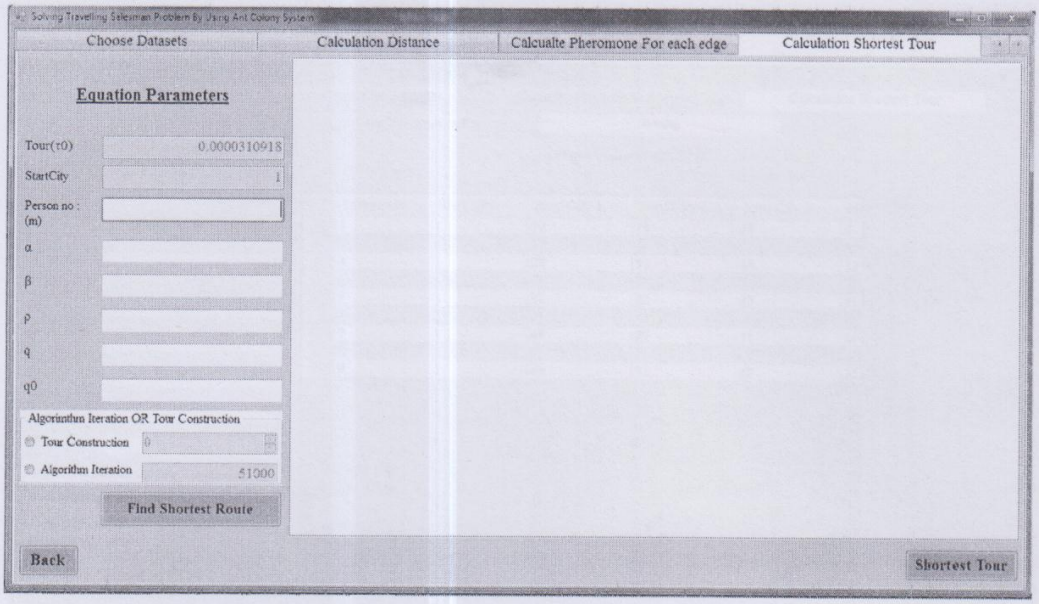

Figure 4.6 Parameters of Ant Colony System Window of the System

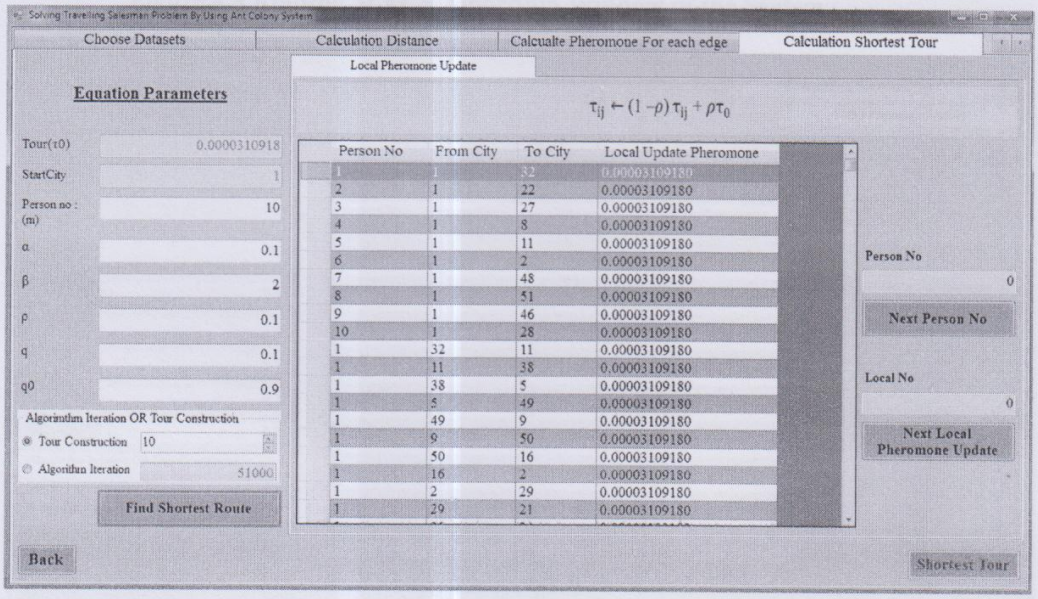

Figure 4.7 Local Pheromone Update of ACS Algorithm Window of the System

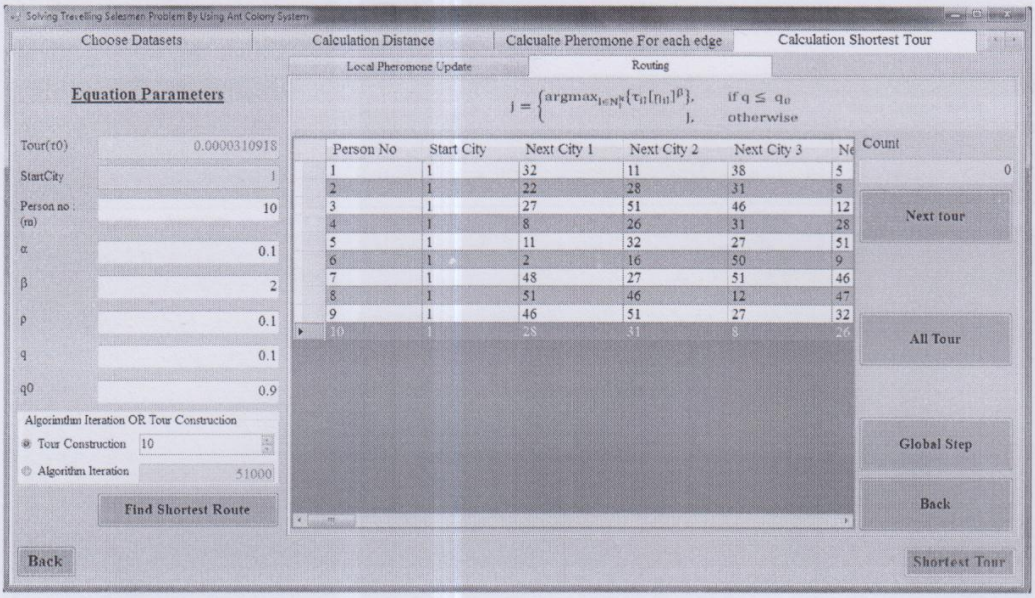

Figure 4.8 Routing of ACS Algorithm Window of the System

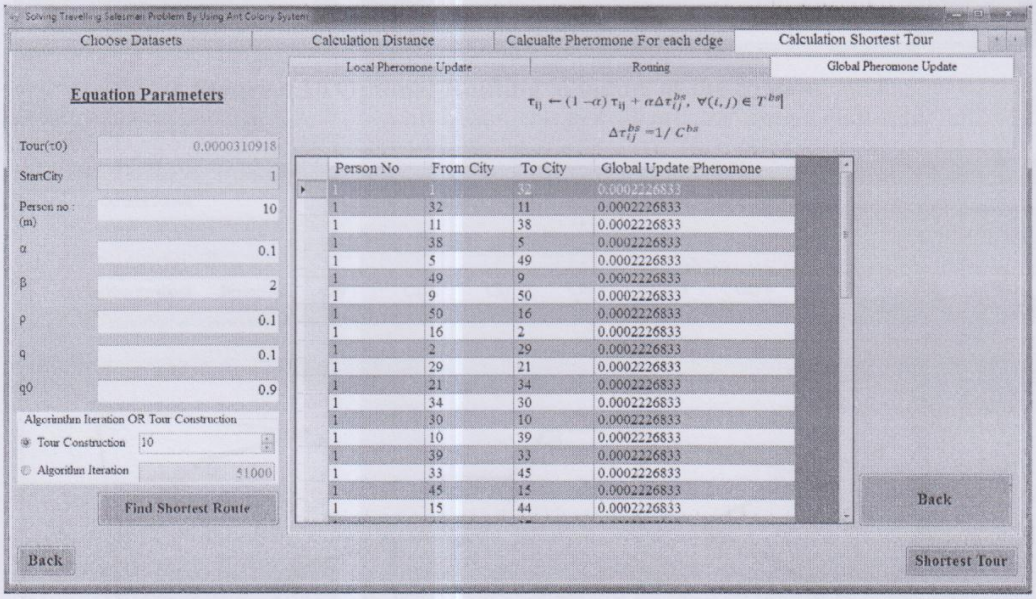

Figure 4.9 Global Pheromone Update of ACS Algorithm Window of the System

Figure 4.10 shows the shortest tour of this dataset which depends on algorithm parameters. This includes three buttons. "Shortest Tour by

Graph" button is to describe the optimal result with graph on panel as shown in Figure 4.11.

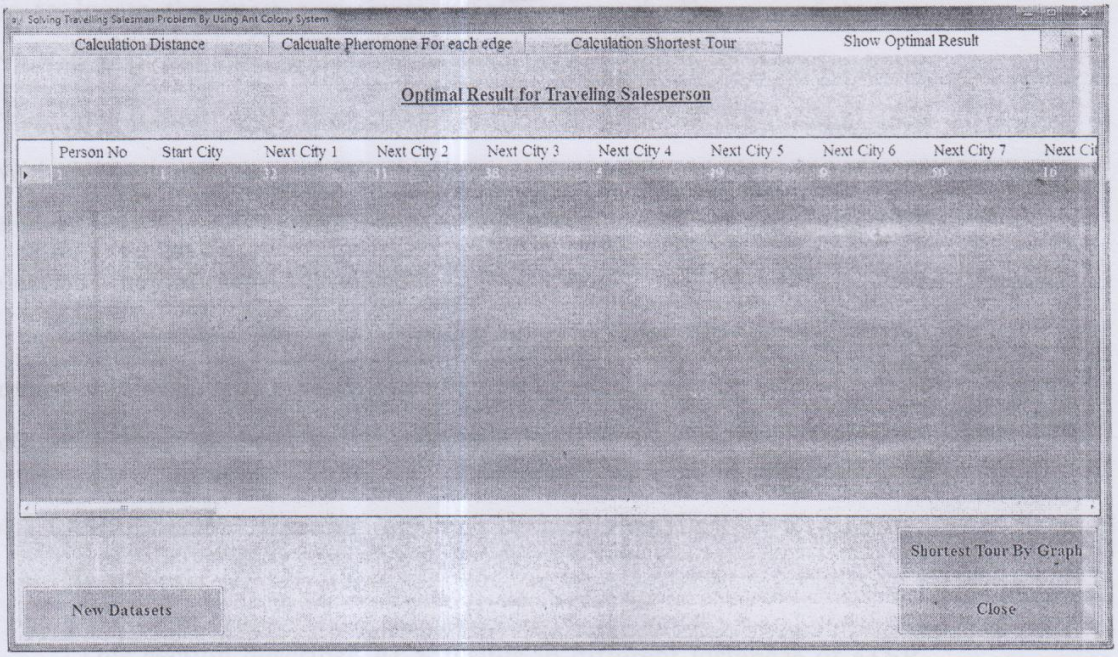

Figure 4.10 Optimal Routing Window of the System

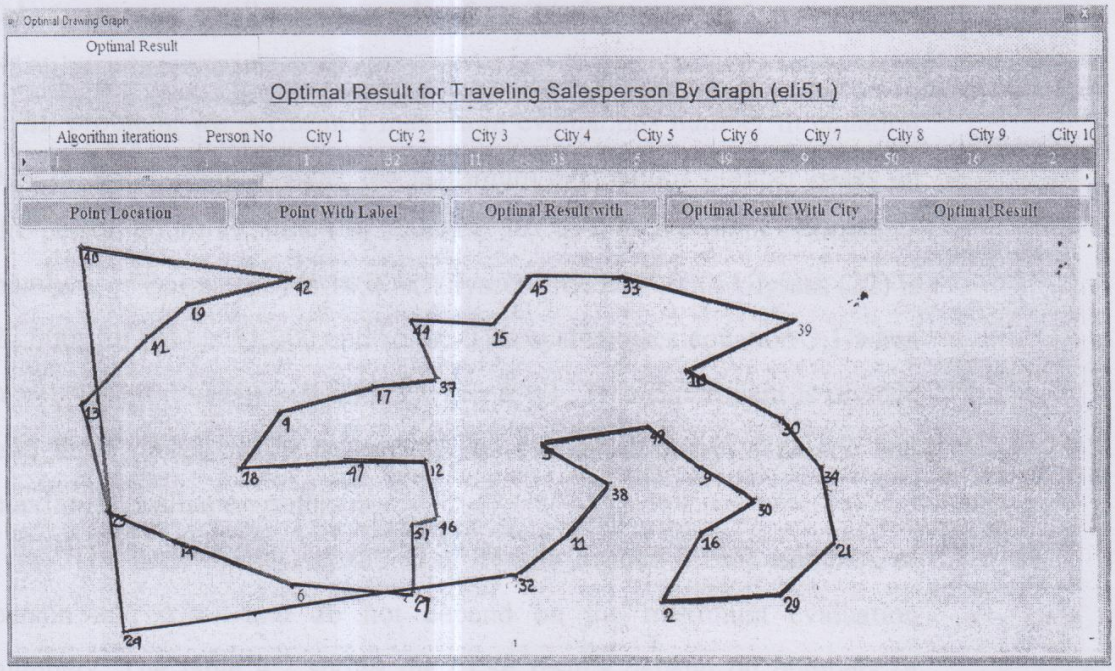

Figure 4.11 Shows Optimal Result with Graph Window of the System (eli 51)

. In Figure 4.11, this form contains five buttons to show optimal result with graph and datatable. The first button, "Point Location" button is to locate the (x,y) coordinate points on panel. The second button, "Point With Label" button is to describe these points with their location. "Optimal Result With Coordinate" button is to show the optimal tour with (x,y) coordinate and "Optimal Result With City No:" is to indicate the optimal tour with city number. Finally, "Optimal Result" button is to describe the optimal result clearly.

To calculate this system with new datasets, user clicks "New datasets button. If not, user can click "Close" button. Then the system closes. In this system, the "Back" button refers to the pervious page.

4.2.1 Comparison Results on m, α and Maximum Evaluation Numbers

This section describes ACS results in different instances of sTSP. The ACS parameters in the following experiment are $\beta = 2$, $q = 0.1$, $\rho = 0.1$, τ_0 =a very small constant value from $1/nC_{nn}$, $q_0 = 0.9$, n = number of cities where numbers of cities in dataset Burma (14) is 14. For each sTSP problem, we performed maximum evaluations number in equation 3.5. In Ant Colony System, the maximum evaluations number bases on the person count as shown in equation 3.6.So the maximum evaluations numbers of Burma (14) is 14,000; Ulysses (16) is 16,000; Ulysses (22) is 22,000; Eli (51) is 51,000 and so on. This system uses equation 3.1 when $\alpha = 0.1$, and it uses equation 2.3 when $\alpha = 1$. This experiment is divided into three tables by the parameter value of α, numbers of person and maximum evaluation numbers.

The table 4.1 shows the sTSP results depend on the numbers of person and $\alpha = 1$ and do not depend on the maximum evaluation numbers. The table 4.2 shows the sTSP results depend on numbers of

person and $\alpha = 0.1$ and do not depend on the maximum evaluation numbers. The table 4.3 shows the sTSP results depend on numbers of person =10, $\alpha = 0.1$ and maximum evaluation numbers on each datasets.

In table 4.1 and 4.2 reported the results obtained for sTSP problems depend on ACS parameter. The first column reports the datasets name and number of cities (in parentheses). The second, third and fourth columns report the best result depending on the number of persons.

Table 4.1 sTSP Problems on $\alpha = 1$ for Four Datasets

Problem (cities)	Best Result [#no: of person=3]	Best Result [#no: of person= 5]	Best Result [#no of person=7]
Burma (14)	45.047	42.487	42.487
Ulysses (16)	103.607	90.907	90.907
Ulysses (22)	132.487	118.973	118.973
Eli (51)	1367.346	1365.111	1357.885

Table 4.2 sTSP Problems on α =0.1 for Four Datasets

Problem (cities)	Best Result [#no: of person= 3]	Best Result [#no: of person= 5]	Best Result [#no of person=7]
Burma (14)	38.688	37.001	37.001
Ulysses (16)	86.467	86.487	84.619
Ulysses (22)	87.542	87.542	87.542
Eli (51)	513.609	513.609	513.609

In table 4.1 and 4.2, the numbers of tour generate, depending on the numbers of person. The tour for each person is different. For instance, the numbers of person =3 in Ulysses (22), the tours of person 1, 2 and 3 are different. In these, the tour of person 3 is shortest. Therefore, we describe the shortest tour length of person 3 in table 2. When we increase the person count to 5 and 7, the tour of person 3 is still shortest. Therefore, the tour length of Ulysses (22) is the same in table 4.2.

Table 4.3 reported the optimal results for sTSP instance depend on ACS parameter. The first column reports the datasets name, number of cities (in parentheses) and maximum evaluations numbers (in square brackets). The second column reports the optimal result for each dataset and the numbers of evaluation required to find it (in square brackets).

Table 4.3 sTSP Problems on No: of Persons = 10 and α =0.1 for Four Datasets

Problem (cities) [#evaluations]	Optimal Result
Burma (14) [14,000]	36.332 [307]
Ulysses (16) [16,000]	74.108 [291]
Ulysses (22) [22,000]	75.665 [477]
Eli (51) [51,000]	426 [1091]

In this system, the evaluations number = 307 and above gets 36.332 for Burma 14. Therefore, the optimal result for Burma (14) is 36.332. The evaluations number = 291 and above gets 74.108 for Ulysses (16). The optimal result for Ulysses (16) is 74.108.The evaluations number = 477 and above gets 75.665 for Ulysses (22). The optimal result for Ulysses (22) is 75.665. The evaluations number = 1091 and above gets 426 for Eli (51). The optimal result for Eli (51) is 426. Therefore, this system gets the optimal result without needing to calculate the

maximum evaluation numbers on each dataset. These tables show only with respect to the optimal result or best result. In these, table 4.2 shows the system best result for each dataset dependss on numbers of person. The table 4.3 shows the system optimal result for each dataset.

CHAPTER 5

CONCLUSION AND FUTURE WORK

5.1 Conclusion

The goal of the system is to make the comprehensive study on ant colony system algorithm for traveling salesman on various datasets such as Burma (14), Ulysses (16), Ulysses (22), eli (51) and so on. This system must be convenient for real marketing persons who apply with real datasets in the real world. Ant colony system also seems to minimize the traveling distance, in order to improve comprehensibility of the traveling salesman problem. This system has ability to find the shortest possible route for sTSP datasets. sTSP datasets in TSPLIB [14] contain over 100 datasets with sizes from 14 cities up to 85,900 cities. To conclude, this system describes the highest achievable results under the given parameters. Especially, the ACS algorithm parameters $\alpha = 0.1$ and numbers of persons = 10 provide the optimal result for sTSP.

5.2 Advantage and Limitation of the System

The advantage of the system is finding the shortest tour without needing to calculate the maximum evaluation numbers on each dataset. If numbers of cities are less than 3, this system cannot be used. The start city of all datasets in this system must initialize 1. This system can use only sTSP datasets in TSPLIB [14].

5.3 Future Work

- This system can be tested on sTSP datasets over 100 datasets with sizes from 14 cities to 85,900 cities in TSPLIB datasets.
- Moreover, this system can be extended for solving on random datasets with (x,y) coordinates and finding the shortest tour

between streets or townships of the city based on real data points.

- In future, another system may be developed for asymmetric traveling salesman problem (aTSP) based on this system.

REFERENCES

[1] Marco Dorigo and Thomas Stützle., "Ant Colony Optimization", MIT Press, Cambridge, 2004

[2] Manu Goyal, "Modified Ant Colony Optimization Algorithm for Traveling Salesman Problem", June 12, Thapar University

[3] Marco Dorigo and Luca Maria Gambardella and, "Solving Symmetric and Asymmetric TSPs by Ant Colonies", 1996

[4] Marco Dorigo and Luca Maria Gambardella, "Ant Colony System: A Cooperative Learning Approach to the Traveling Salesman Problem", 1996

[5] Marco Dorigo and Luca Maria Gambardella, "Ant Colonies for the Traveling Salesman Problem", 1996-3

[6] Marco Dorigo and Krzysztof Socha,"An introduction to Ant Colony Optimization"

[7] Masoud Yaghini, "Ant Colony Optimization, Part 3: Algorithms", 2009

[8] Nadia Abd-Alsabour and Atef Moneim," Applying Ant Colony Optimization to Optimization Problems", International Journal of Research in Engineering and Technology (IJRET) Vol.1,No.6,2012

[9] Nuno Abreu, Muhammad Ajmal, Zafeiris Kokkinogenis and Behdad Bozorg, "Ant Conony Optimization", 2011-01-17

[10] Nitish Sabharwal, Harshit Sharma, "A Recursive Ant Colony System Algorithm for the TSP", 2011 International Conference on Advancements in information Technology With Workshop of ICBMG 2011 IPCSIT vol.20 (2011)

[11] Professor Stein, Sally Kim and James Tsai, "Traveling Salesman Problem", April 30, 2009

[12] Rajesh Matai, Surya Prakash Singh and Murari Lal Mittal, "Traveling Salesman Problem: An Overview of Applications, Formulations, and Solution Approaches"

[13] Zar Chi Su Su Hlaing and May Aye Khine, "Solving Traveling Salesman Problem by Using Improved Ant Colony Optimization Algorithm", International Journal of Information and Education Techonology

[14] http://www.iwr.uni-heidelberg.de/iwr/comopt/soft/TSPLIB95/TSPLIB