# Modified Ant Colony Optimization Algorithm for Traveling Salesman Problem

*Thesis submitted in partial fulfillment of the requirements of the award of degree of*

**Master of Technology**

*in*

**Computer Science and Applications**

*Submitted By*

**Manu Goyal**

(Roll No. 601003012)

*Under the supervision of*

**Mrs. Maninder Kaur**

School of Mathematics and Computer Applications
Thapar University
Patiala – 147004
June 2012

## Certificate

I hereby certify that the work which is being presented in the thesis entitled, "*Modified Ant Colony Optimization Algorithm for Traveling Salesman Problem*", in partial fulfillment of the requirements for the award of degree of Master of Technology in *Computer Science and Applications* submitted in School of Mathematics and Computer Applications of Thapar University, Patiala, is an authentic record of my own work carried out under the supervision of *Mrs Maninder Kaur* and refers other researcher's work which are duly listed in the reference section.

The matter presented in the thesis has not been submitted for award of any other degree of this or any other University.

Signature:

(Manu Goyal)

This is to certify that the above statement made by the candidate is correct and true to the best of my knowledge.

(*Maninder Kaur*)

Assistant Professor

SMCA

**Countersigned by**

**(Dr. S.S.Bhatia)**
Head
School of Mathematics and Computer Applications
Thapar University
Patiala

**(Dr. S. K. Mohapatra)**
Dean (Academic Affairs)
Thapar University
Patiala

## ACKNOWLEDGEMENT

I wish to express my sincere thanks and deep sense of gratitude to my teacher and guide Mrs Maninder Kaur, Lecturer, School of Mathematics and Computer Applications (*SMCA*), Thapar University, Patiala, Punjab, for his constant inspiration, scholarly guidance and helpful suggestion throughout the course of my thesis work.

I am very thankful to Dr. S.S.Bhatia and all other faculty members of *SMCA* for their intellectual support throughout the course. I am also thankful to all staff members of *SMCA* for their kind cooperation and sincere help. My special thanks are due to family members and friends who constantly encouraged me to complete my thesis work.

**Manu Goyal**
Roll No. 601003012
M.Tech (CSA)

# Abstract

Ant colony optimization is a technique for optimization that was introduced in the early 1990's. The inspiring source of ant colony optimization is the foraging behavior of real ant colonies. This behavior is exploited in artificial ant colonies for the search of approximate solutions to discrete optimization problems, to continuous optimization problems, and to important problems in telecommunications, such as routing and load balancing. First, we study with the biological inspiration of ant colony optimization algorithms and how this biological inspiration can be transferred into an algorithm for Traveling Salesman Problem (TSP). Then, ant colony optimization outlined in more general terms in the context of discrete optimization, and some of the nowadays best performing ant colony optimization variants are studied. This research approach lies at initial stage at present, and a new modified ant algorithm is proposed for the traditional ant algorithm easily appears precocious and stagnation behavior phenomenon in this paper. And the various parameter of ant colony algorithm is adjusted. Selecting a typical TSP instance to experiment, the results are indicated that the new modified ant colony algorithm has a better ability to search the global optimal solution and have better stability.

# Table of contents

# List of Abbreviations

| | Abbr. | Details |
|---|---|---|
| 1 | AS | Ant System |
| 2 | ACO | Ant Colony Optimization |
| 3 | TSP | Traveling Salesman Problem |
| 4 | $AS_{rank}$ | Rank Based Ant System |
| 5 | MMAS | Max-Min Ant System |
| 6 | ACS | Ant Colony System |
| 7 | $ACO_{nn}$ | Ant Colony Algorithm with Nearest Neighborhood List |

# List of Figures

# 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 behavior 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, Ant Colony Optimization (ACO) was introduced by M.Dorigo and colleagues as a novel nature-inspired metaheuristic for the solution of hard combinatorial optimization (CO) problems. ACO belongs to the class of metaheuristics [3], which are approximate algorithms used to obtain good enough solutions to hard CO problems in a reasonable amount of computation time. Ant Colony Optimization (ACO) is a recently proposed metaheuristic approach for solving hard combinatorial optimization problems. The inspiring source of ACO is the pheromone trail laying and following behavior of real ants, which use pheromones as a communication medium. In analogy to the biological example, ACO is based on the indirect communication of a colony of simple agents, called (artificial) ants, mediated by (artificial) pheromone trails. The pheromone trails in ACO serve as distributed, numerical information, which the ants use to probabilistically construct solutions to the problem being solved, and which the ants adapt during the algorithm's execution to reflect their search experience.

The first example of such an algorithm is Ant System (AS), which was proposed using as an example application the well-known Traveling Salesman Problem (TSP). Despite encouraging initial results, AS could not compete with state-of-the-art algorithms for the TSP. Nevertheless, it had the important role of stimulating further research on algorithmic variants, which obtain much better computational performance, as well as on applications to a large variety of different problems. In fact, now there exists a considerable amount of

applications obtaining world class performance on problems like the quadratic assignment, vehicle routing, sequential ordering, scheduling, routing in Internet-like networks, and so on.

The (artificial) ants in ACO implement a randomized construction heuristic, which makes probabilistic decisions as a function of artificial pheromone trails and possibly available heuristic information based on the input data of the problem to be solved. As such, ACO can be interpreted as an extension of traditional construction heuristics, which are readily available for many combinatorial optimization problems.

Yet, an important difference with construction heuristics is the adaptation of the pheromone trails during algorithm execution to take into account the cumulated search experience.

## 1.1 The Origin of Ant Colony Optimization

Marco Dorigo and colleagues introduced the first ACO algorithms in the early 1990's [1, 2]. 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 ACO 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.

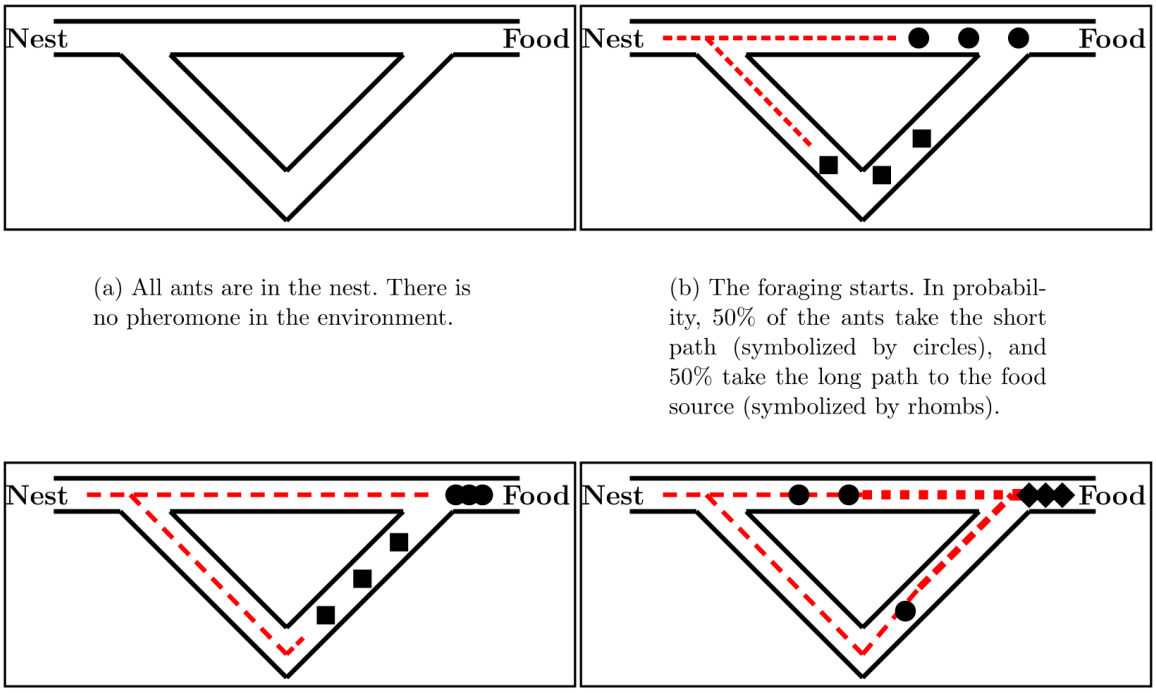
As a first step towards an algorithm for discrete optimization is presented in the following a discretized and simplified model [5] of the phenomenon explained in Figure 1.1. After presenting the model, the differences between the model and the behavior of real ants will be outlined. Model consists of a graph $G = (V, E)$, where $V$ consists of two nodes, namely

$v_s$ (representing the nest of the ants), and $v_d$ (representing the food source). Furthermore, $E$ consists of two links, namely $e_1$ and $e_2$, between $v_s$ and $v_d$. To $e_1$ a length of $l_1$, and to $e_2$ a length of $l_2$ such that $l_2 > l_1$ is assigned. In other words, $e_1$ represents the short path between $v_s$ and $v_d$ and $e_2$ represents the long path. Real ants deposit pheromone on the paths on which they move. Thus, the chemical pheromone trails are modelled as follows.

An artificial pheromone value $\tau_i$ introduced for each of the two links $e_i$ , $i$ = 1, 2. Such a value indicates the strength of the pheromone trail on the corresponding path. Finally, $n_a$ artificial ants are introduced. Each ant behaves as follows: Starting from $v_s$ (i.e., the nest), an ant chooses with probability

$$p_i = \frac{\tau_i}{\tau_1+\tau_2}, \qquad i= 1, 2 \qquad (1.1)$$

between path $e_1$ and path $e_2$ for reaching the food source $v_d$ . Obviously, if $\tau_1 > \tau_2$, the probability of choosing $e_1$ is higher, and vice versa.




(a) All ants are in the nest. There is no pheromone in the environment.

(b) The foraging starts. In probability, 50% of the ants take the short path (symbolized by circles), and 50% take the long path to the food source (symbolized by rhombs).

(c) The ants that have taken the short path have arrived earlier at the food source. Therefore, when returning, the probability to take again the short path is higher.

(d) The pheromone trail on the short path receives, in probability, a stronger reinforcement, and the probability to take this path grows. Finally, due to the evaporation of the pheromone on the long path, the whole colony will, in probability, use the short path.

**Figure 1.1: A double bridge experiment setting that demonstrates the shortest path finding capability of ant colonies. Between the ants' nest and the only food source**

**exist two paths of different lengths. In the four graphics, the pheromone trails are shown as dashed lines whose thickness indicates the trails' strength.**

For returning from $v_d$ to $v_s$ , an ant uses the same path as it chose to reach $v_d$ and it changes the artificial pheromone value associated to the used edge. More in detail, having chosen edge $e_i$ an ant changes the artificial pheromone value $\tau_i$ as follows:

$$\tau_i \leftarrow \tau_i + \frac{Q}{l_i}, \qquad (1.2)$$

where the positive constant $Q$ is a parameter of the model. In other words, the amount of artificial pheromone that is added depends on the length of the chosen path: the shorter the path, the higher the amount of added pheromone. The foraging of an ant colony is in this model iteratively simulated as follows: At each step (or iteration) all the ants are initially placed in node $v_s$ .Then, each ant moves from $v_s$ to $v_d$ as outlined above. In nature the deposited pheromone is subject to evaporation over time. This pheromone evaporation in the artificial model is simulating as follows:

$$\tau_i \leftarrow (1-\rho) \cdot \tau_i, \qquad i = 1, 2. \qquad (1.3)$$

The parameter $\rho \in (0, 1]$ is a parameter that regulates the pheromone evaporation. Finally, all ants conduct their return trip and reinforce their chosen path as outlined above.

## 1.2 Toward Artificial Ants

The double bridge experiments[5] show clearly that ant colonies have a built-in optimization capability by the use of probabilistic rules based on local information they can find the shortest path between two points in their environment. Interestingly, by taking inspiration from the double bridge experiments, it is possible to design artificial ants that, by moving on a graph modelling the double bridge, find the shortest path between the two nodes corresponding to the nest and to the food source.

Unfortunately, if the minimum cost path problem on the graph G is tried to solve using artificial ants whose behavior is a straightforward extension of the behavior of the ants described in the previous section, the following problem arises: the ants, while building a solution, may generate loops. As a consequence of the forward pheromone trail updating mechanism, loops tend to become more and more attractive and ants can get trapped in them. But even if an ant can escape such loops, the overall pheromone trail distribution becomes such that short paths are no longer favoured and the mechanism that in the simpler double bridge situation made the ant choose the shortest path with higher

probability does not work anymore. Because this problem is due to forward pheromone trail updating, it might seem that the simplest solution to this problem would be the removal of the forward updating mechanism: in this way ants would rely only on backward updating. If the forward update is removed the system does not work anymore, not even in the simple case of the double bridge experiment.

Therefore need to extend the capabilities of the artificial ants in a way that, while retaining the most important characteristics of real ants, allows them to solve minimum cost path problems on generic graphs.

- **Memory Allocation** In particular, artificial ants are given a limited form of memory in which they can store the partial paths they have followed so far, as well as the cost of the links they have traversed. Via the use of memory, the ants can implement a number of useful behaviors that allow them to efficiently build solutions to the minimum cost path problem.
- **Distance Estimation** The artificial ant can estimate the distance of the neighbourhood paths they have to traverse while searching the destination.
- **Quality of Solution** With this characteristic, the artificial ant can evaluate the cost of paths of given solution that they have traversed and also can compare with other ants solution.
- **Time and Quantity of Pheromone Deposition** Now ants can deposit pheromone only in backward mode and also ants can decide on the quantity of pheromone deposit.

With these capabilities the ants can (1) probabilistic solution construction biased by pheromone trails, without forward pheromone updating; (2) deterministic backward path with loop elimination and with pheromone updating; and (3) evaluation of the quality of the solutions generated and use of the solution quality in determining the quantity of pheromone to deposit (note that while in the simple case of minimum cost path search an estimate of the solution quality can be made by the ant also during the solution construction, this is not necessarily true in other problems, in which there may not exist an easy way to evaluate partial solutions).

Additionally, by taking into account pheromone evaporation is shown, which was not necessary to explain real ants' behavior, performance can be greatly improved.

In the following section briefly explain how the above-mentioned ants' behavior, as well as pheromone evaporation, is implemented in an algorithm that is called Simple-ACO (S-ACO for short). It should be noted that, although it represents a significant step toward the definition of an efficient algorithm for the solution of minimum cost problems on graphs, S-ACO should be taken for what it is: a didactic tool to explain the basic mechanisms underlying ACO algorithms.

## 1.3 ACO Metaheuristic

Informally, an ACO algorithm [5] can be imagined as the interplay of three procedures: ConstructAntsSolutions, UpdatePheromones and DaemonActions.

- **ConstructAntsSolutions** It manages a colony of ants that concurrently and asynchronously visit adjacent states of the considered problem by moving through neighbour nodes of the problem's construction graph GC. 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.

- **Update Pheromones** 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/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 favouring the exploration of new areas of the search space.
- **Daemon Actions** procedure 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.

**Procedure:** *ACO metaheuristic*
**ScheduleActivities**
ConstructAntSolutions ()
EvaporatePheromone ()
DaemonActions () {Optional}
**End ScheduleActivities**
**End** *ACO metaheuristic*

**Figure 1.2 ACO metaheuristic in pseudo-code.**

In Figure 1.2, the ACO metaheuristic is described in pseudo-code. The main procedure of the ACO metaheuristic manages the scheduling of the three above-discussed components of ACO algorithms via the ScheduleActivities construct: (1) management of the ants' activity, (2) pheromone updating, and (3) daemon actions. 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.
Nowadays numerous successful implementations of the ACO metaheuristic are available and they have been applied to many different combinatorial optimization problems.

# CHAPTER 2

# Literature Survey

Intuitively, the Traveling Salesman Problem [16] is the problem faced by a salesman who, starting from his home town, wants to find a shortest possible trip through a given set of customer cities, visiting each city once before finally returning home. The TSP can be represented by a complete weighted graph G =(N,A) with N being the set of n=|N| nodes (cities), A being the set of arcs fully connecting the nodes. Each arc (i,j) $\epsilon$ 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. There may distinguish between symmetric TSPs, where the distances between the cities are independent of the direction of traversing the arcs, that is $d_{ij} = d_{ji}$ for every pair of nodes, and the asymmetric TSP (ATSP), where at least for one pair of nodes i, j have $d_{ij} \neq d_{ji}$.

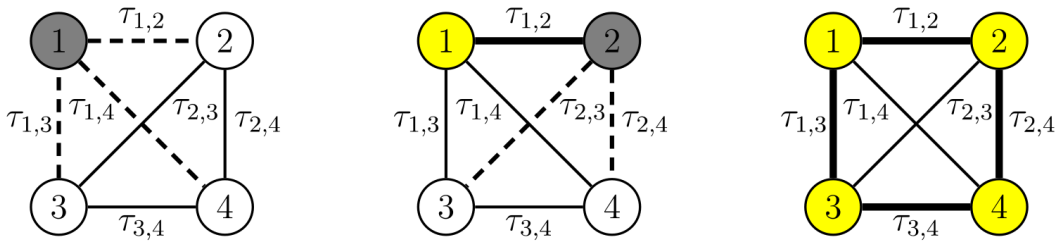
A solution to an instance of the TSP can be represented as a permutation of the city indices; this permutation is cyclic, that is, the absolute position of a city in a tour is not important at all but only the relative order is important (in other words, there are n permutations that map to the same solution).

- **Construction Graph** The construction graph is identical to the problem graph: the set of components C corresponds 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 if an ant at each construction step chooses the next city only among those it has not visited yet (i.e., the feasible neighborhood $N_i^k$ of an ant k in city i, where k is the ant's identifier, comprises all cities that are still unvisited).
- **Pheromone Trails and Heuristic Information** The pheromone trails $\tau_{ij}$ in the TSP refer to the desirability of visiting city j directly after i. The heuristic information $\eta_{jj}$ is typically inversely proportional to the distance between cities i and j, a straightforward choice being $\eta_{jj} = 1/d_{ij}$. In fact, this is also the heuristic information used in most ACO algorithms for the TSP.

- **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.
- **General Comments** The TSP is a paradigmatic NP-hard combinatorial optimization problem which has attracted a very significant amount of research. The TSP has played a central role in ACO, because it was the application problem chosen when proposing the first ACO algorithm called Ant System and it was used as a test problem for almost all ACO algorithms proposed later.

## 2.1 Ant System for TSP

Concerning the AS approach, the edges of the given TSP graph can be considered solution components, i.e., for each $e_{i,j}$ is introduced a pheromone value $\tau_{i,j}$ [6] . The task of each ant consists in the construction of a feasible TSP solution, i.e., a feasible tour. In other words, the notion of *task of an ant* changes from *"choosing a path from the nest to the food source"* to *"constructing a feasible solution to the tackled optimization problem"*. Note that with this change of task, the notions of nest and food source loose their meaning.




(a) First step of the solution construction.

(b) Second step of the solution construction.

(c) The complete solution after the final construction step.

**Figure 2.1: Example of the solution construction for a TSP problem consisting of 4 cities (modelled by a graph with 4 nodes; see Definition 1). The solution construction starts by randomly choosing a start node for the ant; in this case node 1. Figures (a) and (b) show the choices of the first, respectively the second, construction step. Note that in both cases the current node (i.e., location) of the ant is marked by dark gray color, and the already visited nodes are marked by light gray color (respectively yellow color, in the online version of this article). The choices of the ant (i.e., the**

**edges she may traverse) are marked by dashed lines. The probabilities for the different choices (according to Eq. (4)) are given underneath the graphics. Note that after the second construction step, in which exemplary assume the ant to have selected node 4, the ant can only move to node 3, and then back to node 1 in order to close the tour.**

Each ant constructs a solution as follows. First, one of the nodes of the TSP graph is randomly chosen as start node. Then, the ant builds a tour in the TSP graph by moving in each construction step from its current node (i.e., the city in which she is located) to another node which she has not visited yet. At each step the traversed edge is added to the solution under construction. When no unvisited nodes are left the ant closes the tour by moving from her current node to the node in which she started the solution construction. This way of constructing a solution implies that an ant has a memory $T$ to store the already visited nodes. Each solution construction step is performed as follows. Assuming the ant to be in node $v_i$ , the subsequent construction step is done with probability

$$p_{ij}^{k}(t) = \begin{cases} \dfrac{[\tau_{ij}(t)]^{\alpha}[\eta_{ij}]^{\beta}}{\sum_{k \in allowed_k}[\tau_{ik}(t)]^{\alpha}[\eta_{ik}]^{\beta}} & \text{if } j \in \text{allowed}_k \\ 0 & \text{otherwise} \end{cases} \qquad (2.1)$$

where $\eta_{ij} = 1/d_{ij}$ is a heuristic value that is available a priori, α and β are two parameters which determine the relative influence of the pheromone trail and the heuristic information, and $\text{allowed}_k(N_k^i)$ is the feasible neighborhood of ant k when being at city i, that is, the set of cities that ant k has not visited yet (the probability of choosing a city outside $N_k^i$ is 0). By this probabilistic rule, the probability of choosing a particular arc (i, j) increases with the value of the associated pheromone trail $\boldsymbol{\tau}_{ij}$ and of the heuristic information value $\eta_{ij}$. The role of the parameters α and β is the following. If α = 0, the closest cities are more likely to be selected: this corresponds to a classic stochastic greedy algorithm (with multiple starting points since ants are initially randomly distributed over the cities). If β = 0, only pheromone amplification is at work, that is, only pheromone is used, without any heuristic bias. This generally leads to rather poor results and, in particular, for values of a > 1 it leads to the rapid emergence of a stagnation situation, that

is, a situation in which all the ants follow the same path and construct the same tour, which, in general, is strongly suboptimal.

For an example of such a solution construction see Figure 2.1.

Once all ants of the colony have completed the construction of their solution, pheromone evaporation is performed as follows:

$$\tau_{i,j} \leftarrow (1-\rho) \cdot \tau_{i,j} \,, \forall\, \tau_{i,j} \in T \qquad (2.2)$$

where $T$ is the set of all pheromone values. Then the ants perform their return trip. Hereby, an ant—having constructed a solution $s$—performs for each $ei,j \in s$ the following pheromone deposit:

$$\tau_{i,j} \leftarrow \tau_{i,j} + \frac{Q}{f(s)} \qquad (2.3)$$

where $Q$ is again a positive constant and $f(s)$ is the objective function value of the solution $s$. As explained in the previous section, the system is iterated—applying $n_a$ ants per iteration—until a stopping condition (e.g., a time limit) is satisfied.

Even though the AS algorithm has proved that the ants foraging behavior can be transferred into an algorithm for discrete optimization, it was generally found to be inferior to state-of-the-art algorithms. Therefore, over the years several extensions and improvements of the original AS algorithm were introduced. These algorithms are all covered by the definition of the ACO metaheuristic, which will outline in the following section.

## 2.2 Ant System and Its Direct Successors

This section present AS and those ACO algorithms that are largely similar to AS. Initially, three different versions of AS[5] 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 of tour length
- The *ant-density* is an approach wherein 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 wherein 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 AS, one actually refers to ant-cycle since the two other variants were abandoned because of their inferior performance.
The relative performance of AS when compared to other metaheuristics tends to decrease dramatically as the size of the test-instance increases. Therefore, a substantial amount of research on ACO has focused on how to improve AS.

### 2.2.1 Elitist Ant System

A first improvement on the initial AS, called the elitist strategy for Ant System (EAS) was introduced by Dorigo[5]. The idea is to provide strong additional reinforcement to the arcs belonging to the best tour found since the start of the algorithm; this tour is denoted as $T^{bs}$ (best-so-far tour) in the following. Note that this additional feedback to the best-so-far tour (which can be viewed as additional pheromone deposited by an additional ant called best-so-far ant) is another example of a daemon action of the ACO metaheuristic.

**Update of Pheromone Trails**

The additional reinforcement of tour $T^{bs}$ is achieved by adding a quantity $e/C^{bs}$ to its arcs, where e is a parameter that defines the weight given to the best-so-far tour $T^{bs}$ and $C^{bs}$ is its length. Thus, equation for the pheromone deposit becomes

$$\tau_{ij} \leftarrow \tau_{ij} + \sum_{k=1}^{m} \Delta\tau_{i,j}^{k} + e\Delta\tau_{i,j}^{bs} \qquad (2.4)$$

where $\Delta\tau_{i,j}^{k}$ is the amount of pheromone ant k deposits on the arcs it has visited and $\Delta\tau_{i,j}^{bs}$ is defined as follows:

$$\Delta\tau_{ij}^{bs}(t) = \begin{cases} 1/C^{bs}(t) & \text{if arc(i, j)} \in T^{bs} \\ 0 & \text{otherwise} \end{cases} \qquad (2.5)$$

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

### 2.2.2 Rank-Based Ant System

Another improvement over AS is the rank-based version of AS ($AS_{rank}$) proposed by Bullnheimer[8]. In $AS_{rank}$ each ant deposits an amount of pheromone that decreases with its rank. Additionally, as in EAS, the best-so-far ant always deposits the largest amount of pheromone in each iteration.

**Update of Pheromone Trails**

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. Ties can be solved randomly (in implementation they are solved by lexicographic ordering on the ant name k). 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

$$\tau_{ij}(t+1) = (1-\rho)\tau_{ij}(t) + \sum_{r=1}^{w-1}(w-r)\Delta\tau_{ij}^{r}(t) + w\Delta\tau_{ij}^{gb}(t) \qquad (2.6)$$

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 EAS and significantly better than AS.

### 2.2.3 MAX–MIN Ant System

MAX–MIN Ant System (MMAS) introduces four main modifications with respect to AS[9]. First, it strongly exploits the best tours found: only either the iteration-best ant, that is, the ant that produced the best tour in the current iteration, or the best-so-far ant is allowed to deposit pheromone. Unfortunately, such a strategy may lead to a stagnation situation in which all the ants follow the same tour, because of the excessive growth of pheromone trails on arcs of a good, although suboptimal, tour. To counteract this effect, a second modification introduced by MMAS is that it limits the possible range of pheromone trail values to the interval $[\tau_{min}, \tau_{max}]$. Third, the pheromone trails are initialized to the upper pheromone trail limit, which, together with a small pheromone evaporation rate, increases the exploration of tours at the start of the search. Finally, in MMAS, pheromone trails are reinitialized each time the system approaches stagnation or when no improved tour has been generated for a certain number of consecutive iterations.

**Update of Pheromone Trails**

After all ants have constructed a tour, pheromones are updated by applying evaporation as in AS [equation (3.3)] followed by the deposit of new pheromone as follows:

$$\tau_{ij} \leftarrow \tau_{ij} + \Delta\tau_{ij}^{best} \quad (2.7)$$

where $\Delta\tau_{ij}^{best}$ = $1/C^{best}$. The ant which is allowed to add pheromone may be either the best-so-far, in which case $\Delta\tau_{ij}^{best}$ = $1/C^{bs}$ or the iteration-best, in which case $\Delta\tau_{ij}^{best}$ = $1/C^{ib}$, where $C^{ib}$ is the length of the iteration-best tour. In general, in MMAS implementations both the iteration-best and the best-so-far update rules are used, in an alternate way. Obviously, the choice of the relative frequency with which the two pheromone update rules are applied has an influence on how greedy the search is: When pheromone updates are always performed by the best-so-far ant, the search focuses very quickly around $T^{bs}$, whereas when it is the iteration-best ant that updates pheromones, then the number of arcs that receive pheromone is larger and the search is less directed.

Experimental results indicate that for small TSP instances it may be best to use only iteration-best pheromone updates, while for large TSPs with several hundreds of cities the best performance is obtained by giving an increasingly stronger emphasis to the best-so-far tour. This can be achieved, for example, by gradually increasing the frequency with which the best-so-far tour $T^{bs}$ is chosen for the trail update.

**Pheromone Trail Limits**

In MMAS, lower and upper limits $\tau_{min}$ and $\tau_{max}$ on the possible pheromone values on any arc are imposed in order to avoid search stagnation. In particular, the imposed pheromone trail limits have the effect of limiting the probability pij of selecting a city j when an ant is in city i to the interval $[p_{min}, p_{max}]$, with $0 < p_{min} \leq p_{ij} \leq p_{max} \leq 1$. Only when an ant k has just one single possible choice for the next city,that is $|N_i^k| = 1$, $p_{min} = p_{max} = 1$.

It is easy to show that, in the long run, the upper pheromone trail limit on any arc is bounded by $1/\rho C^*$, where $C^*$ is the length of the optimal tour. Based on this result,MMAS uses an estimate of this value, $1/\rho C^{bs}$, to define $\tau_{max}$: each time a new best-so-far tour is found, the value of $\tau_{max}$ is updated.

The lower pheromone trail limit is set to $\tau_{min} = \tau_{max}/a$, where a is a parameter. Experimental results suggest that, in order to avoid stagnation, the lower pheromone trail

limits play a more important role than $\tau_{max}$. On the other hand, $\tau_{max}$ remains useful for setting the pheromone values during the occasional trail reinitializations.

**Pheromone Trail Initialization and Reinitialization**

At the start of the algorithm, the initial pheromone trails are set to an estimate of the upper pheromone trail limit. This way of initializing the pheromone trails, in combination with a small pheromone evaporation parameter, causes a slow increase in the relative difference in the pheromone trail levels, so that the initial search phase of MMAS is very explorative.

As a further means of increasing the exploration of paths that have only a small probability of being chosen, in MMAS pheromone trails are occasionally reinitialized.

Pheromone trail reinitialization is typically triggered when the algorithm approaches the stagnation behavior (as measured by some statistics on the pheromone trails) or if for a given number of algorithm iterations no improved tour is found.

MMAS is one of the most studied ACO algorithms and it has been extended in many ways. In one of these extensions, the pheromone update rule occasionally uses the best tour found since the most recent reinitialization of the pheromone trails instead of the best-so-far tour. Another variant exploits the same pseudorandom proportional action choice rule as introduced by ACS.

### 2.2.4 Ant Colony System

Ant colony system (ACO) [6, 10] differs from AS in three main points. First, it exploits the search experience accumulated by the ants more strongly than AS 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 ði; jÞ 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.

**Tour Construction**

In ACS ants choose the next city using the pseudo-random-proportional action choice rule: When located at city i, ant k moves, with probability $q_0$, to city l for which $\tau_{il}(t).[\eta_{il}]^{\beta}$ is maximal, that is, with probability q0 the best possible move as indicated by the

learned pheromone trails and the heuristic information is made (exploitation of learned knowledge). With probability (1 - q0) an ant performs a biased exploration of the arcs.

**Global Pheromone Trail Update**

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

$$\tau_{ij}(t+1) = (1-\rho)\tau_{ij}(t) + \rho\Delta\tau_{ij}^{best}(t) \quad (2.8)$$

where $\Delta\tau_{ij}^{best} = 1/C^{bs}$. It is important to note that in ACS 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 AS. 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 r represents pheromone evaporation; unlike AS's equations, the deposited pheromone is discounted by a factor r; this results in the new pheromone trail being a weighted average between the old pheromone value and the amount of pheromone deposited.

In initial experiments, the use of the iteration-best tour was also considered for the pheromone updates. Although for small TSP 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.

**Local Pheromone Trail Update**

In addition to the global pheromone trail updating rule, in ACS the ants use a local pheromone update rule that apply immediately after having crossed an arc (i, j) during the tour construction:

$$\tau_{ij} \leftarrow (1-\varepsilon)\tau_{ij} + \varepsilon\tau_0 \quad (2.9)$$

where $\varepsilon$, $0 < \varepsilon < 1$, and $\tau_0$ are two parameters. The value of $\tau_0$ is set to be the same as the initial value for the pheromone trails. Experimentally, a good value for $\varepsilon$ was found to be 0:1, while a good value for $\tau_0$ was found to be $1/nC^{mn}$, where n is the number of cities in the TSP instance and $C^{mn}$, 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 $\tau_{ij}$ is reduced, so that the arc becomes less desirable for the following ants. In other words, this allows an increase in the exploration of arcs that have not been

visited yet and, in practice, has the effect that the algorithm does not show a stagnation behavior (i.e., ants do not converge to the generation of a common path). It is important to note that, while for the previously discussed AS variants it does not matter whether the ants construct the tours in parallel or sequentially, this makes a difference in ACS because of the local pheromone update rule. In most ACS implementations the choice has been to let all the ants move in parallel, although there is, at the moment, no experimental evidence in favour of one choice or the other.

## 2.3 Behaviour of ACO Algorithm

Artificial ants iteratively sample tours through a loop that includes a tour construction biased by the artificial pheromone trails and the heuristic information. The main mechanism at work in ACO algorithms that triggers the discovery of good tours is the positive feedback given through the pheromone update by the ants: the shorter the ant's tour, the higher the amount of pheromone the ant deposits on the arcs of its tour. This in turn leads to the fact that these arcs have a higher probability of being selected in the subsequent iterations of the algorithm. The emergence of arcs with high pheromone values is further reinforced by the pheromone trail evaporation that avoids an unlimited accumulation of pheromones and quickly decreases the pheromone level on arcs that only very rarely, or never, receives additional pheromone.

This behavior is illustrated in Figure 2.2, where AS is applied to the 14-city TSP instance burma14 from TSPLIB [5]. The Figure gives a visual representation of the pheromone matrix: pheromone trail levels are translated into gray scale, where black represents the highest pheromone trails and white the lowest ones.

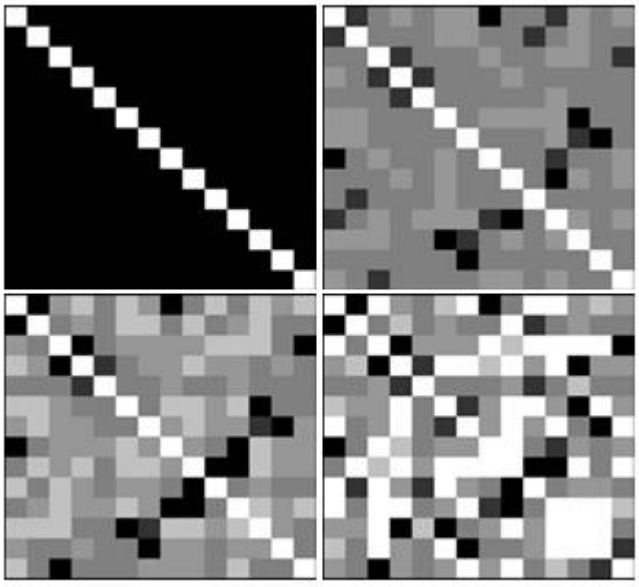


**Figure 2.2: a visual representation of the pheromone matrix. The pheromone values on the arcs, stored in the pheromone matrix, are translated into gray-scale values; the darker an entry, the higher the associated pheromone trail value. The plots, from upper left to lower right, show the pheromone value for AS applied to TSPLIB instance burma14 with 14 cities after 0, 5, 10, and 100 iterations. Note the symmetry with respect to the main diagonal, which is due to the fact that burma14 is a symmetric TSP instance.**

The four plots give snapshots of the pheromone matrix after 0, 5, 10, and 100 iterations (from upper left to lower right) by Dorigo, M., & Gambardella, L. M [8]. At the beginning, all the matrix's cells are black except for those on the diagonal which are always white because they are initialized to zero and never updated. After five iterations, the differences between the pheromone trails are still not very manifest; this is due to the fact that pheromone evaporation and pheromone update could be applied only five times and therefore large differences between the pheromone trails could not be established yet. Also, after five iterations the pheromone trails are still rather high, which is due to the large initial pheromone values.

As the algorithm continues to iterate, the differences between the pheromone values become stronger and finally a situation is reached in which only few connections have a large amount of pheromone associated with them (and therefore a large probability of

being chosen) and several connections have pheromone values close to zero, making a selection of these connections very unlikely.

With good parameter settings, the long-term effect of the pheromone trails is to progressively reduce the size of the explored search space so that the search concentrates on a small number of promising arcs. Yet, this behavior may become undesirable, if the concentration is so strong that it results in an early stagnation of the search (remember that search stagnation is defined as the situation in which all the ants follow the same path and construct the same solution). In such an undesirable situation the system has ceased to explore new possibilities and no better tour is likely to be found anymore.

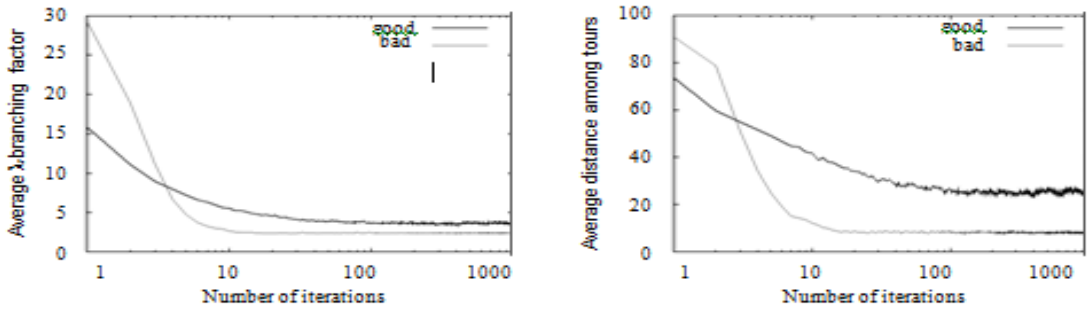



**Figure 2.3: Parameter Tuning. Bad behavior because of early stagnation: The plots give (left) the average λ-branching factor, λ = 0:05, and (right) the average distance among the tours generated by AS on the symmetric TSPLIB instance kroA100. ''Good'' system behavior is observed setting parameters to α = 1, β = 2, m = n. "Bad" system behavior is observed setting parameters α = 5, β = 0, m = n.**

## 2.4 Behavior of various AS with Parameter Tuning

This section show the typical behavior of the average λ-branching factor and of the average distance among tours when AS has parameter settings that result in either good or bad algorithm performance. The parameter settings are denoted by good and bad in Figure 2.3, and the values used are α = 1, β = 2, m = n and to α = 5, β = 0, m = n respectively. Figure 2.3 shows that for bad parameter settings the λ-branching factor converges to its minimum value much faster than for good parameter settings (l is set to 0.05). A similar situation occurs when observing the average distance between tours. In fact, the experimental results of Dorigo et al.[1] suggest that AS enters stagnation behavior if α is set to a large value, and does not find high-quality tours if α is chosen to

be much smaller than tested values of $\alpha \in \{0, 0.5, 1, 2, 5\}$. An example of bad system behavior that occurs if the amount of exploration is too large is shown in Figure 2.4. Here, good refers to the same parameter setting as above and bad to the setting $\alpha = 1$, $\beta = 0$, $m = n$. For both stagnation measures, average $\lambda$-branching factor and average distance between tours, the algorithm using the bad parameter setting is not able to focus the search on the most promising parts of the search space.

The overall result suggests that for AS good parameter settings are those that find a reasonable balance between a too narrow focus of the search process, which in the worst case may lead to stagnation behavior, and a too weak guidance of the search, which can cause excessive exploration.

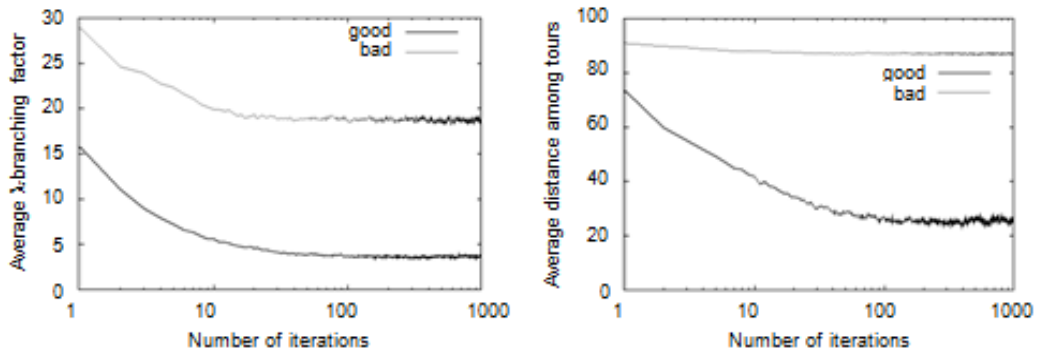



**Figure 2.4: Bad behavior because of excessive exploration: The plots give (left) the average $\lambda$-branching factor, $\lambda = 0.05$, and (right) the average distance among the tours generated by AS on the symmetric TSPLIB instance kroA100. ''Good'' system behavior is observed setting parameters to $\alpha = 1$, $\beta = 2$, $m = n$ ''Bad'' system behavior is observed setting parameters $\alpha = 1$, $\beta = 0$, $m = n$.**

### 2.4.1 Behavior of Extensions of AS

One particularity of AS extensions is that they direct the ants search in a more aggressive way. This is mainly achieved by a stronger emphasis given to the best tours found during each iteration (e.g., in MMAS) or the best-so-far tour (e.g., in ACS).

This stronger focus of the search is reflected by statistical measures of the amount of exploration would be expected. Figure 2.5 indicates the development of the $\lambda$-branching factor and the average distance between tours as observed in AS, EAS, ASrank, MMAS, and ACS. For this comparison the same parameter settings are used as in Figure 2.3, except for the value of $\beta$ which was set to 2 for all algorithms.

The various ACO algorithms show, in part, strongly different behaviors, which gives an indication that there are substantial differences in their ways of directing the search. While ACS shows a low λ-branching factor and small average distances between the tours throughout the algorithm's entire run, for the others a transition from a more explorative search phase, characterized by a rather high average λ-branching factor, to an exploitation phase, characterized by a very low average λ-branching factor, can be observed. While this transition happens very soon in AS and ASrank, it occurs only later in MMAS. On the other hand, ASrank is the only algorithm that enters stagnation when run for a sufficiently high number of iterations.

This observation also suggests that ASrank could profit from occasional pheromone trail reinitializations, as was proposed for MMAS by Stutzle & Hoos, 2000 [9]. It is interesting to note that, although MMAS also converges to the minimum average λ-branching factor, which suggests stagnation behavior, the average distance between the tours it generates remains significantly higher than zero.

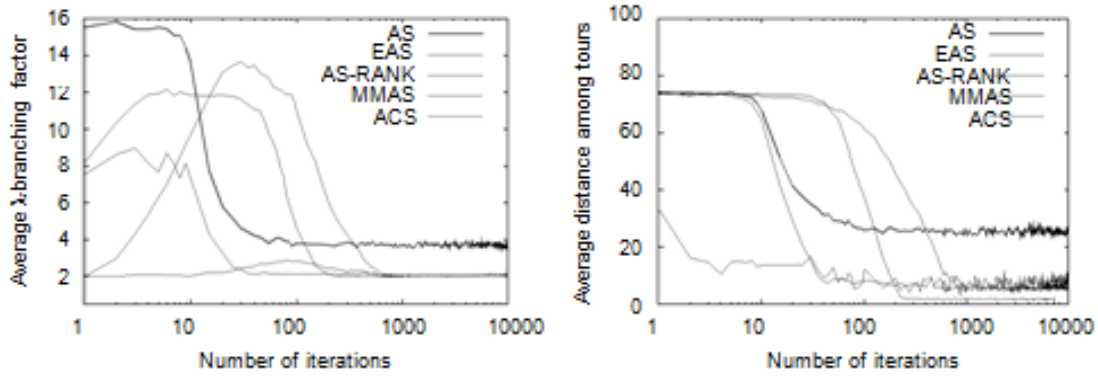



**Figure 2.5: Comparing AS extensions: The plots give (left) the average λ-branching factor, λ= 0.05, and (right) the average distance among the tours for several ACO algorithms on the symmetric TSPLIB instance kroA100. Parameters were set as in Figure 2.2, except for β which was set to β = 2 for all the algorithms.**

The reason for this apparently contradictory result is that MMAS uses pheromone trail limits $\tau_{max}$ and $\tau_{min}$. So, even when the pheromone trails on the arcs of a tour reach the value τmax and all others have the value τmin, new tours will still be explored.

A common characteristic of all of the AS extensions is that their search is focused on a specific region of the search space. An indication of this is given by the lower λ-

branching factor and the lower average distance between the tours of these extensions when compared to AS. Because of this, AS extensions need to be endowed with features intended to counteract search stagnation. It should be noted that the behavior of the various ACO algorithms also depends strongly on the parameter settings. For example, it is easy to force MMAS to converge much faster to good tours by making the search more aggressive through the use of only the best-so-far update or by a higher evaporation rate. Nevertheless, the behavior in Figure 2.5 is typical for reasonable parameter settings are shown.

In the following, the behavior of MMAS and ACS are explained in more detail. This choice is dictated by the fact that these two algorithms are the most used and often the best-performing of ACO algorithms.

### 2.4.2 Behavior of MMAS

Of the ACO algorithms considered in this chapter, MMAS has the longest explorative search phase. This is mainly due to the fact that pheromone trails are initialized to the initial estimate of $\tau max$, and that the evaporation rate is set to a low value (a value that gives good results for long runs of the algorithm was found to be $\rho = 0.02$). In fact, because of the low evaporation rate, it takes time before significant differences among the pheromone trails start to appear.

When this happens, MMAS behavior changes from explorative search to a phase of exploitation of the experience accumulated in the form of pheromone trails. In this phase, the A pheromone on the arcs corresponding to the best-found tour rises up to the maximum value $\tau max$, while on all the other arcs it decreases down to the minimum value $\tau min$. This is reflected by an average $\lambda$-branching factor of 2.0. Nevertheless, the exploration of tours is still possible, because the constraint on the minimum value of pheromone trails has the effect of giving to each arc a minimum probability $\rho_{min} > 0$ of being chosen. In practice, during this exploitation phase MMAS constructs tours that are similar to either the best-so-far or the iteration-best tour, depending on the algorithm implementation.

### 2.4.3 Behavior of ACS

ACS uses a very aggressive search that focuses from the very beginning around the best-so-far tour $T_{bs}$ by Dorigo, M. & Gambardella, L. M.[7,8]. In other words, it generates tours that differ only in a relatively small number of arcs from the best-so-far tour $T_{bs}$.

This is achieved by choosing a large value for $q_0$ in the pseudorandom proportional action choice rule in previous section of ACS, which leads to tours that have many arcs in common with the best-so-far tour. An interesting aspect of ACS is that while arcs are traversed by ants, their associated pheromone is diminished, making them less attractive, and therefore favouring 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. As a consequence, the ants never converge to a common tour, as is also shown in Figure 2.4.

## 2.5 Comparison of Ant System with Its Extensions

There remains the final question about the solution quality returned by the various ACO algorithms [5]. In Figure 2.6 Comparison of the development of the average solution quality measured in twenty-five trials for instance d198 (left side) and in five trials for instance rat783 (right side) of several ACO algorithms as a function of the computation time, which is indicated in seconds on the x-axis. Experimentally found that all extensions of AS achieve much better final solutions than AS, and in all cases the worst final solution returned by the AS extensions is better than the average final solution quality returned by AS.

In particular, it can be observed that ACS is the most aggressive of the ACO algorithms and returns the best solution quality for very short computation times.

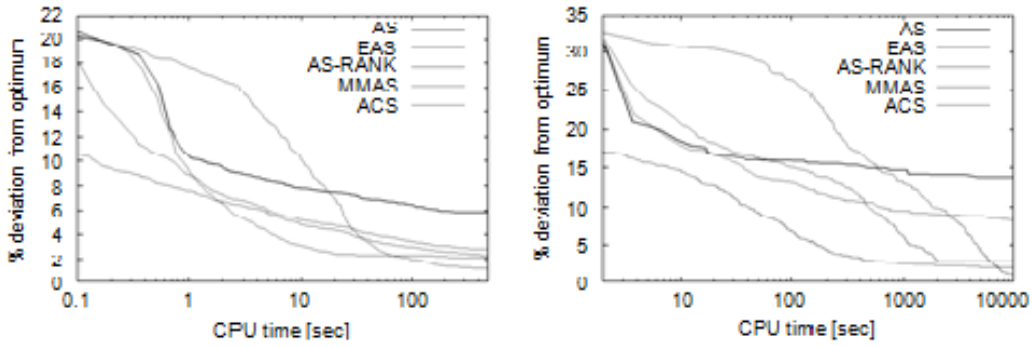



**Figure 2.6: Comparing AS extensions: The plots give the development of the average percentage deviation from the optimum as a function of the computation time in seconds for AS, EAS, $AS_{rank}$, MMAS and ACS for the symmetric TSPLIB instances**

**d198 (left), and rat783 (right). Parameters were set as indicated, except for β, which was set to β = 5 for all algorithms.**

Differently, MMAS initially produces rather poor solutions and in the initial phases it is outperformed even by AS. Nevertheless, its final solution quality, for these two instances, is the best among the compared ACO algorithms. These results are consistent with the findings of the various published research papers on AS extensions: in all these publications it was found that the respective extensions improved significantly over AS performance. Comparisons among the several AS extensions indicate that the best performing variants are MMAS and ACS, closely followed by ASrank.

## 2.6 Number of Ants

In a second series of experiments the role of the number of ants m on the final performance of MMAS [5] are investigated. MMAS using parameter settings of m ϵ {1, 2, 5, 10, 25, 50, 100} ran leaving all other choices the same. The result was that on small problem instances with up to 500 cities, the number of ants did not matter very much with respect to the best final performance. In fact, the best trade-off between solution quality and computation time seems to be obtained when using a small number of ants—between two and ten. Yet, on the larger instances, the usefulness of having a population of ants became more apparent. For instances with more than 500 cities the worst computational results were always obtained when using only one ant and the second worst results when using two ants (see Figure 2.7).

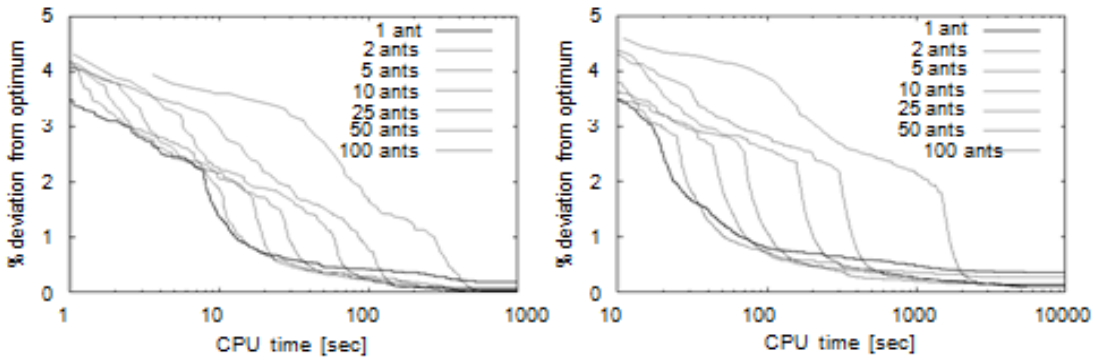



**Figure 2.7: Varying the number of ants used: The plots give the average percentage deviation from the optimal tour as a function of the CPU time in seconds for MMAS**

**using a number of ants varying from 1 ant to 100 ants on the symmetric TSPLIB instances pcb1173 (left) and pr2392 (right).**

## 2.7 Heuristic Information

It is well known that when ACO algorithms are applied to the TSP without local search, the heuristic information is essential for the generation of high-quality tours. In fact, in the initial phases of the search, the pheromones, being set to initial random values, do not guide the artificial ants, which end up constructing (and reinforcing) tours of very bad quality. The main role of the heuristic information is to avoid this, by biasing ants so that they build reasonably good tours from the very beginning.

Once local search is added to the ACO implementation, the randomly generated initial tours become good enough. It is therefore reasonable to expect that heuristic information is no longer necessary.

Experiments with MMAS and ACS on the TSP confirmed this conjecture: when used with local search, even without using heuristic information, very high-quality tours were obtained. For example, Figure 2.8 plots the average percentage deviation from the optimal tour as a function of CPU time obtained with MMAS and ACS with local search on the symmetric TSPLIB instance pcb1173. The Figure shows that MMAS without heuristic information converged in most cases somewhat slower to tours that were slightly worse than those obtained using heuristic information, while in most cases ACS's final tour length with heuristic information was slightly worse than without.

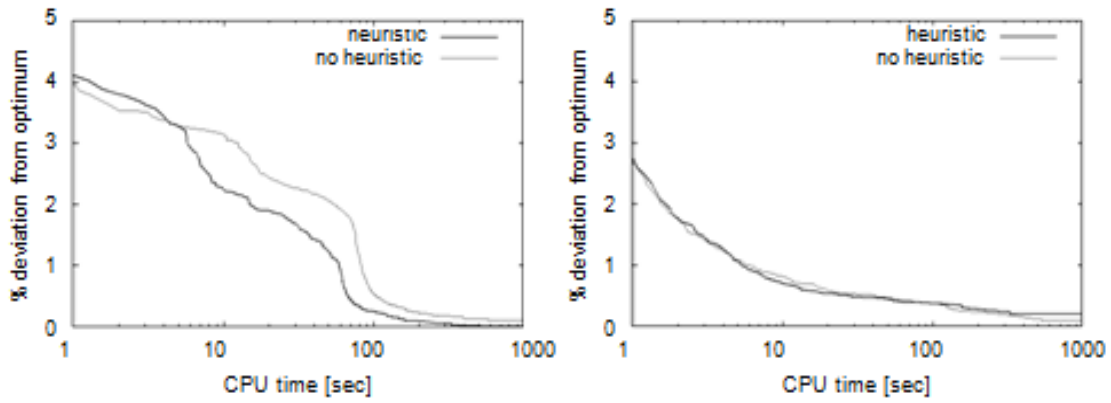



**Figure 2.8: The role of heuristic information: The plots give the average percentage deviation from the optimal tour as a function of the CPU time in seconds for MMAS**

**(left) and ACS (right) with local search on the symmetric TSPLIB instance pcb1173, with and without the use of heuristic information during tour construction.**

One might argue that the question whether heuristic information is used or not is just a matter of parameter settings (not using heuristic information is simply achieved by setting $\beta = 0$). Yet, the importance of our computational results is somewhat more far-reaching. While in the TSP the distance between cities is an obvious and computationally inexpensive heuristic to use, in other problems it may be much more difficult to find, or expensive to compute, meaningful heuristic information which helps to improve performance. Fortunately, if no such obvious heuristic information exists, our computational results suggest that using an ACO algorithm incorporating local search may be enough to achieve good results.

# CHAPTER 3

# Problem Statement

The present work deals with approaches to control the parameters of Ant Colony Optimization (ACO) algorithms for the Traveling Salesman Problem (TSP). An ACO algorithm's strength in solving optimization problems like the TSP is fundamentally based on its ability to balance two factors. On the one hand, the algorithm needs to be able to focus on promising regions of the search space by exploiting the knowledge that gets evolved in the form of the edge's different pheromone levels. On the other hand a sufficient exploration of the search space needs to be ensured by introducing the right amount of diversity into the tour construction process. In order to establish and sustain this balance, choosing a good parameter configuration is crucial. Typically, a fair amount of research is done on determining good static parameter settings. The obtained configurations are usually dependent on the algorithm's runtime, at the end of which the achieved solution quality is evaluated.

Now beside parameter control in ACO algorithms, the major shortcomings of Ant Colony Algorithms are premature convergence and revisits to individual solutions in the search space after studying lots of papers. In other words, Ant Colony Algorithm is a revisiting algorithm that escorts to duplicate routes evaluations which is a clear wastage of time and computational resources. Ant Colony Algorithms do not memorizes the solutions to the problem that it visits in its life time and it revisits lots of search points generating duplicate solutions resulting into redundant computations. The problem of revisit is all the more severe towards the end of an ACO run. In many domains, the fitness evaluation is computationally very expensive and lots of time is wasted in revisiting the parts of the search space and duplicate function evaluations.

Motivated by these needs, it is the main concern of this thesis to investigate possible options to set ACO parameters in a way that ensures a reasonably good performance for a rather large bandwidth of different runtimes.

## 1.1 Thesis Objectives

The objective of this thesis is to study and analyze the various parameters for the Ant Colony Optimization Algorithms on Traveling Salesman Problem and find the suitable

parameters. Afterward, to reduce the iterations, there is need of new methodology which reduces the CPU time to find the best optimal route. The objectives of the project are:

1. To optimize the various parameters of Ant Colony Optimization (ACO) algorithm for Traveling Salesman Problem.
2. To develop the new Ant Colony System for minimized iterations to find best optimal route by avoiding revisits of the same solution.

# CHAPTER 4

# Proposed System

Ant colony optimization (ACO) is an emerging technique. ACO is an algorithm that models the emergent behavior observed in ant colonies and utilizes this behavior to solve NP hard problems. This technique has been applied to several problems, most of which are graph related because the ant colony metaphor can be most easily applied to these types of problems.

The new method is called Ant System with Nearest Neighborhood ($AS_{nn}$) in corporation of Trie to avoid revisit of same solution and derives its name from setting the nearest neighbourhood list used in tour construction (i.e., the parameter nn_list). A detailed explanation of this approach is given below. The second section presents the results obtained from experimental analysis.

## 4.1 Methodology

The method described in the following paragraphs is based on the idea $k^{th}$ nearest neighbourhood list with Trie. In addition to the distance matrix, it is convenient to store for each city a list of its nearest neighbors. Let $d_i$ be the list of the distances from a city i to all cities j, with j = 1, 2, 3… n and $i \neq j$ (taking assumption here that the value $d_{ii}$ is assigned a value larger than $d_{max}$, where $d_{max}$ is the maximum distance between any two cities). The nearest-neighbor list of a city i is obtained by sorting the list $d_i$ according to non decreasing distances, obtaining a sorted list $d_i$; ties can be broken randomly. The position r of a city j in city i's nearest-neighbor list nn_list[i] is the index of the distance $d_{ij}$ in the sorted list $d_i$ , that is, nn_list[i][r] gives the identifier (index) of the r-th nearest city to city i (i.e., nn_list[i][r]= j). Nearest-neighbor lists for all cities can be constructed in $O(n^2 \log n)$ (in fact, you have to repeat a sorting algorithm over n-1 cities for each city).

An enormous speedup is obtained for the solution construction in ACO algorithms, if the nearest-neighbor list is cut off after a constant number nn of nearest neighbors, where typically nn is a small value ranging between 15 and 40. In this case, an ant located in city i chooses the next city among the nn nearest neighbors of i; in case the ant has already visited all the nearest neighbors, then it makes its selection among the remaining cities.

This reduces the complexity of making the choice of the next city to O(1), unless the ant has already visited all the cities in nn_list[i].

## 4.2 Experimental Setup

This section illustrates the properties of the examined TSP instances, the basic algorithm configurations that were used, as well as some general aspects of the setup.

### 4.2.1 Problem Instance

The experimental results presented in subsequent section were generated using TSP instance taken from the standard tsp library. This TSP instance is Symmetric Traveling Salesman problem in which given a set of n nodes and distances for each pair of nodes, find a roundtrip of minimal total length visiting each node exactly once. The distance from node i to node j is the same as from node j to node i. This TSP instance is known as Oliver30 which contains 30 cities. The symmetric TSP instance is given as the coordinates of a number of n points. In this case, the x and y coordinates of the cities are stored in two arrays and then compute on the fly the distance between the cities as needed.

Table 4.1: TSP Instance

| TSP Instance | Lower Bound for Optimal Route |
|---|---|
| Oliver30 | 431 |

### 4.2.2 Parameter Tuning in ACO

This section presents the analysis that has been undertaken in the scope of this work regarding fixed parameter settings. By varying the previously introduced reference configurations with respect to one parameter at a time, the algorithm's sensitivity to changes in the considered parameters was investigated. The goal of this analysis was to identify parameters with potential for improving algorithm performance when being adapted in the course of a run.

After reviewing the examined algorithm configurations in the previous sections, the respective results for Ant System will be presented. The section closes with a third section performing some analysis on the different algorithms' reference configurations.. The additional network is used to have the ants choose their values for α, β, ρ, m and n. Before an ant builds its tour on the former network, it travels the latter to derive the parameter values it uses for the remainder of the iteration. Every time it chooses an edge

in the additional network, it effectively assigns a specific value to one of its parameters. Except the global evaporation rate, all adapted parameters are ant-specific, i.e., each ant can use an individual value in its tour construction.

**Table 4.2: Parameter Tuning**

| Parameter used in Proposed Algorithm | Specific Value |
|---|---|
| α | 1 |
| β | 5 |
| ρ | 0.5 |
| m | Same no as of TSP instance cities |
| n | Same no as of TSP instance cities |

### 4.2.3 Software and Hardware

This section is intended to give the some general information on the experimental setup. The software which is used to implement the system is ANSI C. The parameter setting methods described in the remainder of the work required to extend it in various regards. The necessary runs for the experimental analysis were executed on a cluster of Intel Core2Duo CPUs (2 x 1.8 GHz, 2MB L2 cache). One computational node of the cluster contained one of these CPUs, which shared a total of 2 GB main memory.

## 4.3 Experimental Results

To test the performance of the advanced ant colony algorithm, TSP case Oliver30 TSP is selected from the General TSPLIB library, programming with the ANSI C language. A large number of comparative tests of the Basic Ant Colony and Ant colony Algorithm with Nearest Neighbourhood List ($ACO_{nn)}$ are carried out.

The maximum number of iteration in algorithm is 500.

Parameters in the Basic Ant Colony Algorithm are selected as above Table 4.2

$\alpha = 1, \beta = 4, \rho = 0.5$

The parameters of the Ant Colony Algorithm are selected as the following:

Oliver30TSP select the number of ants Antnum = 30

Algorithm is run 10 times for each example; the results of the experiment obtained are as following:

Best Distance=423.7406

Best Route is: 12->10->4->29->23->25->5->17->0->1->2->18->26->19->9->13->27->6->16->21->11->24->3->7->8->28->15->14->22->20

**Table 4.3: Experimental Result**

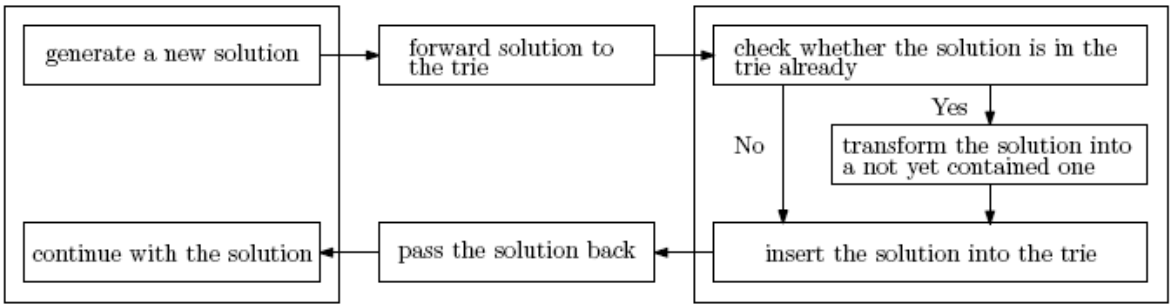
| TSP Instance | The Best Solution in TSPlib | Algorithm | Best Solution | Avg. Solution | Optimal No of Iteration |
|---|---|---|---|---|---|
| Oliver30TSP | 423.74 | BACA | 434.37 | 437.92 | 218 |
| | | $ACO_{nn}$ | 423.7406 | 426.64 | 83 |

BACA – Basic Ant Colony Algorithm

$ACO_{nn}$ - Ant Colony Algorithm with Nearest Neighbourhood List

## 4.4 Incorporation of Trie Data Structure into Proposed System

Ant Colony Optimization (ACO) share a common weakness with most other metaheuristics: Candidate solutions are in general revisited multiple times, lowering diversity and wasting precious CPU time. A complete solution archive based on a special binary trie structure for ACO with binary representations is proposed that efficiently stores all evaluated solutions during the heuristic search. Solutions that would later be revisited are detected and effectively transformed into similar yet unconsidered candidate solutions. However, also in scenarios which are believed to be rather well-configured, solutions are sometimes revisited and evaluated multiple times. Especially when the evaluation function is costly in terms of running time, such reconsiderations of the same candidate solutions may degrade performance substantially; re-evaluation in a general sense is a clear waste of precious computation time. Consider a complete solution archive based on a memory-efficient trie data structure for ACOs with binary solution representations in order to (a) efficiently detect already evaluated candidate solutions and to (b) efficiently transform them by typically small adaptations into yet unconsidered candidate solutions.




Classical ACO Trie

**Figure 4.1: Cooperation between ACO and Trie.**

### 4.4.1 The Trie

There are kinds of data structure that store things more efficiently, such as a *binary search tree, graph*. A trie is the most suitable data structure for making a dictionary of words, when concerned about the prefixes of a word. A practical use of this structure can be in auto completion for search bars, or while searching names in a phone dictionary. There are multiple implementations of Trie, but a basic one here. A Trie to store pieces of data that have a key (used to identify the data) and possibly a value (which holds any additional data associated with the key).

Trie need to store the routes that are found by Ants in iterations i.e. bunch of cities.

Here are some routes:

12345

13452

24351

24153

A trie allows us to share prefixes that are common among value. Let's start off with 12345. Trie build a tree with each character in her name in a separate node. There will also be one node under the last character in the route. In this final node, put the nul character (\0) to represent the end of the route.

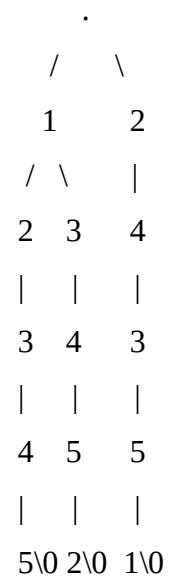
```
.    <- level 0 (root)
|
1    <- level 1
|
```

```
2    <- level 2
|
3    <- level 3
|
4  <- level 4
|
5\0  <- level 5
```

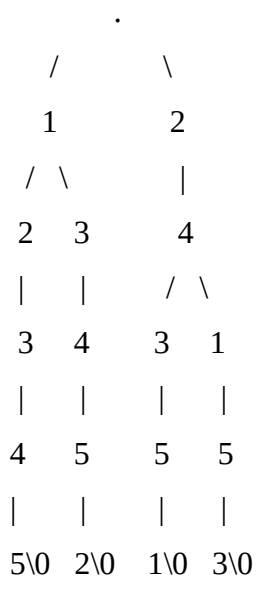
Note that each level in the trie holds a certain character in the string 12345. The first character of a string key in the trie is always at level 1, the second character at level 2, etc. Now, when to add second route 13452, the same method will be followed; however, Trie already have stored the number 1 at level 1, so there is no need to store it again, just reuse that node with 1 as the first character. Under 1 (at level 1), however, there is only a second character of 2...But, since 13452(route) has a second character of 3, Trie have to add a new branch for the rest of 13452, giving:

```
   .
   |
   1
  / \
 2   3
 |   |
 3   4
 |   |
 4   5
 |   |
 5\0 2\0
```

Now, add the third route 24351. Remember 2 is the first character and should go at level 1. Since there is no node with character 2 at level 1, Trie have to add it. In addition, Trie have to add nodes for all the other characters of 23451 under the 2. The first 3 will be a child of the 23451, the next 4 will be below the first 3, etc., giving:



Now, let's add the last route, namely 24153, giving:



### 4.4.2 Trie Operations

The list of various operations used for trie implementation are:.

- **Add** already given examples of adding in Trie.
- **IsMember** See if data with a certain string key is in the trie. For example, IsMember(trie, "12345") should report a true value and and IsMember(trie, "43215") should report a false value. There could be other

variations something with the value (like return it) once find something with the matching key.

- **Remove** Remove something from the trie.

### 4.4.3 IsMember Algorithm

Remember that a trie is a special kind of tree. Since a trie organizes its data via the keys (as specified above), it is easy to find whether a particular key is present.

Finding a key can be done with iteration (looping).

Here is an outline of such an algorithm. It looks in a particular trie and determines whether data with a particular string key is present.

IsMember(trie, key) [iterative]

1. Search top level for node that matches first character in key
2. If none,
   Return false
   Else,
3. If the matched character is \0?
   Return true
   Else,
4. Move to subtrie that matched this character
5. Advance to next character in key*
6. Go to step 1

The new search key becomes the old one without its first character.

### 4.4.4 Use of Trie in ACO Algorithm

Since the concept of Trie is explained in previous section. Now in this section, the basic idea of the ACO Algorithm ($ACO_{nn}$) is given with the help of Trie. Now question is that what is Non revisiting concept? How it will favour the algorithm efficiency?

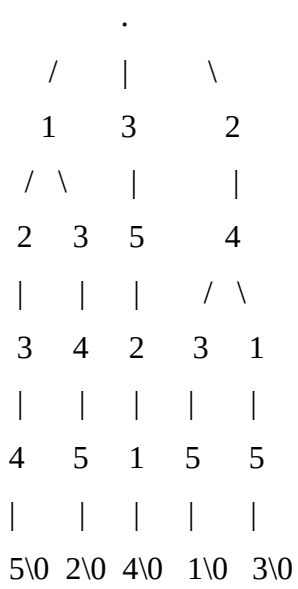
Non revisiting ACO will have these following features:

- **Remember History** First of all, The new proposed system will remember each and every ant route for each iteration unlike the previous algorithms where only best optimal route of each iteration was remembered. So All the route of each ant of each iteration will be remembered with the help of data structure Trie. The little example of how Trie store the entire paths will be given later.

- **Intelligent Future Search Decisions** On the basis of entire previous search history stored in the Trie data structure, The previous repeated path will be ignored instead of that new path will be found that means the algorithm will search the better solution in less iterations.
- **Main Objection** Now one main objection over Non revisiting approach i.e. remembering the each path of each ant will take up too much memory. Answer for this objection is that Memory is cheap in IT age so that vast majority of engineering problems involves expensive functions evaluations by which the solution is achieved in lesser iterations.

Now in this section how $ACO_{nn}$ with Trie working is explained with the help of little example of TSP. Taking TSP instance of 5 cities given below

Various routes without repetition that are stored in trie are shown below:




Now, if there is one repeated route generated by ACO algorithm for example 35214 now first of all algorithm will check by trie operation i.e. IsMember(trie, "35214"). As this route is already present in the trie, it will return yes and then proposed algorithm will transform this solution and try to find solution which has not found yet. Then solution will pass into the trie and other solution will be generated and checked wherby the mechanism giving non-revisited feasible solution.

# CHAPTER 5

# Conclusion and Future Scope

## 5.1 Conclusion

As the ACO research is not only about theory. On the contrary, most of the field is concerned with experimental work. After outlining four of the most prominent ACO algorithms for the TSP, prior work in related fields has been reviewed. Parameter setting methods introduced for Evolutionary Algorithms were examined with respect to their applicability in an ACO context and work done in the latter field has been discussed.

A first experimental analysis presented in this work focused on the performance of Ant Colony Optimization with different static parameter settings. This study started by a previously dependent reference configuration and changed only one parameter at a time.

Typically, the best performing parameter setting is dependent on the point in time at which the solution quality is measured. While some settings quickly obtain a relatively good solution but stagnate only little later, others perform worse in the beginning but improve over the former in later stages. Based on this intuition, the best performing parameters m, n, $\alpha$, $\beta$ and $\rho$ are decided during the various run of the algorithm. Computational experiments results proved that the parameters are catalyst to achieve the good performance of the algorithm.

A method called Ant Colony Optimization with Nearest Neighbourhood list is developed which provide the optimal solution for the TSP instance called Olivier30 as shown in the experimental result of the previous chapter. For further enhancement of the system, new proposed system will remember all the route of each ant of each iteration with the help of data structure Trie. Given this additional information, this proposed system is expected to be rather robust and may be worth further investigation in the future.

## 5.2 Future Scope

In the course of this work, a series of opportunities for future research have been pointed out. Now, as of idea of Ant Colony Algorithm ($ACO_{nn}$) with Trie can be run for larger TSP Instances. Further Trie data structure can be enhanced for storing solutions dynamically.

# References


[1] Dorigo M, Maniezzo V, Colorni A. *Ant System: Optimization by a colony of cooperating agents,* IEEE Trans Syst Man Cybernet Part B 26(1):29–41, 1996.

[2] Dorigo M. *Optimization, Learning and Natural Algorithms*, Ph.D. thesis, Dipartimento di Elettronica, Politecnico di Milano, Italy,1992.

[3] Blum C. *Theoretical and practical aspects of Ant colony optimization: Dissertations in Artificial Intelligence,* Berlin: Verlagsgesellschaft Aka GmbH, 2004.

[4] Blum.C. *Ant colony optimization Introduction and recent trends,* ALBCOM, LSI, Universitat Politècnica de Catalunya, Jordi Girona 1-3, Campus Nord, 08034 Barcelona, Spain,2005.

[5] Dorigo M, Stützle T. *Ant Colony optimization*, Cambridge, MA: MIT Press, 2004.

[6] Dorigo, M. & Gambardella, L. M. *Ant colonies for the traveling salesman problem, BioSystems,* 43(2), 73–81, 1997.

[7] Dorigo, M., & Gambardella, L. M. (1997b). *Ant Colony System: A cooperative learning approach to the traveling salesman problem,* IEEE Transactions on Evolutionary Computation, 1(1), 53–66.

[8] Bullnheimer B, Hartl R, Strauss C. *A new rank-based version of the Ant System: A computational study,* Central European J Operations Res Econom;7(1):25–38, 1999.

[9] Stützle T, Hoos HH. *Improvements on Ant-System: Introducing MAX–MIN Ant system, Future Generation Computer System,* 16(8):889–914, 2000.

[10] Dorigo, M., & Di Caro, *G. Ant colony optimization: A new meta-heuristic Proceeding of the 1999 Congress on Evolutionary Computation (pp. 1470–1477),* Piscataway, NJ, IEEE Press, 1999.

[11] J.L. Bentley. *Fast Algorithms for Geometric Traveling Salesman Problems*, ORSA Journal on Computing, 4(4):387-411, 1992.

[12] E. Bonabeau, M. Dorigo, and G. Theraulaz. *From Natural to Artificial Swarm Intelligence*, Oxford University Press, 1999.

[13] M. Dorigo and G. Di Caro. *The Ant Colony Optimization Meta-Heuristic.* in D. Corne, M. Dorigo, and F. Glover, editors, New Ideas in Optimization. McGraw-Hill, 1999.

[14] Stützle T and Dorigo M. *ACO Algorithms for the Traveling Salesman Problem*, IRIDIA, Universit¶e Libre de Bruxelles,Belgium,1999.

[15] T. Stutzle and H.H. Hoos. *The MAX-MIN Ant System and Local Search for the Traveling Salesman Problem.* in T. Baeck, Z. Michalewicz, and X. Yao, editors, Proceedings of the IEEE International Conference on Evolutionary Computation (ICEC'97), pages 309-314, 1997.

[16] G. Reinelt. *The Traveling Salesman: Computational Solutions for TSP Applications*, volume 840 of LNCS. Springer Verlag, 1994.

[17] Randall M. and Montgomery J. *Candidate Set Strategies for Ant Colony Optimisation*, School of Information Technology Bond University, QLD 4229, 2002.

[18] Knuth D. and Addison-Wesley *The Art of Computer Programming: Sorting and Searching*, Second Edition, 1998.

[19] Gusfield D. *Algorithms on Strings, Trees, and Sequences.* Cambridge University Press, 1997.

[20] Yuen S.Y. and Chow C.K. *A non-revisiting genetic algorithm*. In IEEE Congresson Evolutionary Computation (CEC 2007), pages 4583{4590. IEEE Press, 2007.