

9-9-2010

A more robust ant colony learning algorithm : with application to travelling salesman problem

Viswanath Nandina

Follow this and additional works at: https://digitalrepository.unm.edu/ece_etds

Recommended Citation

Nandina, Viswanath. "A more robust ant colony learning algorithm : with application to travelling salesman problem." (2010).
https://digitalrepository.unm.edu/ece_etds/187

This Thesis is brought to you for free and open access by the Engineering ETDs at UNM Digital Repository. It has been accepted for inclusion in Electrical and Computer Engineering ETDs by an authorized administrator of UNM Digital Repository. For more information, please contact disc@unm.edu.

Viswanath Nandina

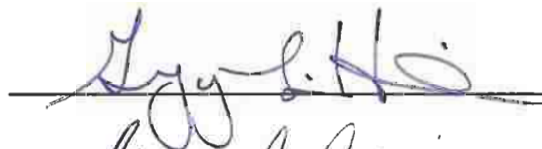
Candidate

Electrical and Computer Engineering

Department

This thesis is approved, and it is acceptable in quality and form for publication:


Approved by the Thesis Committee:



Dr. Gregory L. Heileman, Chairperson



Dr. Stephen J. Verzi



Dr. Nasir Ghani

**A MORE ROBUST ANT COLONY LEARNING ALGORITHM:
WITH APPLICATION TO TRAVELLING SALESMAN
PROBLEM**

BY

VISWANATH NANDINA

B.TECH., UTTAR PRADESH TECHNICAL UNIVERSITY

THESIS

Submitted in Partial Fulfillment of the
Requirements for the Degree of

Master of Science

Computer Engineering

The University of New Mexico
Albuquerque, New Mexico

July, 2010

© 2010, Viswanath Nandina

DEDICATION

To my family

ACKNOWLEDGEMENTS

During the development and creation of this thesis I encountered many obstacles but was able to overcome them due to the help and support of many gracious people. First, I would like to express my appreciation to my advisor Dr. Gregory Heileman. I would like to thank Dr. Stephen Verzi for his encouragement, guidance, and advice. I am grateful to Dr. Nasir Ghani for agreeing to be on my committee and reviewing this work. I would also like to thank my family members and good friends for their encouragement and support.

**A MORE ROBUST ANT COLONY LEARNING ALGORITHM:
WITH APPLICATION TO TRAVELLING SALESMAN
PROBLEM**

BY

VISWANATH NANDINA

ABSTRACT OF THESIS

Submitted in Partial Fulfillment of the
Requirements for the Degree of

Master of Science

Computer Engineering

The University of New Mexico
Albuquerque, New Mexico

July, 2010

A MORE ROBUST ANT COLONY LEARNING ALGORITHM: WITH APPLICATION TO TRAVELLING SALESMAN PROBLEM

by

Viswanath Nandina

B.Tech., Uttar Pradesh Technical University

M.S., Computer Engineering, University of New Mexico, 2010

ABSTRACT

Graph problems models many of real life applications, where the quantity of the nodes often changes with time. In such graphs, the evaluation of shortest tour is important as various guiding and navigation systems use this information. Nodes of a graph, in many applications, often change over time, and evaluation of shortest tour is essential whenever a new node is added or deleted. We, propose an algorithm that deals with such situations. We have used the Ant System with a different meta-heuristics, to find the shortest tour in a graph. We have analyzed the performance of our proposed algorithm with other algorithms by using the problem instances given in TSPLIB. The proposed modification to the Ant System heuristics will also work for directed and non-fully connected graphs. We show the use of meta-heuristics that make our algorithm free from stagnation, that is, we prevent the ants from taking up the same tour repeatedly which helps to continuously search for better results. Our approach further adopts a method, that is, a modification to Gallant's Technique, to choose the appropriate convergence within the reasonable computation time.

TABLE OF CONTENTS

DEDICATION.....	iv
ACKNOWLEDGMENTS.....	v
ABSTRACT.....	vii
TABLE OF CONTENTS.....	viii
LIST OF FIGURES.....	x
LIST OF TABLES.....	xi
CHAPTER 1.....	1
INTRODUCTION.....	1
1.1 Background.....	3
1.2 Problem Statement.....	4
1.3 Approach.....	5
CHAPTER 2.....	8
RELATED BACKGROUND.....	8
CHAPTER 3.....	13
METHODOLOGY.....	13

3.1	Initialization and Ant Memory.....	13
3.2	Probability Function and Control Parameters.....	15
3.3	Pheromone Evaluation.....	16
3.4	Stagnation and Pocket Algorithm.....	20
3.5	Formal Algorithm.....	22
3.6	Dynamic Graphs.....	23
CHAPTER 4.....		25
RESULTS.....		25
CHAPTER 5.....		42
CONCLUSION AND FUTURE WORK.....		42
REFERENCES.....		46

LIST OF FIGURES

Figure 1: Example of artificial ants.	4
Figure 4.1: Shows the solution to a 36 node Manhattan graph.....	27
Figure 4.2: Shows the relative effect of various values of α and β on the best solution for the problem ‘Ulysses 22’	30
Figure 4.3: Shows the best path found by Simple ACO per cycle..	31
Figure 4.4: Shows the best path found by our algorithm per cycle.	31
Figure 4.5: Shows the average branching of node per cycle for simple ACO.....	33
Figure 4.6: Shows the average branching of node per cycle for our algorithm.....	33
Figure 4.7: Shows the Standard Deviation for Atta 48 per cycle.	34
Figure 4.8: Shows the Standard Deviation for Berlin 52 per cycle.	34
Figure 4.9: Shows the Standard Deviation for Eli 51 per cycle.....	35
Figure 4.10: Shows the Standard Deviation for st 70 per cycle.....	35
Figure 4.11: Shows the Standard Deviation for Ulysses 16 per cycle.....	36
Figure 4.12: Shows the Standard Deviation for Ulysses 22 per cycle.....	36

LIST OF TABLES

Table 4.1: The shortest path yielded for the underlying problems	26
Table 4.2: The optimal solution yielded for the underlying problems	28
Table 4.3: A comparison of our Ant based approach to Simulated annealing.....	28
and Tabu search	
Table 4.4: A comparison between our method and the reset method for the	38
evaluation of shortest path on ‘ulyssess22’ for the addition of nodes.	
Table 4.5: A comparison between our method and the reset method for the	39
evaluation of the shortest path on ‘ulyssess22’ for the random deletion of nodes.(Continuing with the same pheromone level yielded in table 4.3)	
Table 4.6: Various convergence found in a dynamically changing graph problem	41

Chapter 1

Introduction

The travelling salesman problem (TSP) is one of the most widely studied problems in the area of operations research and theoretical computer science. In this thesis we will consider solutions to this problem that make use of ant colony heuristics. TSP is a problem where a salesman, starting from his hometown, has to find the shortest tour that will take him through a given set of cities and then back home, such that each city is visited exactly once. Formally, the TSP can be represented by a complete undirected weighted graph $G = (V, E)$ with V being the set of vertices representing the cities, and E being the set of edges, and we need to find the shortest tour connecting all nodes. In other words, the goal in the TSP is to find a minimum length Hamiltonian circuit of the graph, where a Hamiltonian circuit is a closed path visiting each node of G exactly once [27].

There are various real-life applications of the TSP problem. The TSP naturally arises as a problem in many transportation and logistics applications, for example the problem of planning and organizing school bus routes to pick up the children in a school district. The TSP can also be modeled into the problem of DNA sequencing, where cities represent DNA fragments and we need to measure similarity between DNA fragments. Another application is the problem of placement of electronic components when designing a microchip.

This problem also has importance in theory of computational complexity, since TSP is a member of the class of NP-Hard problems. This problem can be solved by either using an exact algorithm or using heuristic algorithms. An exact algorithm is an algorithm that tries to find an optimal solution for a problem. Heuristic algorithms, on the other hand, are algorithms that use educated guesses and intuitive judgments to find a general way of finding good solutions that may not be optimal. For all known exact algorithms for the TSP, as the problem size increases the number of steps necessary to solve the problem increase exponentially. So for solving the TSP problem we will choose heuristic-based approach as they can provide reasonably good solutions in polynomial time.

There are various heuristic-based algorithms that provide good solutions for the TSP problem. These methods include genetic algorithms, simulated annealing and Tabu search. We use ant-based heuristics, as they can run continuously and adapt to changes in real-time. An individual ant can behave as salesman who is independently trying to solve the problem.

Another advantage of ant-based heuristics is that they can be applied to dynamic problems. A dynamic problem is defined as a problem in which the input data is continually changing [26]. The dynamic problem in the graphs are those which either deal with the addition and removal of nodes at any instance of time or those where edge-weights change between the pair of nodes [7] [12] [17]. The dynamics that we consider are situations where nodes can be added or removed. There are various graph problems, where data changes with time and there is immediate need for a shortest path such as in robotic path planning, 3D applications, and various routing and guiding systems [4] [8] [10] [25].

So, in this thesis, we try to improve the performance of the existing Ant heuristics and solve TSP in directed or undirected dynamically changing graphs.

1.1 Background

We have used the Ant system [16], which is a class of meta-heuristics that emulates real ants. The real ants start to wander randomly from their nest, and upon finding a food source they return to their colony while laying down pheromone trails. The remaining ants on finding such a path are not likely to wander randomly, but instead follow the trail, returning and reinforcing it if they find more food. However, the pheromone trail starts to evaporate due to environmental conditions such as the sun, rain, wind, etc, thus reducing its attractive strength. The more time it takes for an ant to travel down the path and back again, the more time the pheromones have to evaporate.

Thus on a shorter path, by comparison, ants march over frequently and faster, and thus the pheromone density remains high as it is laid on the path. Pheromone evaporation also has the advantage of avoiding the convergence to a locally optimal food site. If there were no evaporation at all, the paths chosen by the first ants would likely be more attractive to the following ones. In that case, the exploration of the solution space would be constrained. Thus, when one ant finds a good path from the colony to a food source, other ants are more likely to follow that path.

1.2 Problem Statement

This thesis focuses on solving the travelling salesman problem. In particular, it comes up with a meta-heuristics that will enable it to search continuously through the search domain to find a single-source shortest tour in a dynamically changing environment.

1.3 Approach

To obtain a near-optimal solution, the procedure for laying pheromone and the function for controlling evaporation of pheromones should be based upon solution quality [16] [18]. So in our meta-heuristics we have proposed a new procedure for pheromone updating and pheromone evaporation.

Our pheromone evaporation procedure has close similarities with the rank-based ant system approach [3]. The n solutions found in a single cycle are prioritized according to their length, and the pheromone is evaporated inversely, where n is the number of nodes. We experimented with other changes in the ant system which eventually helped us to solve dynamically changing problems [16]. These modifications include changes in the control parameters, pheromone-updating procedure and in the pheromone evaporation procedure.

Our proposed method uses the elitist strategy [3] [5] [16]. We also use the lower-bound property of the max-min ant system [20] [21] [22]. In the elitist strategy a small amount of pheromone is laid on the first few shortest tours found, thus forcing the algorithm to make a biased selection towards these tours in the later iterations. The elitist strategy and the lower

bound property both helped to improve the quality of solutions. The changes adopted in these functions are described in sections 3.2 and 3.3.

Convergence is a desirable property in optimization and search algorithms. Convergence is a situation where the possible solutions unify to approach towards a definite value or a fixed or equilibrium state [17]. In ACO (Ant Colony Optimization) convergence is easy to achieve because of the desired stagnation condition of artificial ants [11]. If we force the new ants to follow the initially found shortest path, and if other ants originating from other nodes try to follow this path, they can easily move on this path. However, as each ant will select a path based on the distance they have traversed until now, the result is we get different solutions. Therefore to determine the termination condition in our proposed algorithm we apply the pocket algorithm [9]. We will treat the pocket content as the pheromone levels which will lead to an identical solution for a longer period of time and the misclassification as the shortest path length. This procedure will help us in selecting the desired convergence in a relatively fewer number of iterations.

The Ant System meta-heuristic, when used for undirected graphs, causes the ants to follow any path involving the nodes that have not been traversed and use the probability function in order to complete a tour [14] [16]. In the case of a directed graph, there are fixed paths for the movement. When the ant moves on the graph, the computation effort increases as the number of possible rated solutions decreases. So it is a more complex task to get the optimal solution in directed graphs compared to undirected graphs [28].

As an ant starts its movement in a directed graph, there is a possibility that it can move to any of the paths. The movement depends on two factors: pheromone level and edge length [16]. But there is a possibility that an ant can get trapped on a node where further movement is impossible. We call this a valid blocked path. To overcome these blocked paths, the Ant system takes a large number of cycles. Further, if the alternate solutions found are similar, then the number of cycles increases. So in our proposed method, we calculate the valid length of the valid edges in this path by calculating the summation of all the valid edges in this path. We also calculate the number of irregularities associated with this path, i.e., the total number of nodes for which a valid path could not be found.

For a dynamically changing environment, often the reset method and the population-based approach are used to compute the tour [12]. In the reset method, whenever a new node is added or removed, the current pheromone values present on each edge of the graph are reset to the initial value of pheromone. In other words, whenever a new node is added or removed, this graph is treated as a new graph and ant heuristics are re-applied to this graph from the beginning. So we compare how our meta-heuristics perform against the reset method.

In the next chapter we will consider the previous approaches and related background work done on the problem. In Chapter 3 we present our algorithm, and Chapter 4 describes the results. Finally Chapter 5 concludes this thesis and outlines future research possibilities.

Chapter 2

Related Background

The two popular heuristic-based algorithms for finding solutions of TSP are Simulated annealing and Tabu search [14]. Simulated annealing is a generic probabilistic meta-heuristic for the global optimization problem of applied mathematics, namely locating a good approximation to a global optimum of a given function in a large search space [29]. In the simulated annealing method, each point of the search space is comparable to some physical system, and the fitness function to be minimized is comparable to the internal energy of the system in that state. The goal is to bring the system, from an initial state, to a state with the minimum possible energy [29].

Greedy algorithms can also be used to find solutions for the TSP, but they may not always yield a global optimal solution [23]. Thus we propose a meta-heuristics which not only has the advantages of the greedy heuristics, but also involves ant system ideologies, in order to find a good solution.

Another approach that is being used is expansion-based algorithms; they work on surface modeling. A surface model can consist of millions of uncommon polygons. For the surface modeling, naïve algorithms are used. A naïve algorithm works by initially randomly analyzing a polygon. Then a selection for the next polygon surface is made such that it shares its boundaries and satisfies a fitness function, defined by the algorithm. A naïve algorithm tries to skip a few parts of the surface model without considering them

further [15]. This naïve algorithm uses a two-step procedure: Initially it computes a local path and then later tries to select a search region based upon the initial value.

In many cases, the naïve approach fails to find a healthy solution because the search domain has been reduced. Further, the search is also biased towards the solutions that are found during the initial local search. The algorithm tries to compute a local path based upon some global optimal criteria. If this solution belongs to a set of non-healthy solutions, then the algorithm can be unnecessarily pushed to a large search region [15].

So without searching the complete domain, it will be impossible to judge which region has the optimal path. Therefore, it is desired for an algorithm to consider all the regions irrespective of which one is better and that algorithm should continuously search for solutions. By reducing the search domain, we may increase the performance of our system, but the optimal solution is not guaranteed [15]. Thus even if a healthy solution is found, a desired algorithm should never stop the search for solutions because there is always a chance for better solutions.

So our methodology involves the use of a number of heuristics, described in later sections, which will help us to find alternative solutions. These alternative paths may or may not be better than the existing best shortest path. But with the help of these heuristics, we are able to justify which regions in the search space cannot have optimal or good solutions and thus searching in those regions will not provide us with good solutions.

We now describe the Simple Ant Heuristics. Given a fully connected graph $G = (V, E)$, with Euclidean distances, where V is the number of vertices and E is the edges between the vertices. Let $b_i(t)$ ($i=1, \dots, n$) be the number of ants in city i at time t , and let n be the total number of ants. Each ant is a simple agent with the following characteristics:

- It chooses the town to go to with a probability that is a function of the town distance called the “visibility” and the amount of trail present on the connecting edge.
- To force the ant to make legal tours, that is transitions to already visited cities are disallowed until a tour is completed.
- When it completes a tour, it lays a substance called a *trail* on each *edge* (i,j) that it has visited.

Each ant at time t chooses the next city, where it will be at time $t+1$. Therefore, we call an iteration of the algorithm, the m moves carried out by the m ants in the interval $(t, t+1)$. Then after every m iterations of the algorithm (which we call a cycle) each ant has completed a tour. At the end of each cycle we have m ants that found m tours. So at this point the trail intensity is updated by laying pheromone on the edges traversed by an ant and by evaporating the pheromone. Formally, the ant Heuristics algorithm is stated as follows [16]:

Formal Algorithm

Initilize()

1. Set $t:=0$ *{t is the time counter}*
2. Set $C:=0$ *{C is the cycles counter}*
3. For every edge (i,j)
4. Set an initial value $\tau_{ij}(t)$ for trail intensity and $\Delta\tau_{ij}=0$
5. Place the m ants on the n nodes

Make_Ant_tour()

6. Set $s:=1$ *{s is the tabu list index}*
7. For $k:=1$ to m do *{This step initializes the Tabu List}*
8. Place the starting town of the k-th ant in **tabu_k**(s)
9. Repeat until tabu list is full *{this step will be repeated (n-1) times}*
10. Set $s:=s+1$
11. For $k:=1$ to m do
12. Choose the town j to move to, with probability $p_{ij}^k(t)$ given by equation (4)
 {the k-th ant is now on town $i=\mathbf{tabu}_k(s-1)$ at time t}
13. Move the k-th ant to the town j
14. Insert town j in **tabu_k**(s)

Compute_Path_Legth(tabu_k)

15. For $k:=1$ to m do
16. Compute the length L_k of the tour described by **tabu_k**
17. Update the shortest tour found

Evaluate_Pheremone_to_be_laid()

18. For every edge (i,j)
19. For $k:=1$ to m do
20. $\Delta\tau_{ij}^k = 1/L_k$ if (i,j) tour described in **tabu_k**
21. $\Delta\tau_{ij} := \Delta\tau_{ij} + \Delta\tau_{ij}^k$

Update_pheremone()

22. For every edge (i,j) compute $\tau_{ij}(t+n)$ according to equation $\tau_{ij}(t+n)=\rho\cdot(\tau_{ij})+ \Delta\tau_{ij}$

Check_termination_condition()

23. Set $t:=t+n$
24. Set $C:=C+1$
25. For every edge (i,j)
26. set $\Delta\tau_{ij}:=0$
27. If $(C < C_{max})$ and (not stagnation behavior) then
28. Empty all tabu lists
29. Go to step 6
30. Else

31. Print shortest tour
32. Stop

The algorithm starts by placing a single ant on each node. Each ant will use the probability function to evaluate the next node to visit. Once a decision is made the ants will travel to the next node, laying the pheromone on the travelled edge. This process is repeated while all the ants have not completed a legal tour. Once all ants have completed a legal tour, pheromone on all the edges is updated, and later pheromone is allowed to evaporate. These are the steps involved in a single cycle. The algorithm is allowed to run for a user-defined number of cycles, or until the stagnation is achieved.

Stagnation is a situation where all the ants start to follow the same tour. We believe stagnation is not desirable, as better solutions to the problem may exist in the search space, and once stagnation is achieved ants will not be able to search the search space. So our algorithm avoids stagnation and continuously searches the search space.

The complexity of the ant-cycle algorithm is $O(Cn^2m)$ after C cycles, where n is the number of cities and m is the number of ants. In fact, step 1 is $O(n^2+m)$, step 2 is $O(m)$, step 3 is $O(n^2m)$, step 4 is $O(n^2m)$, step 5 is $O(n^2)$, step 6 is $O(nm)$ [16]. Since there is linear relation between the number of towns and the best number of ants, the complexity of the algorithm is $O(Cn^3)$ [16].

Chapter 3

Methodology

3.1 Initialization and Ant memory

We start with initializing the ant memory. We model our artificial ants to have some memory. This is represented by using a two-dimensional data structure in computer memory of size $N \times N$, where N is the number of nodes in the graph. A row indicates the path followed by an ant. The algorithm proceeds by placing an ant on each node of the graph, which we will term as the “home node.” This is done by putting the address of each respective home node in the first column of every row in the ant memory.

We only place a single ant on each node as we want the number of ants present in the graph searching for a solution to be linear in the number of nodes. If there are a large number of ants present in the graph, then each route will have a considerably large amount of pheromone deposited on it due to the ant movement. The sufficient amount of pheromone will also be deposited on solutions with a large length. Thus a significant amount of time will be needed to evaporate the pheromone on the longer edges. If we have a smaller number of ants as compared to the number of nodes in the graph, then a very small amount of pheromone will be laid on the edges. The resulting high rate of pheromone evaporation could cause pheromone levels on some of the shorter edges to drop to zero. Hence it may take a long time to increase the amount of pheromone present on these edges.

As the ants are placed onto each node, we will initialize all the edges in the graph with a constant, very small amount of pheromone. This small amount of pheromone will act like a limiting factor. If the pheromone on an edge falls below this limit, we can discard the edge from being used in the next few iterations. Another importance of this value is that during the very first iteration, when probability is computed, this provides some relative influence of pheromone to probability function, without which the algorithm will start to behave as a greedy heuristic.

As the ants start to move from one edge to another, we must keep a record of the route they have covered and the amount of pheromone they have laid on the edges. So we make use of two other data structures that are similar to ant memory. After calculating the probability of whatever edge has to be selected, the ant that moves on it and lays a relative amount of pheromone on it, based on the length of the edge and also based on the length of the tour that it has covered. As the ant moves from one node to another, we record the amount of pheromone it lays on the node and the total tour length it has travelled. Based on this record, the edge is subjected to a varying rate of pheromone evaporation. The rate of pheromone evaporation is so varied that the routes with large tour lengths are subjected to a higher extent of pheromone evaporation, and shorter and better tour lengths are subject to lesser pheromone evaporation. This is done to ensure that pheromone is deposited on the edge based on the length of the tour. If an edge of small length is part of a big tour, then this shorter edge should receive more pheromone compared to longer edges of the same tour. If a large edge is part of a short tour, then a considerably large amount of pheromone is deposited on that edge.

We consider $N * (N - 1)$ iterations as a single cycle, where N is the number of nodes in the graph. A cycle is complete when each ant has traversed all the nodes exactly once. After the completion of the cycle, the ant memory matrix will be full. All the tour lengths are computed and the shortest of them is stored separately. Once a single cycle is completed, we assume that these ants are dead, and new ants are placed on the nodes for the next cycle.

3.2 Probability Function and Control Parameters:

We have used the probability function as given in the equation and its purpose is to find the best node for the ant to move further. Thus probability of the k^{th} ant moving from node i to node j is:

$$p_{ij}^k(t) = \frac{[\tau_{ij}(t)]^\alpha \cdot [\eta_{ij}]^\beta}{\sum_{k \in N_i^h} [\tau_{ik}(t)]^\alpha \cdot [\eta_{ik}]^\beta} \quad \text{if } j \in N_i^k \quad (3.1)$$

Where,

η_{ij} is the Visibility, the quantity $1/d_{ij}$

τ_{ij} is the pheromone present on the *edge* _{ij} .

N_i^k specifies the nodes that ant_i has not traversed.

N_i^h specifies the nodes that ant_i has traversed

We have also used the concept of control parameters with some modifications in their values. As the control parameters, α and β are used to manage the relative influence of the visibility factor and pheromone. If the value of α is zero, then the heuristic adopts a greedy nature and if α is greater than one, then the heuristic tends to encourage a speedy development of

inferior solutions. β is used to control the effect of pheromone on the probability. If β is equal to zero, then the heuristic considers the amplification of pheromone information.

Our experiments revealed that if we sequentially increase both the values of α and β , the algorithm tends to generate good solutions that have a minimum percentage deviation from the optimal value. As discussed in ant system, to find the shortest path we get the inferior solutions for $\alpha \geq 1$ [18]. Through experimental results we found that for $\alpha = 1$, we get inferior solutions. Thus, based on the results from the experiments, shown in Figure 4.1, in our meta-heuristics we will be using $\alpha = 0.5$ and the $\beta = 6$.

3.3 Pheromone Evaluation:

In the beginning of our proposed heuristics, we have suggested the method for pheromone biasing along with pheromone updating and excessive pheromone removal method. After the completion of the first cycle as discussed in the Ant Memory section, we enforce an additional amount of pheromone on the edges of the shortest path, which was generated in the first cycle.

This is done by a procedure similar to elitist strategy [4] [16]:

$$T_{ij} = T_{ij} + 1/(N * PathLength_n) \quad (3.2)$$

Where,

N is the number of nodes

$PathLength_n$ is the length of shortest path generated in the first cycle.

This addition of pheromone will cause these edges to occur in the next few cycles, and when a shorter path is found, considerable amounts of pheromone will be evaporated on this path that we will discuss later under `Rem_Excessive_Pheromone` module.

After applying equation 3.1, we further analyze the methods of pheromone deposits and evaporations in the Ant System and suggest the following changes in the subsequent sections.

The quantity of pheromone laid by an ant should be based on the quality of solution yielded, or else we have to face an array of premature solutions. Thus we evolved a new approach for updating the pheromone. Let the initial pheromone level present on the edges be IT_{ij} .

The pheromone is deposited by

$$T_{ij} \leftarrow T_{ij} + \sum_{s=1}^n \Delta T_{ij}^k \quad (3.3)$$

Where

$$\Delta T_{ij}^k = \frac{\eta}{leg^k} \quad \text{if } edge_{ij} \text{ belongs to } P_k \quad (3.4)$$

Where,

P_k is the path covered by ant_k

leg^k is the length of the path P_k

η is quantity of trail laid down by ants.

ΔT_{ij} amount of pheromone to be added.

We have also proposed the new approach for the pheromone evaporation which is treated separately, rather than including it with the pheromone updating method. After the completion of

a cycle, we prioritize the paths found in the ant memory. The priority of path P_k is denoted Ω_k based on its length. The smaller the value of Ω_k , the higher the priority. If all the paths yielded in a cycle are greater than the existing best path found, then we will degrade their priority by one. And if all the yielded paths are identical to GBSP, then we ensure that only pheromone deposition takes places without any evaporation.

Now for each ant_k , and for every $edge_{ij}$ belonging in P_k , pheromone is evaporated by:

Evapo_Pheromone_Path (P_k, Ω_k)

$$\begin{aligned} \Delta T_{ij} &= T_{ij} \\ \text{For } a: &= 0 \text{ to } \Omega_k \\ \Delta T_{ij} &= \mu \Delta T_{ij} \\ T_{ij} &= IT_{ij} + \Delta T_{ij} \end{aligned}$$

Where,

T_{ij} is the trail present on the $edge_{ij}$, which is present in the P_k and $\mu = 0.5$, is rate of pheromone evaporation.

The result of this multi-staged pheromone evaporation yields an extremely small value of the pheromone on edges belonging to longer tours. But still we do not allow the pheromone levels to drop below the IT_{ij} . This helps in preventing the edge from being branched off from the next coming iterations. Once a smaller tour than the current existing shortest tour is found, then an additional layer of pheromone is deposited on these $edge_{ij}$, as shown in equation 3.5.

Add_Excessive_Pheromone (New found Shortest Tour)

$$T_{ij} = \Psi + T_{ij} \quad (3.5)$$

Where, T_{ij} is the trail present on the $edge_{ij}$ which belongs to the new found shortest tour $\Psi=0.3$, is the excessive pheromone added. This $\Psi=0.3$ is a fixed value found using experiments.

Now we also remove this excessive pheromone ‘ Ψ ’, added on the $edge_{ij}$ from the previous best shortest tour. This pheromone accumulation process is only conducted when a new shortest tour is found. For this we use:

Rem_Excessive_Pheromone (Old Shorter tour)

$$\Delta T_{ij} = T_{ij}$$

For $a: = 0$ to n

$$\Delta T_{ij} = \Psi \Delta T_{ij}$$

$$T_{ij} = \Delta T_{ij}$$

We found that our algorithm is making inclined decisions toward the initially found paths, or a decision toward any set of solutions found. But the removal of excessive pheromone by different stages of evaporation, from the $edge_{ij}$ of the shortest tour found, will prevent new ants from making a biased decision toward this tour. If solutions tend to lean toward the set of found solutions, then these solutions must also be present in the shortest tour. When pheromone is evaporated, these edges will have a greatly reduced amount of pheromone. Thus, this algorithm will not make any inclined decisions.

Also the removal of excessive pheromone from the $edge_{ij}$ will cause the pheromone level to eventually drop below the initial level IT_{ij} . This in turn will prevent the ants from following this tour and eventually yield new solutions.

3.4 Stagnation and Pocket Algorithm

Stagnation was termed as a situation where all the ants will follow the same tour. The stagnation occurs when a strong amount of pheromone is laid on a considerably good solution and, after each iteration, the amount of pheromone is so enforced on the path. Then the ants tend to make the decision based on the probability function in favor of this tour. Thus the ants keep on selecting this tour. This is the point where the ant system converges to a solution. But a relatively best tour does not imply that this is always the globally best tour or nullify the fact that a better tour may exist. Thus an algorithm must continue to perform such operations that will enable it to find an alternative solution. The alternative solution found may or may not be better than the existing good solution found.

Thus we implement the above described methodology. During the execution of the above procedures, we don't allow the pheromone to fall below the initial value pheromone on any edge. But when a better tour than the existing found tour is established then we allow the pheromone on those edges to fall below the initial value, knowing that a better solution exists. This is where branching is performed. Once the pheromone is dropped below the initial value, there is a very little probability that this edge will be selected in the next coming iterations, as all other edges will have a considerably large amount of pheromone compared to it.

But as the algorithm proceeds, the quantity of pheromone on the smaller edges of the tours starts to increase in size, and the ants do tend to select these edges due to the relative importance of β in the probability function. Thus eventually the amount of pheromone on these edges rises above the initial value and thus alternative solutions can be found.

Our algorithm will terminate when one of the following three conditions is satisfied:

A). The number of cycles exceeds beyond the maximum number of cycles i.e. C_{\max} . This value is generally inputted by the user.

B). The total number of times the algorithm provides us with a set of constant good solutions, represented by the PocketValue, exceeds some user-defined constant. We decide to terminate the algorithm when we get the same solution for some fixed number of iterations. This again is a user-defined constant.

C). The total number of continuous reoccurrences of the same solution exceeds some user-defined constant value. Whenever we find a continuous set of constant values, we will store them in memory as a single packet. When such a packet is equal to a user-inputted number, we can terminate the algorithm. Each packet found can be of variable size but must be greater than one.

Formal Algorithm

Initialize()

1. $C = 0$, *{Counter for number of cycles}*
2. T = user defined value for termination
3. For every edge $_{ij}$, set Pheremone Trail to IT_{ij} *{Initial pheromone trail value}*
4. Place a new ant on each node, i.e., on the first column of each row in 'ntrav'.

Make_Ant_tour()

5. Until the 'ntrav' is full *{Until each ant has completed a tour}*
6. Chose the town j to move ant_k to, i.e., select the next node that yielded the maximum probability.
7. Move ant_k to node j
8. Put this node j in the subsequent column ntrav for ant_k *{i.e., column k }*

Evaluate_CBSP(ntrav)

9. For $k:=1$ to n
10. Compute the length the tour length traversed by the ant_k
11. Find out CBSP *{Find the shortest path in ntrav}*
12. If ($C=0$) then GBSP = CBSP

Evaluate_priority_for_each_path(ntrav)

13. Compute the priority Ω_k of each path $_k$ based on its length. *{Smaller path has higher priority}*
14. If ($GBSP > CBSP$) then
15. For every path path $_k$, we decrease the priority of the path by one

Evaluate_Pheremone_to_be_laid()

16. For every edge $_{ij}$ evaluate the amount of pheromone deposited on the edge *{shown in equation 3.3 and equation 3.4}*
17. Evaporate pheromone on every edge $_{ij}$, by Evapo_Pheremone (path $_k$, Ω_k) *{shown in section 3.3}*

Pheremone_update_on_new_GBSP()

18. If ($CBSP < GBSP$) then Remove excessive pheromone from the edges $_{ij}$ of GBSP, by Rem_Excessive_Pheromone (GBSP) *{shown in section 3.3}*
19. Add_Excessive_Pheromone (CBSP) *{shown in section 3.3}*
20. GBSP = CBSP
21. If ($CBSP < GBSP$) then
22. Update Pocket Contents, Store the path CBSP, Store Pheromone instance

```

Evaluate_Pockets()                                {Checks the pocket size, no of pockets}
23.  If (CBSP = GBSP) then
24.    Current_ren_length = Current_ren_length +1
25.  If (Current_ren_length > Best_run_length) then
26.    Store Pheromone instance
27.    Best_run_length = Current_ren_length        {Determining Pocket Size}
28.  If ((PREVIOUS_CBSP! = GBSP) and (CBSP = GBCP)) then
29.    PocketValue = PocketValue + 1             {Determining number of pockets}
30.  PREVIOUS_CBSP = CBSP

Termination_check()
31.  C = C +1;
32.  If (C > Cmax) or (PocketValue > PocketValuemax) or
    (Best_run_length > PocketSizemax) then
33.    Print GBSP
34.  Else Goto Step 4
35.  Stop

```

3.5 Dynamic Graphs:

The above-mentioned procedures work well for dynamically changing graphs. In dynamic graphs, nodes can be randomly added or removed. Whenever a new node is added, all other nodes are directly linked to this node through their Euclidean distance and the initial value of pheromone on these edges. When a node is removed, the pheromone levels of the node and of all the edges connecting to it will be set to zero.

Each time a new node is added, the number of columns in the ant memory will be altered. Thus to make a complete legal tour, the ants will have to take the new added node into account and then would have to select this node. This would now alter the tour length and amount of pheromone laid on each edge. Thus when the pheromone evaporation is done, varying amounts of pheromone will be evaporated on these edges. So now the ants will tend to increase the

amount of pheromone on the shorter tours. These new tours formed may have a tour length lesser or greater than the initially found shorter tours. This actually depends on coordinates of where the new node is added. A similar action will be performed when a node is removed. It takes a longer time to find a new shorter tour when a node is removed because the ants would have already laid a considerable amount of pheromone on those edges. Trying to reduce the intensity of pheromone on those edges and finding a new tour takes more time compared to when a new node is added.

Chapter 4

Results

The main goal of this thesis is to show how both the control parameters and the dynamic changing behavior of the graph affect convergence. This thesis uses the problems from the TSPLIB for analysis [1]. TSPLIB is a library of sample instances for the TSP and related problems from various sources and types of graphs, such as fully connected graphs or directed graphs.

TSPLIB defines a number of problems like Berlin 52, where ‘Berlin’ is the name of the graph that represents 52 locations in Berlin. The edge weight type is ECU_2D [31], which is represented as a fully connected, two dimensional graph with Euclidean distances between each location. Another problem used was bays29; edge weight type is GEO meaning edge weights are geographical distances. The locations of nodes are given in latitudes and longitudes. Atta48 was also one of the graphs used. It is a graph with Pseudo-Euclidean distances, where distances are represented by the number of hops required to communicate. Another graph used was the Dsj1000, a 1000 node graph. The edge weight type in this graph is CEIL_2D [31], which means the Euclidean distances in two dimensions are rounded up.

We also performed experiments with a 36 node graph with Manhattan distances. Manhattan distance is the distance between two points measured along axes at right angles. The results for which are shown in Figure 4.1. Figure 4.1a shows the Manhattan graph with unit distances and Figure 4.1b shows the solution yielded by the algorithm shown in blue lines.

TABLE 4.1: The shortest path yielded for the underlying problems

Problem Name	Edge Weight Type	Number of Nodes	First Occurrence of Solution (on Cycle number)	Convergences (After number of cycles)	Path Length	Path
Atta48	Pseudo-Euclidean	48	21	272	10628	8,1,9,38,31,44,18,7,28,36,6,37,19,27,43,17,30,20,12,15,33,46,40,3,2,16,41,34,14,25,13,23,11,47,21,39,32,48,5,29,2,4,26,10,42,24,45,35,8
bays29	GEO	29	174	246	2705	9,17,24,13,7,28,8,26,19,1,18,6,27,21,1000,12,4,10,11,23,5,16,3,14,22,2,2,5,20,15
Berlin52	ECU_2D	52	15	306	7542	46,5,15,24,48,37,44,35,36,39,40,38,6,4,25,12,28,27,26,47,14,13,52,11,51,33,43,10,9,8,41,19,45,32,49,1,22,31,18,3,17,21,23,20,50,16,29,30,42,7,2,46
Eli51	ECU_2D	51	411	798	442	40,13,41,19,42,44,15,45,33,10,39,30,34,50,9,49,5,38,11,32,1,22,2,21,20,3,35,36,28,31,8,26,7,23,6,14,25,18,4,17,37,12,47,46,51,27,48,24,43,40
St70	ECU_2D	70	91	1264	675	30,20,14,3,32,7,2,4,18,42,6,41,43,17,9,40,61,39,45,25,46,27,68,44,8,28,26,49,55,19,24,15,57,22,63,66,59,69,31,13,29,70,35,38,23,1,36,16,47,37,58,50,10,5,53,52,60,12,21,34,33,54,62,48,67,11,56,51,65,64,30
Swiss42	ECU_2D	42	57	358	1284	33,1,2,7,5,4,3,28,29,31,30,9,10,24,42,11,26,12,13,19,27,6,14,20,15,27,16,38,8,18,32,37,36,34,21,35,39,23,40,22,41,25,33
ulysses16	ECU_2D	16	20	167	6859	11,9,10,7,6,5,15,14,13,12,16,1,8,4,2,3,11
ulysses22	ECU_2D	22	63	216	7013	3,2,17,4,18,22,8,1,16,12,16,14,15,5,6,7,19,21,20,10,9,11,3

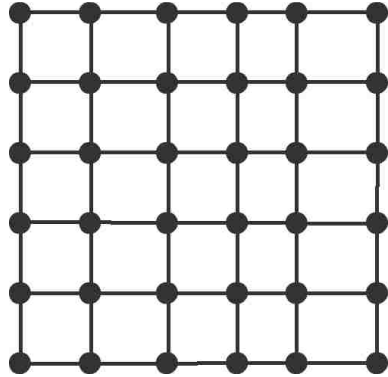


Figure 4.1a

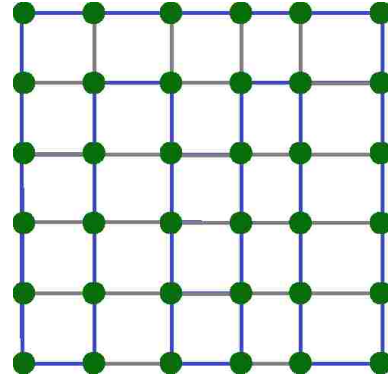


Figure 4.1 b

Figure 4.1: Shows the solution to a 36 node Manhattan graph.

The problems defined in the Table 4.1 are standard problems taken from the TSPLIB. TSPLIB is a library of sample instances for the TSP (and related problems) from various sources and of various types. TSP is defined as: 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 . We also made sure that there is one ant on every node. The optimal solutions for the above stated problems are given in Table 4.2 [1].

TABLE 4.2: The optimal solution yielded for the underlying problems

Problem Name	Optimal Solution
Atta48	10628
Berlin52	7542
Eli51	426
St70	675
Swiss42	1273
ulysses16	6859
ulysses22	7013

A comparison of Table 4.1 and Table 4.2 shows that we found optimal solutions for problems like Atta 48, Berlin 52, Swiss 42, Ulysses 16 and Ulysses 22 in less than 500 cycles.

We also compared our algorithm with Simulated annealing [33] and Tabu search [34]. The algorithms were executed on Berlin 52. The results are shown in Table 4.3

TABLE 4.3: A comparison of our Ant based approach to Simulated annealing and Tabu search on Berlin 52.

Algorithm	First Occurrence of Solution (on Cycle number)	Shortest Path Length found
Ant based algorithm	15	7542
Tabu search	236	8667
Simulated annealing	157	7452

We also performed our experiments to evaluate a range of the control parameters α and β . When the test was performed on Ulysses 16, we get the path length 6859 and convergence in less than 500 cycles, especially for the pair values of:

1. ($\alpha=0.4$, $\beta=6$)
2. ($\alpha=0.5$, $\beta=6$)
3. ($\alpha=0.6$, $\beta=6$)

The result of path lengths found for various combination values of α and β are shown in Figure 4.2.

α and β are the control parameters, where α controls the relative influence of the pheromone laid and β controls the relative influence of the length of the path covered. Therefore the transition probability function is a trade-off between visibility and pheromone intensity present on the path at time t . If α is set to zero, the closest cities are more likely to be selected. This corresponds to a classic greedy algorithm (with multiple starting points, since ants are initially randomly distributed over the cities). If β is set to zero, only pheromone amplification is at work, i.e., only pheromone is used, without any heuristic bias. This generally leads to rather poor results and, in particular, for values of $\alpha > 1$, it leads to a rapid emergence of a stagnation situation. That is, a situation in which all the ants follow the same path and construct the same tour.

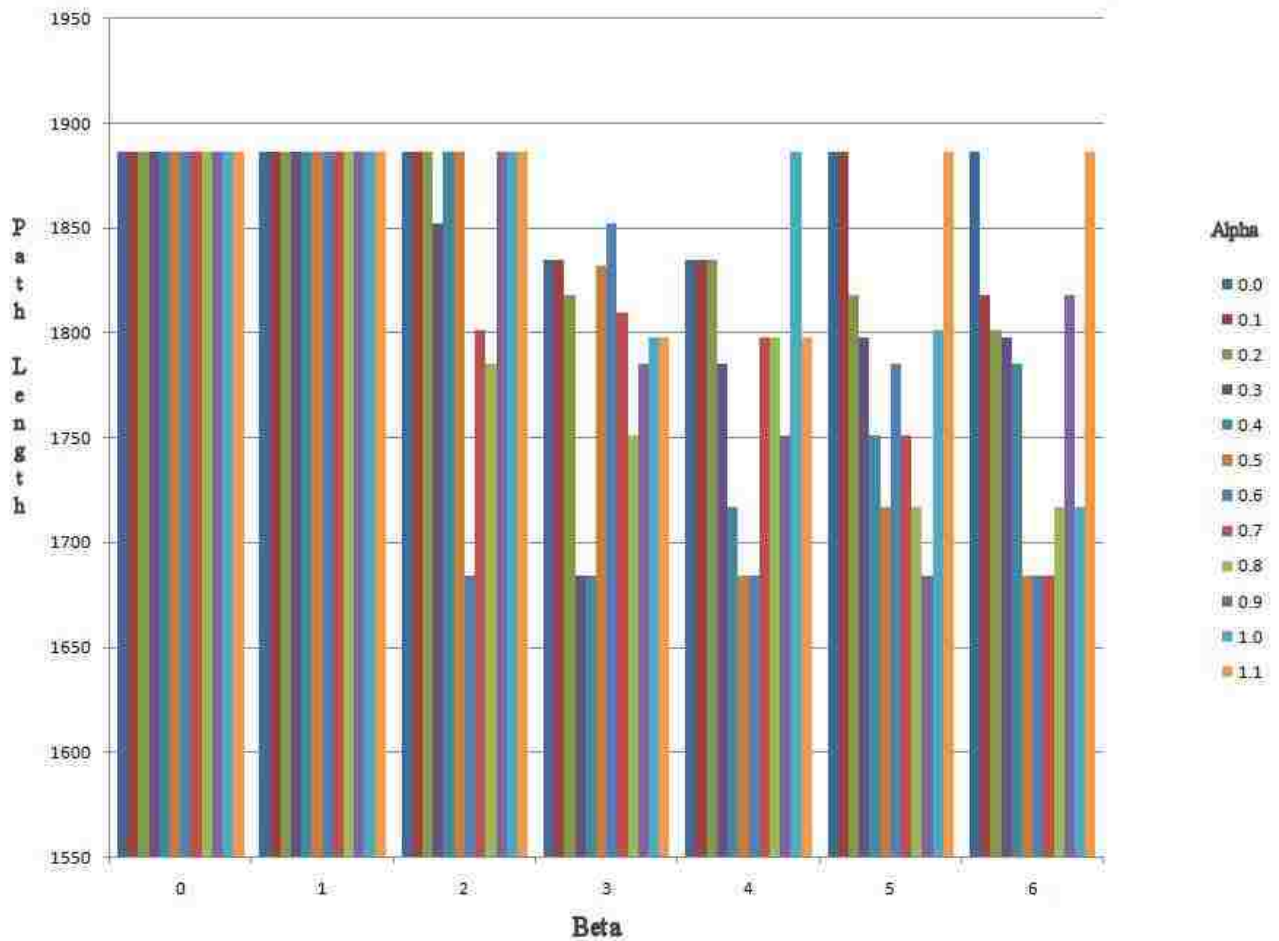


Figure 4.2: Shows the relative effect of various values of α and β on the best solution for the problem ‘Ulysses 22’.

We performed a comparison of our algorithm and the Simple ACO on the problem named Oliver 30. This is because most of the solutions stated in for Simple ACO are shown with respect to Oliver 30 [16]. Oliver 30 is a problem from TSPLIB for which the optimal solution is 423 [1].

The following graph shows and compares the shortest path found by the Simple ACO and by our approach. The first occurrence of the best solution path was found on the 34th cycle by

our approach, shown in Figure 4.4, while it took Simple ACO roughly 300 cycles, shown in Figure 4.3 [16].

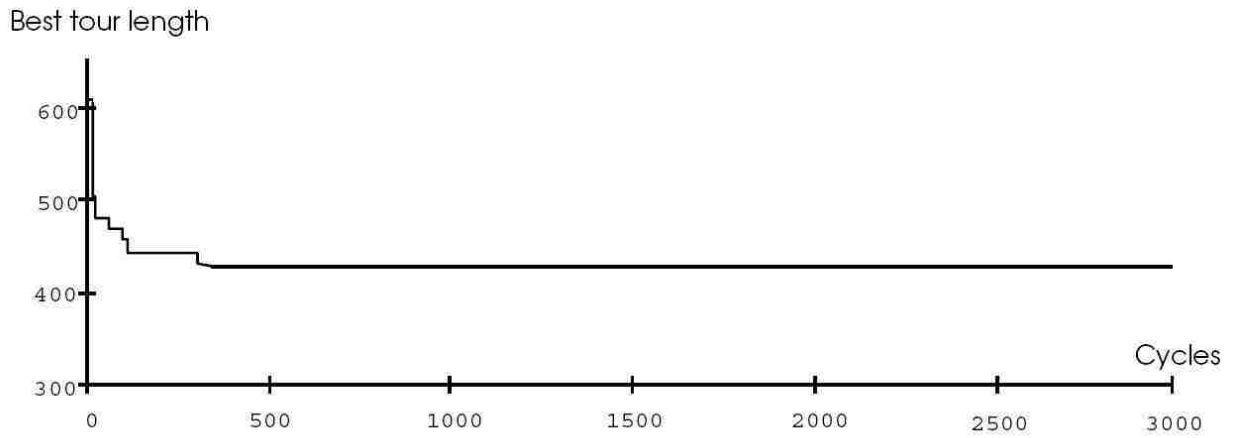


Figure 4.3: Shows the best path found by Simple ACO per cycle.

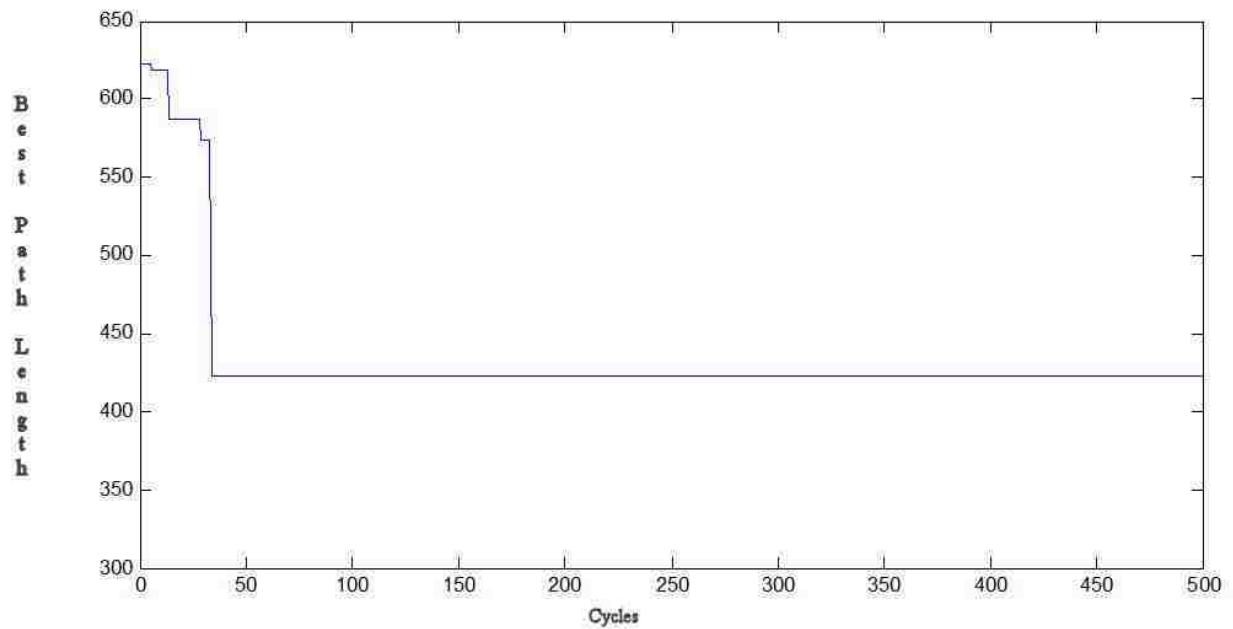


Figure 4.4: Shows the best path found by our algorithm per cycle

We observe such changes because we laid the pheromone on the edge based on the quality of the solution. That is if we have a longer path, then a smaller amount of pheromone will be deposited. When we have a shorter path, more pheromone will be deposited on this path. Another important point to note is that a shorter edge in a longer path will receive more pheromone compared to a longer edge in a longer path, which will receive less pheromone. This procedure is described by equation 3.4.

Also, the average node branching of the graphs in our approach does not fall as rapidly as of the Simple ACO. Figure 4.4 shows that the average branching factor falls below 15 in the first 100 cycles and falls below 10 in less than 250 cycles [16]. This means the algorithm is rapidly discarding the possible solutions without any scope of re-considering them for a possible solution, as they have a branching factor of 0.1. So, once the pheromone level has fallen below 0.1, that edge is branched off. On the other hand, in our approach we do not let the pheromone level on any edge fall below the initial level of pheromone IT_{ij} . We only let the pheromone level fall below the IT_{ij} when we find a path that is better than the existing path by removing excess pheromone on the previously found best path by the procedure defined in section 3.3. But still, these edges in the previously found best path can be reconsidered by any ant, if the ant reaches these edges and has to traverse them in order to complete the tour. So even if we branch off a node, we make sure to reconsider the edge if an ant wants to complete a legal tour using this edge. Figure 4.6 shows that in our approach, the average branching factor does not decrease rapidly. Thus, we ensure that our domain of solutions does not decrease rapidly and we still have a lot of solutions to consider before arriving on a final solution path.

Average node branching

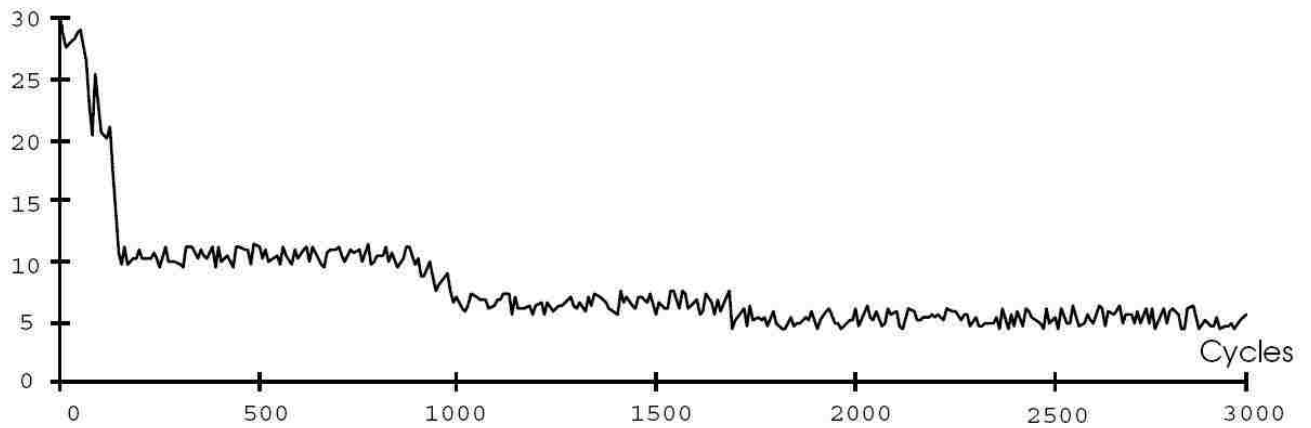


Figure 4.5: Shows the average branching of node per cycle for simple ACO.

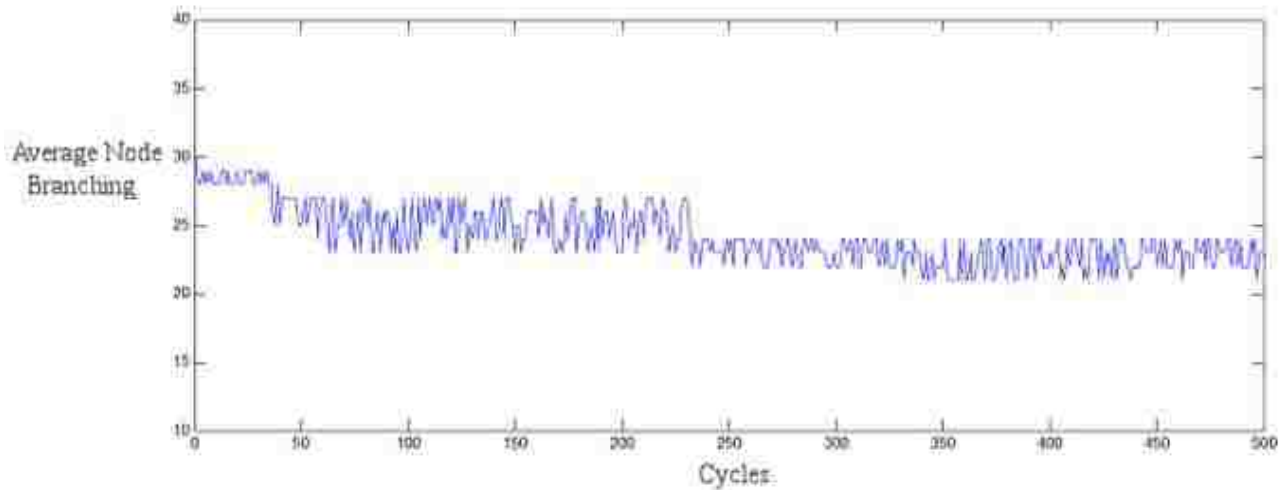


Figure 4.6: Shows the average branching of node per cycle for our algorithm.

Another feature of this algorithm is that the standard deviation of the all the tours found during a single cycle never becomes constant or approaches to zero. This is credited to the fact that the ant lays pheromone and makes a decision on the amount of pheromone to lay on the edge by taking into consideration the overall distance it had travelled until now, and the total number of nodes still left in order to complete a legal tour. This is done in order to ensure that the

pheromone is laid based on the quality of the solution. Figure 4.7 – Figure 4.13 show the standard deviation of tours per cycle.

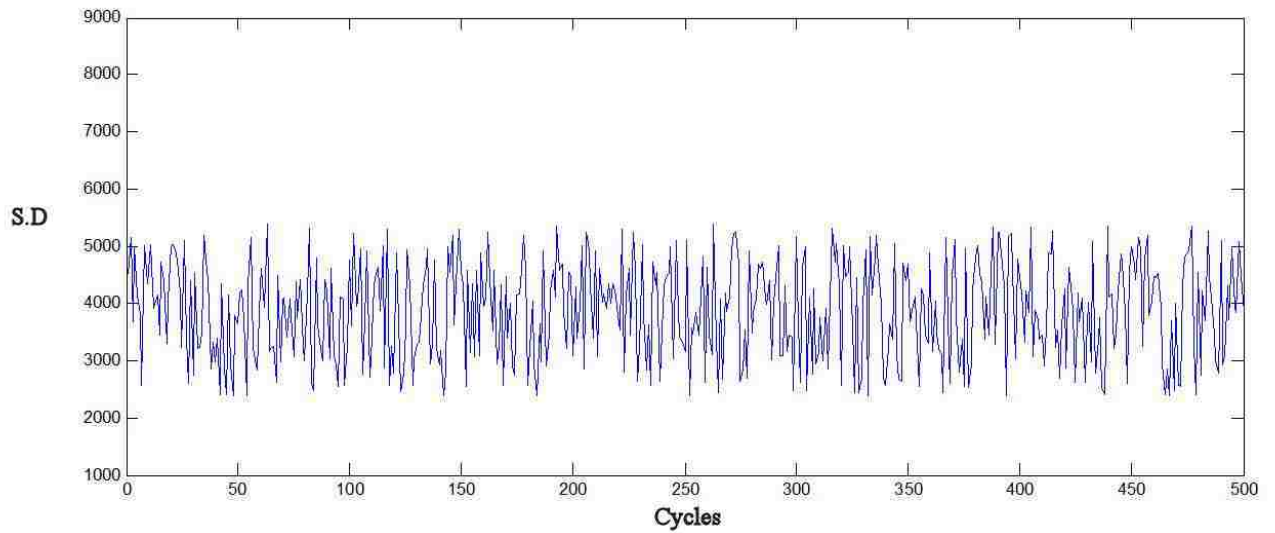


Figure 4.7: Shows the Standard Deviation for Atta 48 per cycle

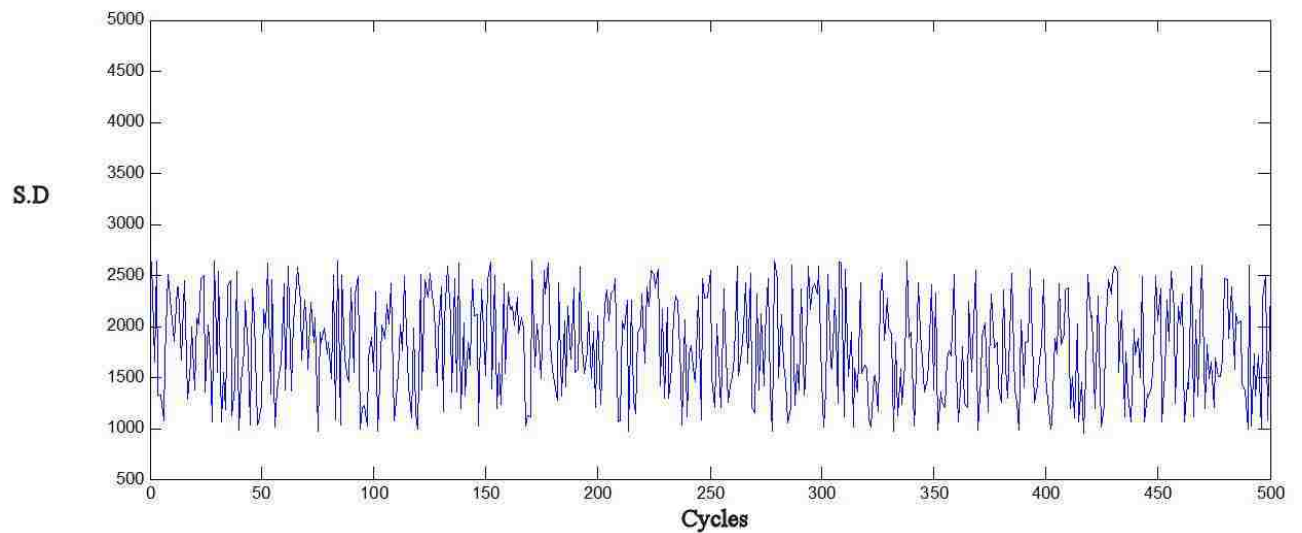


Figure 4.8: Shows the Standard Deviation for Berlin 52 per cycle

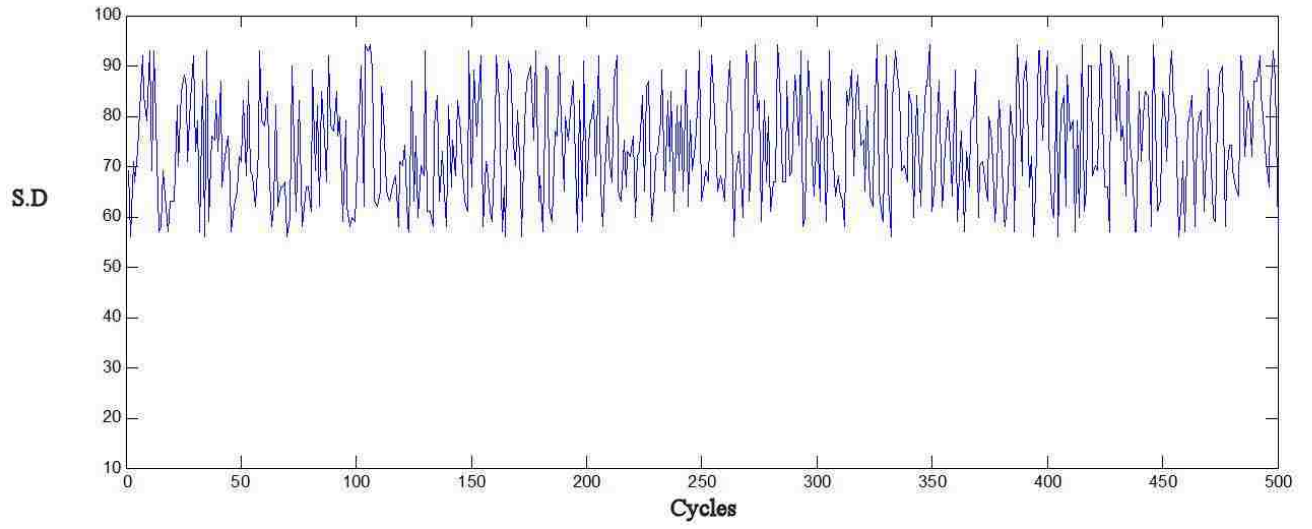


Figure 4.9: Shows the Standard Deviation for Eli 51 per cycle

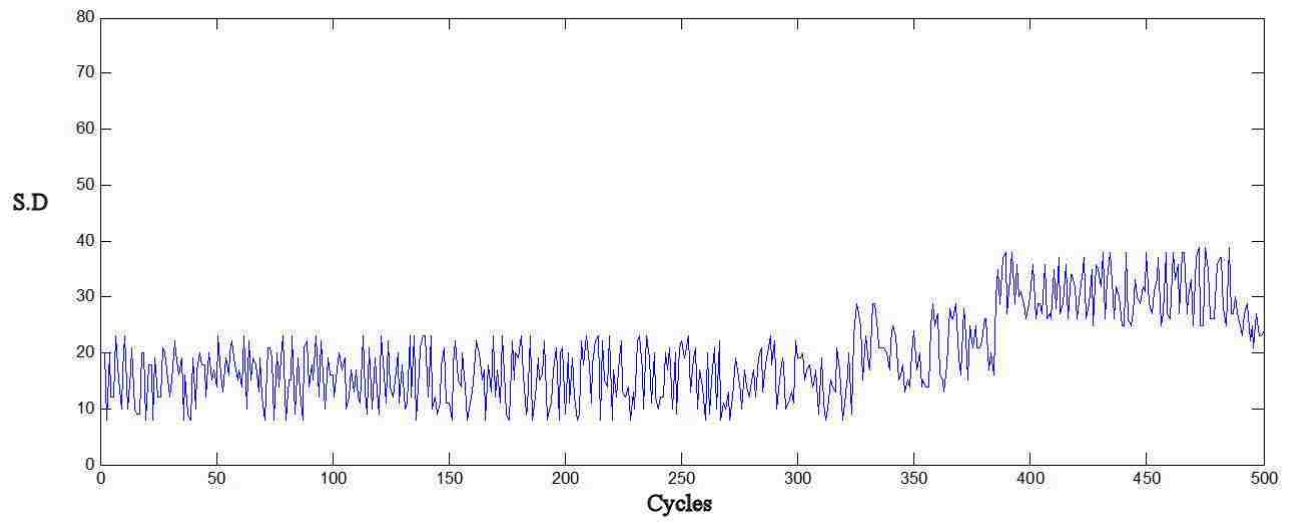


Figure 4.10: Shows the Standard Deviation for st 70 per cycle

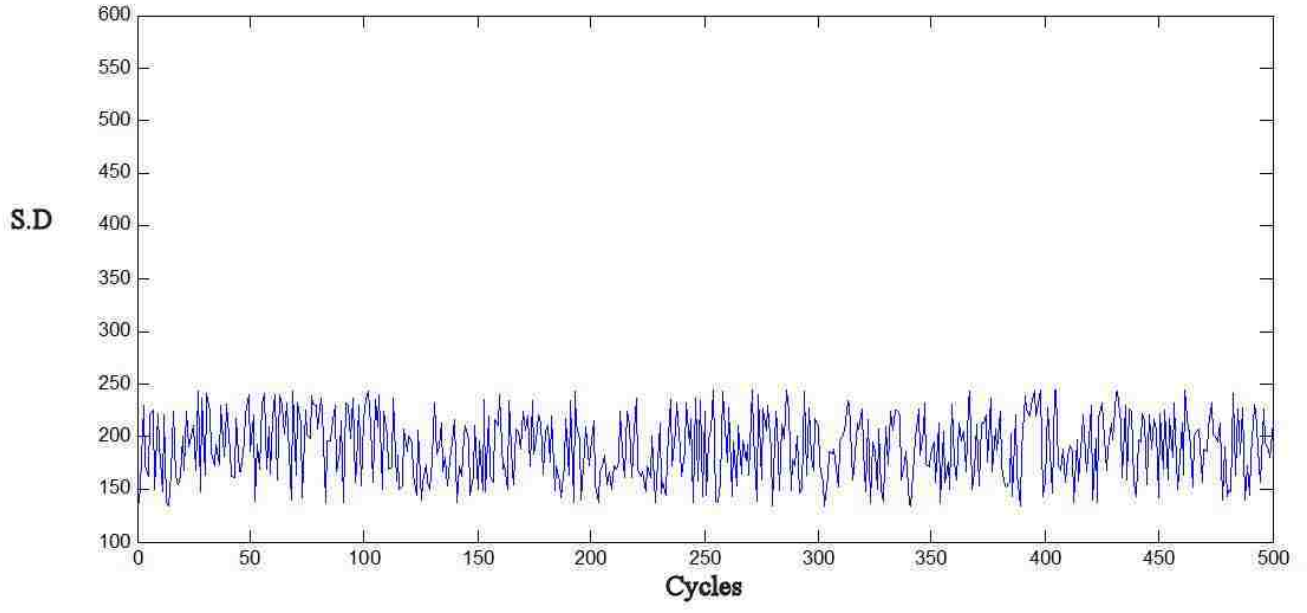


Figure 4.11: Shows the Standard Deviation for Ulysses 16 per cycle

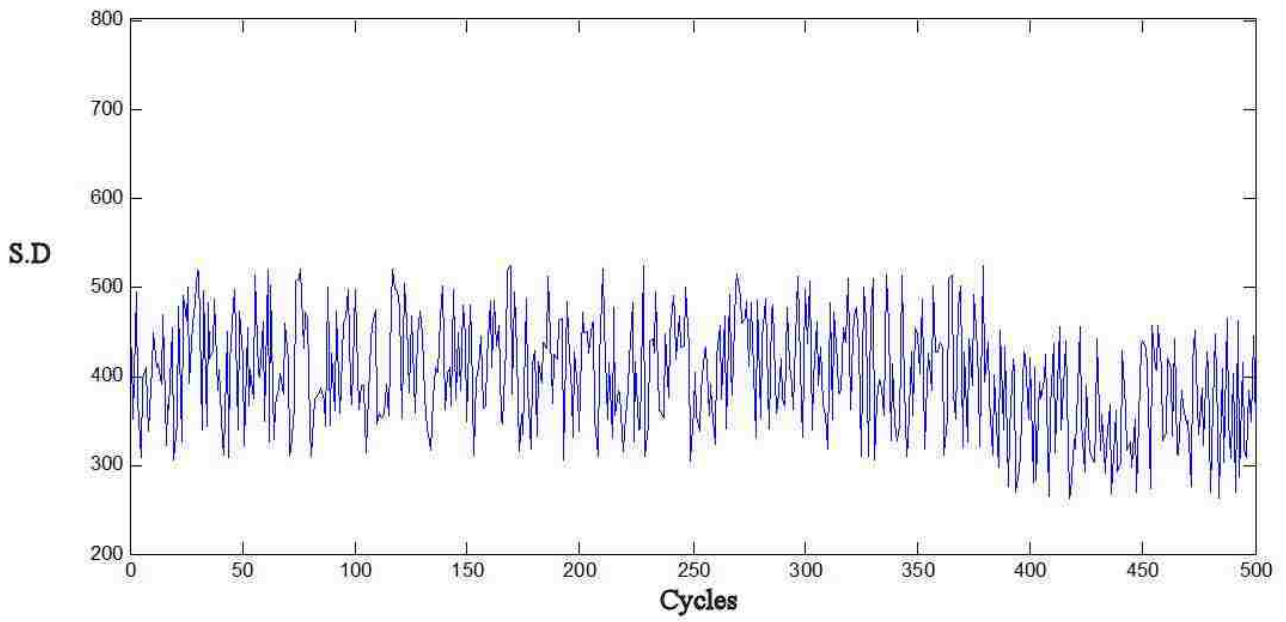


Figure 4.12: Shows the Standard Deviation for Ulysses 22 per cycle

We can observe that when the standard deviation becomes zero, then all the ants will follow the same path. This is the situation of stagnation. And when the standard deviation becomes constant, we can say that in each cycle the same tours are formed again and again. But from the graphs in Figure 4.7 to Figure 4.12, we can conclude that our approach is free from stagnation, and a continuous search for a better solution is carried out in each cycle of the algorithm.

Another feature we tried to incorporate in our approach is a method to handle dynamically changing graphs. For Ulysses 22, we added and removed nodes from a graph over time. We compare our method to the reset method which is described as applying the Simple ACO algorithm to the new graph formed by adding or removing nodes [16]. When a new node is added, a fully connected graph is formed by connecting all nodes to this new node, and all these edges are initialized with the initial value of pheromone τ_{ij} . And when a node is removed, then all edges which connect this node to the rest of the nodes are removed, and the pheromone values are removed from the NTRAV matrix. Table 4.4 shows a comparison between our approach and reset methods when nodes are added, and Table 4.5 shows a comparison between our approach and reset method when nodes are removed. The algorithm runs for 500 cycles before a node is added or removed.

TABLE 4.4: A comparison between our method and the reset method for the evaluation of shortest path on ‘ulysses22’ for the addition of nodes.

No. of Nodes	Coordinates of the Nodes added	Our Proposed Method		Reset Method	
		Path Length	Path	Path Length	Path
22	INITIAL	7013	3, 2, 17, 4, 18, 22, 8, 1, 16, 12, 16, 14, 15, 5, 6, 7, 19, 21, 20, 10, 9, 11	7013	3, 2, 17, 4, 18, 22, 8, 1, 16, 12, 16, 14, 15, 5, 6, 7, 19, 21, 20, 10, 9, 11
23	(500,500)	7959	11, 9, 6, 7, 12, 13, 14, 15, 1, 8, 16, 22, 18, 4, 17, 2, 3, 10, 19, 21, 20, 5, 1	8846	3, 2, 17, 4, 18, 22, 8, 1, 16, 12, 13, 14, 15, 6, 7, 19, 21, 20, 10, 9, 5, 11, 23
24	(1200,3000)	11226	11, 5, 9, 10, 19, 20, 21, 12, 13, 14, 7, 6, 15, 16, 1, 8, 22, 4, 18, 17, 2, 3, 24,23	11512	3, 2, 17, 4, 18, 22, 8, 1, 16, 21, 20, 19, 10, 7, 6, 12, 13, 14, 15, 5, 9, 11, 23, 24
25	(4500,900)	11991	3, 2, 17, 4, 18, 22, 1, 8, 16, 12, 13, 14, 7, 6, 5, 15, 20, 21, 19, 10, 9, 25, 11, 23, 24	11923	11, 6, 7, 12, 13, 14, 15, 5, 9, 25, 10, 19, 20, 21, 16, 1, 8, 22, 4, 18, 17, 2, 3, 24, 23
26	(3200,1750)	11898	11, 25, 9, 10, 19, 20, 21, 7, 6, 5, 15, 14, 13, 12, 16, 1, 8, 22, 4, 18, 17, 2, 3, 26, 24, 23	12994	11,5,6,7,21,20,19,10,9,25,13,12,14,15,8,1,16,17,18,4,22,3,2,26,24,23
27	(2350,1970)	12476	11, 5, 6, 7, 13, 12, 14, 15, 21, 20, 19, 10, 9, 25, 16, 1, 8, 22, 4, 18, 17, 2, 3, 26, 24, 23	12648	11,5,15,13,12,14,7,6,20,21,19,10,9,25,16,1,8,22,4,18,2,3,17,26,27,24, 23
28	(1560,2300)	12488	11, 5, 15, 13, 12, 14, 7, 6, 20, 21, 19, 10, 9, 25, 16, 1, 8, 22, 4, 18, 17, 2, 3, 26, 27, 24, 23	12540	11, 5, 15, 13, 12, 14, 7, 6, 20, 21, 19, 10, 9, 25, 16, 1, 8, 22, 17, 3, 24, 18, 26, 27, 28, 24, 23
29	(970,2800)	12907	11, 5, 6, 7, 12, 13, 14, 15, 21, 20, 19, 10, 9, 25, 1, 8, 16, 22, 4, 18, 17, 2, 3, 26, 27, 29, 24, 28, 23	13042	11, 5, 15, 13, 12, 14, 7, 6, 20, 21, 19, 10, 9, 25, 16, 1,8, 22, 4, 18, 17, 3, 2, 26, 27, 28, 29, 24, 23
30	(1800,2650)	13117	3, 2, 17, 4, 18, 22, 1, 8, 16, 13, 12, 14, 7, 6, 19, 20, 21, 10, 9, 25, 11, 5, 15, 26, 27, 29, 24, 28, 23	13172	11, 5, 15, 14, 13, 12, 7, 6, 20, 21, 19, 10, 9, 25, 16, 1, 8, 4, 18, 22, 17, 3, 2, 26, 27, 30, 28, 24, 29, 23
31	(2700,1450)	13485	23, 27, 29, 24, 28, 26, 30, 56, 7, 12, 13, 14, 15, 1, 8, 16, 22, 4, 18, 17, 2, 3, 19, 20, 21, 10, 9, 25, 11	13753	11, 5, 15, 14, 13, 12, 7, 6, 19, 21, 20, 10, 9, 25, 1, 8, 22, 4, 18, 17, 3, 2, 16, 26, 31, 27, 30, 28, 24, 29, 23

TABLE 4.5: A comparison between our method and the reset method for the evaluation of the shortest path on ‘ulyssess22’ for the random deletion of nodes. (Continuing with the same pheromone level yielded in table 4.3)

Total number of nodes (initially 31)	Particular Node Removed	Our Proposed Method		Reset Method	
		Path Length	Path	Path Length	Path
31	None	13485	23,27,29,24,28,26,30,56,7,12,13,14,15,1,8,16,22,4,18,17,2,3,19,20,21,10,9,25,11	13753	11,5,15,14,13,12,7,6,19,21,20,10,9,25,1,8,22,4,18,17,3,2,16,26,31,27,30,28,24,29,23
30	18	13166	11,5,15,14,13,12,7,6,18,19,20,10,9,24,16,1,8,21,17,2,3,4,25,30,26,27,29,23,28,22	13822	11,5,6,7,12,13,14,15,20,19,18,10,9,24,1,8,21,4,17,2,3,16,25,30,26,27,29,23,28,22
29	9	13237	3,2,16,4,20,1,8,15,12,11,13,7,6,17,18,19,9,23,10,5,14,24,29,25,26,28,22,27,21	13599	21,26,28,22,27,25,29,24,1,8,20,4,16,2,3,15,12,11,13,7,6,14,5,9,17,18,19,23,10
28	25	13547	10,5,6,7,11,12,13,14,1,8,20,4,16,2,3,15,18,19,17,9,23,24,28,25,27,22,26,21	14577	10,5,6,7,17,18,19,9,23,12,11,13,14,1,8,20,4,16,2,3,15,24,28,25,27,22,26,21
27	12	13559	10,5,6,7,11,12,13,1,8,19,4,15,2,3,14,17,18,16,9,22,23,27,24,26,21,25,20	14003	10,6,7,16,17,18,9,22,11,12,13,5,23,1,8,19,4,15,2,3,14,27,24,26,21,25,20
26	15	13420	19,20,24,23,25,26,22,1,8,18,4,2,3,14,11,12,7,6,13,5,9,15,16,17,21,10	13252	19,20,24,23,25,26,22,1,8,18,4,2,3,14,11,12,7,6,13,5,16,17,15,9,21,10
25	5	12932	18,19,23,24,22,25,21,12,11,10,6,5,8,14,15,16,1,7,17,4,2,3,13,20,9	13075	18,19,23,22,24,25,21,1,7,13,17,4,2,3,6,5,10,11,12,16,15,14,8,20,9
24	21	12381	3,2,4,17,1,7,13,10,11,12,5,6,16,15,14,8,20,9,24,21,23,19,22,18	12381	3,2,4,17,1,7,13,10,11,12,5,6,16,15,14,8,20,9,24,21,23,19,22,18
23	13	12495	3,2,4,16,7,1,11,10,6,5,12,14,15,13,8,19,9,23,20,22,18,21,17	12566	3,2,4,16,1,7,11,10,6,5,12,13,14,15,8,19,9,23,20,22,18,21,17

Stagnation could not occur in the experiments because of the large number of varying solutions generated per cycle. So in order to put a stopping criterion to our approach, we used the method called the Pocket Algorithm [7]. As shown in Table 4.6, we apply this approach to another graph that initially has 6 nodes, and the graph is made to change dynamically. For 10 nodes, with use of the reset method, we get the convergence after 28 cycles (Simple ACO has stagnation). After using our proposed method, we get the minimum shortest path after 9 cycles, and for 9 further cycles, it remains constant. But after the 18th cycle, we get inferior solutions for the next 9 cycles and then get the same minimum shortest path constantly for the next 22 cycles. So we have two options for the convergence: either we can take the convergence, after 9(inferior solutions)+9=18 cycles (better than 28 cycles), or we take the convergence after, 18+10(inferior solutions) +22=50 cycles (costlier than 28 cycles) as shown in table 4.6. So by choosing the prior semi-stable pocket the solution becomes computationally inexpensive.

The complexity of the ant-cycle algorithm is $O(C.n^3)$ after C cycles. In fact step 1 is $O(n^2+n)$, step 2 is $O(n)$, step 3 is $O(n^3)$, step 4 is $O(n^3)$, step 5 is $O(n^2)$, step 6 is $O(n^2)$. The step 7 will take $O(n^2)$ and step 8 and beyond will take $O(n)$. Here n is the number of nodes and C is the number of cycles the algorithm runs for.

Chapter 5

Conclusion and Future Work

Shortest path algorithms are extensively used in many real life applications. Shortest path algorithms are applied to automatically find directions between physical locations, such as driving directions. They are also used to find the optimal sequence of choices in a nondeterministic abstract machine to establish lower bounds on the time needed to reach a given state [30]. For example, if vertices represent the states of a puzzle like a Rubik's Cube and each directed edge corresponds to a single move or turn, shortest path algorithms can be used to find a solution that uses the minimum possible number of moves [30]. In a networking or telecommunications mindset, this shortest path problem is sometimes called the min-delay path problem and the objective is to minimize routing distances between two communicating systems.

Modifications to our methodology can be made in order to treat non-fully connected graphs. A non-fully connected graph can be visualized as a fully connected graph with edge weights of infinite weight. If a vertex is independent, that is, the vertex is not connected to another vertex in the graph; we can connect these two vertices with an edge having an infinitely large edge weight. This way we can connect all the vertices to each other to make it a fully connected graph. When we apply the algorithm, the ants may traverse these edges to complete a legal tour, but as the amount of pheromone we lay on the path is inversely proportional to edge length, the pheromone laid on it will be zero. As visibility is defined in the probability selection of the path, the ants would take up the paths that do not have infinite weight.

So after a cycle, we will have n tours, whenever we get a solution of infinite weight it would mean that an ant was not able to find a legal tour. In addition if all the ants come up with infinitely large tours, then there exists no legal tour in that graph. However, more analysis is required to conclude whether or not a legal tour exists in a graph or not.

Our methodology is inspired by the Ant System in order to find the shortest tour. The algorithm mainly works for fully connected undirected graphs in a dynamically changing environment. From this algorithm we received promising results for the addition of nodes and marginally better results for the removal of nodes (see Table 4.5). The performance of the method is unaffected by the quantity of nodes added or removed in a single instance. So whenever a new node is added, a fully connected graph is made. The ants have to traverse through this node in order to complete a legal tour. So the ants make use of the probability function and deposit pheromone based on the total tour length they have covered. So in a dynamically changing graph whenever a new node is added, ants have to travel through this new node, in order to form a legal tour, and thus a new shortest tour traversing through this node can be found easily.

On the other hand, when a node is removed from the graph, all the edges connecting this node to other nodes are also removed. But the ants are still forced to use the pheromone level values, unknowing to the fact that a node is removed. So the probability function will force these ants to traverse on the edges with higher values of pheromone, in order to complete a legal tour.

So now each ant will travel a different distance compared to what they used to do in the previous cycle, as they will now modify the pheromone levels based on the path they found.

So whenever the node is removed, the algorithm tries to normalize the pheromone values in the graph, and then proceeds further to find a new shortest path. Thus we believe this normalization of pheromone values is what causes the algorithm to perform slowly when compared to its performance to the reset method and also when compared to its performance, when a new node is added.

This method gives worse solutions for the control parameter $\alpha \geq 1$ as compared to $\alpha > 1$ used in the Ant System [14]. We performed experiments on these control parameters, as α controls the probability of the selection of node based on the pheromone levels of that vertex and β controls the probability for the weight of the edge that has to be traversed into order to reach the next node. So both factors play an important role for making the selection of the next node to traverse, but a proper balance of these parameters is required. We ran our algorithm for various values of α and β . From the analysis we conclude that the best results in our methodology are achieved with α as 0.5 β as 6.

Convergence is obtained after a large number of cycles or in non-regular intervals of the cycles, with the same path lengths of different frequencies. Our method works effectively in such situations. It finds the appropriate solution in a fewer number of cycles once the semi-stable convergence is achieved (path length remains constant with respect to time). This modification in

pocket algorithm helped to achieve the termination condition faster rather than actually waiting for the method to converge on a solution.

More modifications in the ACO can be adopted to treat graphs with negative weights or another approach can be to partition a large graph into smaller sub graphs and apply our methodology invaluablely and later combine their solutions. The scope also lies to use these ant heuristic algorithms with other algorithms present in the class of swarm intelligence to solve even more complex problems.

References

- 1) [Online].Available: <http://elib.zib.de/pub/mp-testdata/tsp/tsplib/tsp/index.html> maintained by Zuse Institute, Berlin.
- 2) Bullnheimer, B., Hartl, R. F., & Strauss, C. "A New Ant based version of Ant system: A computational Study", Central European Journal for Operations Research and Economics, 7(1), 25-38. 1999.
- 3) Bullnheimer. B., Hartl. R. F., Strauss. C., "A New Rank Based Version of the Ant System – A computational Study", Institute of Management Science, University of Vienna, Austria, Tech Rep., 1997.
- 4) Chengming Qi, "A Modified Ant Algorithm for Solving the Vehicle Routing Problem", 2007 International Conference on Intelligent Systems and Knowledge Engineering (ISKE 2007), October 2007.
- 5) D.E.Goldberg, "Genetic Algorithms in Search, Optimization & Machine Learning". Addison-Wesley, Reading, MA. 1989.
- 6) E.W Dijkstra. "A Note on Two Problems in Connexion with Graphs." Numerische Mathematik, Vol. 1, pp. 269-271, 1959.
- 7) Eyckelhof, C.J., Snoek, M.: "Ant System for a Dynamic TSP. Ants Caught in Traffic Jam". ANTS 2002, LNCS 2463, pp- 88-99, 2002.
- 8) Fidanova S., "3D HP Protein Folding Problem Using Ant Algorithm", In proc. of BioPS International Conference, Sofia, Bulgaria, pp.III.19-III.26,2006.
- 9) Gallant, SI, "Perceptron Based Learning Algorithms", IEEE Transactions on Neural Networks, 1(2), 179-191, 1990.
- 10) Gengqian Liu; Tiejun Li; Yuqing Peng; Xiangdan Hou, "The Ant algorithm for solving robot path planning, Third International Conference on Information Technology and Applications, ICITA 2005. Volume 2, pp.-25-27, 4-7 July 2005.
- 11) Gülüzar KEKEÇ, Nejat YUMUŞAK, Numan ÇELEBİ, "Data Mining and Clustering With Ant Colony Op Proceedings of 5th International Symposium on Intelligent Manufacturing Systems, pp. 1178-1190, May 29-31, 2006.
- 12) Guntsch, M., & Middendorf, M., "Applying Population Based ACO to Dynamic Optimization Problems Ant Algorithms", Proceedings of Third International Workshop ANTS 2002, Springer Verlag, LNCS 2463, pp.111-122, Brussels, Belgium, 2002.

- 13) Guntsch, M., Middendorf, M., & Schmeck, H. "An Ant Colony Optimization approach to Dynamic TSP". Proceedings of the Genetic and Evolutionary Computation Conference(GECCO-2001),pp 860–867, San Francisco, California, USA, 7-11 July,2001.
- 14) Ke Deng and Xiaofang Zhou, "Expansion-Based Algorithms for Finding Single Pair Shortest Path on Surface", in Proceedings of 4th International Workshop on Web and Wireless Geographical Information Systems (W2GIS2004), (LNCS 3428), pages 151-163, Springer, Seoul, Korea., November 2004.
- 15) Ke Deng and Xiaofang Zhou, "Expansion-Based Algorithms for Finding Single Pair Shortest Path on Surface", in Proceedings of 4th International Workshop on Web and Wireless Geographical Information Systems (W2GIS2004), (LNCS 3428), pages 151-163, Springer, Seoul, Korea., November 2004.
- 16) M. Dorigo, V. Maniezzo, and A. Colorni., "Ant system: Optimization by a colony of cooperating agents". IEEE Transactions on Systems, Man and Cybernetics - Part B, 26(1), pp. 29–41, 1996.
- 17) M. Guntsch and M. Middendorf. "Pheromone modification strategies for ant algorithms applied to dynamic TSP". In E.J.W. Boers et al., editor, Application of evolutionary computing: Proceedings of Eco Workshops 2001.
- 18) Marco Dorigo and Thomas Stützle "Ant Colony Optimization", Prentice-Hall of India Private Limited, 2005.7.2, pp .263-271.
- 19) Rong Zhou, Eric A. Hansen, "A Breadth First Approach for Memory-Efficient Graph Search", 21st National Conference on Artificial Intelligence(AAAI-06), Boston, MA.,July,2006.
- 20) Stützle, T. and Hoos, H.H. "Improvements on the ant system: Introducing MAX (MIN) ant system". Proceedings of the International Conference on Artificial Neural Networks and Genetic Algorithms, Springer-Verlag, pp 245–249, 1997.
- 21) Stützle, T. and Hoos, H.H. "MAX-MIN Ant System. Future Generation Computer Systems Journal", 16(8), pp. 889–914, 2000.
- 22) Stützle, T. and Hoos, H.H., "The MAX-MIN Ant System and local search for the Traveling salesman problem", Proceedings of the 1997 IEEE International Conference on Evolutionary Computation (ICEC'97), IEEE Press, pp. 309–314, 1997.
- 23) Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest and Clifford Stein, "Introduction to Algorithms", McGraw-Hill, 2001.

- 24) Vijay V. Vazirani, Approximation Algorithms, Springer-Verlag, 2001.
- 25) Xia Liu, Huan Qi, Yingchun Chen, "Optimization of Special Vehicle Routing Problem Based on Ant ICIC (2), pp. 1228-1233, 2006.
- 26) [Online].Available: [http://en.wikipedia.org/wiki/Dynamic_problem_\(algorithms\)](http://en.wikipedia.org/wiki/Dynamic_problem_(algorithms)), maintained by Wikipedia under Dynamic problems
- 27) [Online].Available: http://en.wikipedia.org/wiki/Travelling_salesman_problem maintained by Wikipedia under Travelling Salesman problem.
- 28) Holm, J., K. de Lichtenberg, and M. Thorup. "Poly-logarithmic deterministic fully dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and Bi-connectivity". In Proceedings of the 30th Annual ACM Symp, On Theory of Computing. pp 78-89, 1998.
- 29) [Online].Available: http://en.wikipedia.org/wiki/Simulated_annealing, maintained by Wikipedia under Simulated annealing.
- 30) [Online].Available: http://en.wikipedia.org/wiki/Shortest_path_problem, maintained by Wikipedia under Shortest path problem.
- 31) [Online].Available: <http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/DOC.PS>, maintained by Universität Heidelberg.
- 32) [Online].Available: http://en.wikipedia.org/wiki/Haversine_formula, maintained by Wikipedia under Haversine Formula.
- 33) [Online].Available: <http://www.mathworks.com/matlabcentral/fileexchange/9612-traveling-salesman-problem-tsp-using-simulated-annealing>, maintained by MATLAB Central.
- 34) [Online].Available: http://www.eng.uwaterloo.ca/~sjayaswa/projects/MSCI703_project.pdf maintained by University of Waterloo.