A New Local Search Based ACO Algorithm for Solving Combinatorial Optimization Problems

By

Md. Rakib Hassan
Department of Computer Science and Engineering
Bangladesh University of Engineering and Technology
June 2007

Submitted to

Bangladesh University of Engineering and Technology

In partial fulfillment of the requirements for
Master of Science in Engineering (Computer Science and Engineering) degree.
The thesis titled “A New Local Search Based Ant Colony Optimization Algorithm for Solving Combinatorial Optimization Problems” submitted by Md. Rakib Hassan, Roll No. 040505023F, Session April 2005 has been accepted as satisfactory in partial fulfillment of the requirements for the degree of Master of Science in Engineering (Computer Science and Engineering) held on June 05, 2007.

BOARD OF EXAMINERS

1. Dr. Md. Monirul Islam
   Associate Professor
   Department of CSE
   BUET, Dhaka–1000
   Chairman (Supervisor)

2. Dr. Muhammad Masroor Ali
   Professor and Head
   Department of CSE
   BUET, Dhaka–1000
   Member (Ex–officio)

3. Dr. Md. Mostofa Akbar
   Associate Professor
   Department of CSE
   BUET, Dhaka–1000
   Member

4. Dr. Md. Humayun Kabir
   Assistant Professor
   Department of CSE
   BUET, Dhaka–1000
   Member

5. Dr. Mohammad Zahidur Rahman
   Associate Professor and Head
   Department of CSE
   Jahangirnagar University
   Savar, Dhaka
   Member (External)
DECLARATION

I, hereby, declare that the work presented in this thesis is the outcome of the investigation performed by me under the supervision of Dr. Md. Monirul Islam, Associate Professor, Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology, Dhaka. I also declare that no part of this thesis and thereof has been or is being submitted elsewhere for the award of any degree or diploma.

Signature

Md. Rakib Hassan

(Md. Rakib Hassan)
Candidate
To

My Beloved Family
ACKNOWLEDGEMENTS

Firstly, I would like to express my honor and gratefulness to Dr. Md. Monirul Islam, Associate Professor, Department of Computer Science and Engineering, Bangladesh University of Engineering and Technology, for his continuous support, advice and care. His endless patience, scholarly guidance, continual encouragement, constant and energetic supervision, constructive criticism, valuable advice, reading many inferior drafts and correcting them at all stage have made it possible to complete this thesis.

Secondly, I would like to thank Dr. Muhammad Masroor Ali, Professor and Head, Department of CSE, BUET for his fruitful advice and suggestions.

Thirdly, I would like to express my gratitude to all the members of the graduate committee for their valuable suggestions.

I would also like to thank all the faculty members and staff of Department of CSE, BUET, for their support and cooperation.

Finally, I would like to express my deep respect and gratitude to my parents, sister and my wife, without whose sacrifice, support and encouragement, this work would have been impossible.
ABSTRACT

ACO algorithms are a new branch of swarm intelligence. ACO algorithms have been introduced in the last decade. They have been applied to solve different combinatorial optimization problems successfully. Their performance is very promising when they solve small problem instances. When they try to solve large problems, their time complexity increases and their solution quality decreases. They get stuck in local optima due to improper balancing of exploration and exploitation of the search space. When their solution quality is tried to improve using local search, their time complexity is increased. When time complexity is tried to reduce, they produce poor quality solutions. So, it is crucial to reduce the time requirement and at the same time to increase the solution quality produced by the algorithms for solving large combinatorial optimization algorithms.

This thesis introduces Local Search based Ant Colony Optimization algorithm (LSACO), a new ACO algorithm to solve large combinatorial optimization problems. The basis of LSACO is to apply an adaptive local search method to improve the solution quality. Adaptive local search automatically determines the number of edges to exchange during the run time of the algorithm. LSACO also applies pheromone updating rule and constructs solutions in a new way so as to decrease the convergence time. LSACO makes it possible to produce very good quality solutions for large problem instances in a short time.

Performance of LSACO has been evaluated on a number of benchmark combinatorial optimization problems and results are compared with several existing ACO algorithms. Experimental results show that LSACO performs better optimization with a higher rate of convergence for most of the problems in a reasonable amount of time.
# TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acknowledgements</td>
<td>ii</td>
</tr>
<tr>
<td>Abstract</td>
<td>ii</td>
</tr>
<tr>
<td>Table of Contents</td>
<td>iii-iv</td>
</tr>
<tr>
<td>List of Figures</td>
<td>v</td>
</tr>
<tr>
<td>List of Tables</td>
<td>vi</td>
</tr>
<tr>
<td>List of Symbols</td>
<td>vii</td>
</tr>
<tr>
<td><strong>Chapter 1: Introduction</strong></td>
<td></td>
</tr>
<tr>
<td>1.1 Introduction</td>
<td>1</td>
</tr>
<tr>
<td>1.2 Literature Survey</td>
<td>2</td>
</tr>
<tr>
<td>1.3 Aim of the Thesis</td>
<td>6</td>
</tr>
<tr>
<td>1.4 Thesis Organization</td>
<td>7</td>
</tr>
<tr>
<td><strong>Chapter 2: Background</strong></td>
<td>8-30</td>
</tr>
<tr>
<td>2.1 Introduction</td>
<td>8</td>
</tr>
<tr>
<td>2.2 Combinatorial Optimization Problem</td>
<td>8</td>
</tr>
<tr>
<td>2.3 Solving Combinatorial Optimization Problems</td>
<td>11</td>
</tr>
<tr>
<td>2.3.1 Exact Algorithms</td>
<td>11</td>
</tr>
<tr>
<td>2.3.2 Approximate Algorithms</td>
<td>12</td>
</tr>
<tr>
<td>2.3.2.1 Tour Construction Algorithms</td>
<td>12</td>
</tr>
<tr>
<td>2.3.2.2 Tour Improvement Algorithms</td>
<td>12</td>
</tr>
<tr>
<td>2.3.2.3 Composite Algorithms</td>
<td>13</td>
</tr>
<tr>
<td>2.4 ACO Metaheuristic</td>
<td>13</td>
</tr>
<tr>
<td>2.5 Ant System-The Basic ACO Algorithm</td>
<td>15</td>
</tr>
<tr>
<td>2.5.1 Artificial Ants</td>
<td>17</td>
</tr>
<tr>
<td>2.5.2 The AS Algorithm</td>
<td>18</td>
</tr>
<tr>
<td>2.5.3 Tour Construction of AS</td>
<td>19</td>
</tr>
<tr>
<td>2.5.4 Update of Pheromone Trails in AS</td>
<td>21</td>
</tr>
<tr>
<td>2.5.5 The Pseudo Code of AS Algorithm</td>
<td>22</td>
</tr>
<tr>
<td>2.5.6 Problems of AS</td>
<td>22</td>
</tr>
<tr>
<td>2.6 Ant Colony System</td>
<td>23</td>
</tr>
<tr>
<td>2.6.1 Differences with AS</td>
<td>23</td>
</tr>
<tr>
<td>2.6.2 The ACS Algorithm</td>
<td>24</td>
</tr>
<tr>
<td>2.7 Features of ACO Algorithm</td>
<td>27</td>
</tr>
<tr>
<td>2.8 Local Search</td>
<td>28</td>
</tr>
<tr>
<td>2.9 Conclusion</td>
<td>30</td>
</tr>
<tr>
<td><strong>Chapter 3: Proposed Algorithm</strong></td>
<td>31-46</td>
</tr>
<tr>
<td>3.1 Introduction</td>
<td>31</td>
</tr>
<tr>
<td>3.2 Overview of LSACO Algorithm</td>
<td>31</td>
</tr>
<tr>
<td>3.3 New Tour Construction Phase</td>
<td>33</td>
</tr>
<tr>
<td>3.4 New Local Pheromone Update</td>
<td>35</td>
</tr>
<tr>
<td>3.5 Adaptive Local Search</td>
<td>36</td>
</tr>
<tr>
<td>Figure</td>
<td>Figure Name</td>
</tr>
<tr>
<td>--------</td>
<td>------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>2.1</td>
<td>An example with real ants</td>
</tr>
<tr>
<td>2.2</td>
<td>An example with artificial ants</td>
</tr>
<tr>
<td>2.3</td>
<td>Flow chart of ACS algorithm</td>
</tr>
<tr>
<td>2.4</td>
<td>An example 2-opt move</td>
</tr>
<tr>
<td>2.5</td>
<td>An example 2.5-opt move</td>
</tr>
<tr>
<td>2.6</td>
<td>An Example 3-opt move</td>
</tr>
<tr>
<td>3.1</td>
<td>Flow chart of the LSACO Algorithm</td>
</tr>
<tr>
<td>3.2</td>
<td>Flowchart of the adaptive local search method</td>
</tr>
<tr>
<td>3.3</td>
<td>Edge exchanging process</td>
</tr>
<tr>
<td>3.4</td>
<td>Sharing common endpoints</td>
</tr>
<tr>
<td>3.5</td>
<td>Sets X and Y are disjoint</td>
</tr>
<tr>
<td>3.6</td>
<td>Try an untried edge</td>
</tr>
<tr>
<td>3.7</td>
<td>Replaced edges will not be added or added edges will not be deleted.</td>
</tr>
<tr>
<td>3.8</td>
<td>Cost(T2)&lt;Cost(T1)</td>
</tr>
<tr>
<td>4.1</td>
<td>Time requirement of LSACO with respect to problem size</td>
</tr>
<tr>
<td>4.2</td>
<td>Comparison of error rates of LSACO and other algorithms with respect to problem dimension</td>
</tr>
<tr>
<td>4.3</td>
<td>Comparison of required time to find the optimum solution</td>
</tr>
<tr>
<td>Table</td>
<td>Table Name</td>
</tr>
<tr>
<td>-------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>4.1</td>
<td>Performance of LSACO on some problems of Symmetric TSP over 15 trials</td>
</tr>
<tr>
<td></td>
<td>(Edge weight is in Geographical distance)</td>
</tr>
<tr>
<td>4.2</td>
<td>Performance of LSACO on some problems of Symmetric TSP over 15 trials</td>
</tr>
<tr>
<td></td>
<td>(Edge weight is in matrix form)</td>
</tr>
<tr>
<td>4.3</td>
<td>Performance of LSACO on some problems of Symmetric TSP over 15 trials</td>
</tr>
<tr>
<td></td>
<td>(Edge weight is in special form)</td>
</tr>
<tr>
<td>4.4</td>
<td>Performance of LSACO on some problems of Symmetric TSP over 15 trials</td>
</tr>
<tr>
<td></td>
<td>(Edge weight is in Euclidean distances in 2-D)</td>
</tr>
<tr>
<td>4.5</td>
<td>Performance of LSACO on some problems of Symmetric TSP</td>
</tr>
<tr>
<td></td>
<td>(Edge weight is in Euclidean distances in 2-D)</td>
</tr>
<tr>
<td>4.6</td>
<td>Performance of LSACO on some problems of Asymmetric TSP over 15 trials</td>
</tr>
<tr>
<td></td>
<td>(Edge weight is in matrix form)</td>
</tr>
<tr>
<td>4.7</td>
<td>Performance of LSACO for Hamiltonian Cycle Problems (over 15 trials)</td>
</tr>
<tr>
<td>4.8</td>
<td>Experimental Results of LSACO on some larger TSP problems (over 15 trials)</td>
</tr>
<tr>
<td>4.9</td>
<td>Comparison between LSACO and ACS-3-opt on some symmetric TSP problems</td>
</tr>
<tr>
<td></td>
<td>(over 15 trials)</td>
</tr>
<tr>
<td>4.10</td>
<td>Comparison between LSACO and ACS-3-opt on some asymmetric TSP problems</td>
</tr>
<tr>
<td></td>
<td>(over 15 trials)</td>
</tr>
<tr>
<td>4.11</td>
<td>Comparison between LSACO and STSP-GA on some symmetric TSP problems (over 15</td>
</tr>
<tr>
<td></td>
<td>trials)</td>
</tr>
<tr>
<td>4.12</td>
<td>Comparison between LSACO and ATSP-GA on some asymmetric TSP problems</td>
</tr>
<tr>
<td></td>
<td>(over 15 trials)</td>
</tr>
<tr>
<td>4.13</td>
<td>Comparison of LSACO with MMAS on symmetric problems of TSP (over 15 trials)</td>
</tr>
</tbody>
</table>
List of Symbols

$\alpha$  relative importance of pheromone trail, $0 < \alpha < 1$

$\beta$  relative importance of the distance or heuristic information, $\beta > 0$

$\eta_0$  heuristic value

$m$  the number of ants

$q$  a random number uniformly distributed in $[0 .. 1]$

$q_0$  a parameter $[0 \leq q_0 \leq 1]$

$\tau_0$  initial pheromone

$\xi$  pheromone decay parameter, $0 < \xi < 1$
1.1 Introduction

Artificial intelligence [1] is now a well-known term than it was a few years ago. At present, it has been applied in many objects or in software so that the objects or software can take decisions themselves and solve difficult problems intelligently without the help of human beings. For this purpose, many intelligent algorithms are present now. Among them, swarm intelligence [2] is of demand. It emerges from the collective behavior of the social insect colonies and other animal societies.

Swarm intelligence exhibits a number of interesting properties such as flexibility, robustness, decentralization and self-organization. It has attracted more attention from computational intelligence researchers. Implementation of optimization and control algorithms based on swarm intelligence such as Ant Colony Optimization (ACO) [3] is of growing interest.

ACO algorithms have been applied to combinatorial optimization problems [4] such as traveling salesman problem [5], job-shop scheduling [6] and other sophisticated problems [7-16]. ACO algorithms are currently state-of-the-art for solving the sequential ordering problem [7], the resource constraint project scheduling problem [8] and the open shop scheduling problem [9].

In ACO, a number of artificial ants [2] are given a set of simple rules that take inspiration from the behavior of real ants. Artificial ants are then left free to move on an appropriate graph representation of the considered problem. They probabilistically build a solution to the given problem and then deposit on the graph some artificial pheromones that bias the probabilistic solution construction activity of future ants. The amount of pheromone deposited and the way it is used to build solutions are such that the overall search process is biased toward the generation of approximate solutions of better quality.
The performance of ACO algorithms in solving difficult combinatorial optimization problems is very promising [16]. But when the problem size grows, the solution quality of ACO algorithms decreases [17-21]. And the time requirement to solve these large problems also increases. To improve the solution quality and to reduce the time requirement, several ACO algorithms have been proposed so far. Among these ACO algorithms, Ant Colony System (ACS) [22] is the best performing one.

ACS performs very well compared to other ACO algorithms. It produces very good results for small problems in a reasonable time. But when it is applied to large problems, its performance also decreases. Because the existing ACO algorithms get stuck in local optima in a certain stage of the search process. Some ACO algorithms stuck in local optima in the early stages of the search process and some get stuck in later stages of the search process. Besides the problem of local optima, the existing ACO algorithms require huge time to produce near optimum results for large problem instances.

1.2 Literature Survey

In the early 1990s, ACO was introduced by M. Dorigo and colleagues as a novel nature-inspired metaheuristic for the solution of hard combinatorial optimization problems. ACO belongs to the class of metaheuristics which are approximate algorithms used to obtain good enough solutions to hard combinatorial optimization problems in a reasonable amount of computation time. Other examples of metaheuristics are tabu search [23, 24], simulated annealing [25] and evolutionary computation [26].

The first ACO algorithm called Ant System (AS) [2] was published in 1991 by Dorigo. It was applied to the traveling salesman problem [27-30]. Starting from AS, several improvements of the basic algorithm have been proposed. Typically, these improved algorithms have been tested again on the TSP. All these improved versions of AS have in common a stronger exploitation of the best solutions found to direct the ants' search process; they mainly differ in some aspects of the search control.
Besides AS, other existing ACO algorithms are Elitist Ant System (Dorigo, 1992) [31], Ant-Q (Dorigo and Gambardella, 1996) [32-33], Ant Colony System (Dorigo and Gambardella, 1997) [22], Max-Min Ant System (Stutzle, 1997) [34], Rank-based Ant System (Bullnheimer et al., 1999) [35], Approximate Nondeterministic Tree Search (Maniezzo, 1999, 2000) [36-37] and Hyper-cube Ant System (Blum & Dorigo, 2001, 2004) [38-39].

In ant system, three different versions were proposed. These were called ant-density, ant-quantity and ant-cycle [2]. In the ant-density and ant-quantity versions, the ants updated the pheromone directly after a move from one node to an adjacent node. In the ant-cycle version, the pheromone update was only done after all the ants had constructed the tours and the amount of pheromone deposited by each ant was set to be a function of the tour quality. Among these three versions, only ant-cycle is used for AS and the other two versions were abandoned because of their inferior performance. The two main phases of the AS algorithm constitute the ant’s solution construction and the pheromone update. The relative performance of AS when compared to other metaheuristics tends to decrease dramatically as the size of the test-instance increases.

A first improvement on the initial AS is the elitist strategy for AS. It is called the Elitist AS. The idea is to provide strong additional reinforcement to the edges belonging to the best tour found since the start of the algorithm. This tour is called the best-so-far tour. This additional feedback to the best-so-far tour can be viewed as additional pheromone deposited by an additional ant called best-so-far ant. In Elitist AS, pheromone evaporation is implemented as in AS.

Rank-based AS (AS_{rank}) is another improvement over AS. In AS_{rank}, each ant deposits an amount of pheromone that decreases with its rank. As in Elitist AS, the best-so-far ant always deposits the largest amount of pheromone in each iteration. Before updating the pheromone trails, the ants are sorted by increasing tour length. The quantity of pheromone on ant deposits is weighted according to the rank r of the ant. Ties can be solved randomly. In each iteration, only the (w-1) best-ranked ants and
the ant that produced the best-so-far tour are allowed to deposit pheromone, where $w$ is the weight. The best-so-far tour gives the strongest feedback. AS\textsubscript{rank} performs slightly better than Elitist AS and significantly better than AS. It has some additional computations to sort the ants in each iteration for ranking the ants. Since the number of ants was equal to the number of nodes in the problem, sorting in each iteration is an additional overhead.

Ant-Q is an algorithm that is based on AS and Q-learning algorithms. Q-learning is a reinforcement learning technique that works by learning an action-value function that gives the expected utility of taking a given action in a given state and following a fixed policy thereafter. Reinforcement learning differs from the supervised learning problem in that correct input/output pairs are never presented, nor sub-optimal actions explicitly corrected. The strength of Q-learning is that it is able to compare the expected utility of the available actions without requiring a model of the environment. Although having a good performance, Ant-Q was abandoned for the equally good but simpler Ant Colony System.

Max-Min Ant System (MMAS) introduces four main modifications with respect to AS. Firstly, 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. Secondly, to avoid local optima, a second modification introduced by MMAS is that it limits the possible range of pheromone trail values to the interval $[\tau_{\text{min}}, \tau_{\text{max}}]$. Thirdly, 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. And last of all, 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.

Ant Colony System (ACS) 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 edges belonging to the best-so-far tour. Third, each time an ant uses an edge $(i,j)$ to move from node $i$ to node $j$, it removes
some pheromone from the edge to increase the exploration of alternative paths. There are some similarities between MMAS and ACS: they both use pheromone trail limits, although these are explicit in MMAS and implicit in ACS.

Approximate Nondeterministic Tree Search (ANTS) is an ACO algorithm that exploits ideas from mathematical programming. In particular, ANTS computes lower bounds on the completion of a partial solution to define the heuristic information that is used by each ant during the solution construction. The name ANTS derives from the fact that the proposed algorithm can be interpreted as an approximate nondeterministic tree search since it can be extended in a straightforward way to a branch & bound procedure. Apart from the use of lower bounds, ANTS also introduces two additional modifications with respect to AS: the use of a novel action choice rule and a modified pheromone trail update rule. The use of lower bounds to compute the heuristic information has the advantage in that otherwise feasible moves can be discarded if they lead to partial solutions whose estimated costs are larger than the best-so-far solution. A disadvantage is that the lower bound has to be computed at each single construction step of an ant and therefore a significant computational overhead might be incurred.

The hyper-cube framework for ACO was introduced by Blum, Roli, & Dorigo (2001) to automatically rescale the pheromone values in order for them to lie always in the interval [0, 1]. This choice was inspired by the mathematical programming formulation of many combinatorial optimization problems, in which solutions can be represented by binary vectors. In such a formulation, the decision variables, which can assume the values {0, 1}, typically correspond to the solution components as they are used by the ants for solution construction. A solution to a problem then corresponds to one corner of the n-dimensional hyper-cube, where n is the number of decision variables. The hyper-cube framework is a complex method compared to the existing ACO algorithms and so, it has not been applied extensively like the other ACO algorithms.
1.3 Aim of the Thesis

Current ACO algorithms have shown very promising results in solving combinatorial optimization problems [40]. They can find good quality solutions in a reasonable time. But all the existing ACO algorithms have been applied in small-sized problems. When they are applied in large problems, their performance degrades very much. Their solution quality decreases and their time complexity also increase noticeably.

The existing ACO algorithms have applied 2-opt and 3-opt local search strategy [22, 38] to improve performance. But all these search strategies have to predefine the number of edges to exchange. Predefining a fixed number of edges is a major drawback because it is difficult to know in advance what to use in achieving the best compromise between running time and quality of solution. If \( \Delta \) is assumed to be the number of edges to exchange, then if \( \Delta \) increases, the number of operations to test all \( \Delta \)-exchanges increases rapidly with problem dimension. Existing algorithms cannot be applied for large problems because the time to check and exchange edges grows exponentially.

Considering all the above mentioned problems and findings we have decided our objectives as:

- To develop a new ACO algorithm that will automatically determine the number of edges to be exchanged for finding good quality solutions.
- To introduce a number of control parameters that will prevent premature convergence and will increase the probability of finding the global optima in a reasonable time.
- To reduce the time complexity of the ACO algorithm so that it can be applied to large problems.
- To evaluate and compare the performance of the new algorithm with the existing ACO algorithms on different benchmark optimization problems.
1.4 Thesis Organization

The remaining chapters of this thesis are organized as follows:

- **Chapter 2** provides background material for the rest of the thesis. Combinatorial optimization problems and their solving techniques are described first. Then ACO metaheuristic is discussed. Thereafter, the Ant System, which is the first algorithm of ACO metaheuristic, is discussed with the nature of the artificial ants, its pheromone model, solution construction technique, pheromone updating rules and the solution quality obtained. Local search is then described with its working method and its capability to improve the performance of the ACO algorithms in solving combinatorial optimization problems. Then the Ant Colony System, the best performing ACO algorithm, is also described in short as our proposed algorithm is based on this ant colony system algorithm.

- **Chapter 3** presents the proposed algorithm LSACO: Local Search based ACO, which is the main contribution of this thesis. Advantages of LSACO over other conventional approaches are enlisted first. Algorithm of LSACO is then elaborated; detailed description of different processes and methods used in LSACO are then described.

- **Chapter 4** presents a detailed experimental evaluation of LSACO. In the reported experiments, LSACO is applied to solve combinatorial optimization problems. This chapter evaluates LSACO’s performance on several well-known benchmark optimization problems. Types and sources of data, experimental details, results, analysis and comparison with other ACO algorithms are described.

- **Chapter 5** presents the contribution and limitations of the research and the proposed future research tasks aimed at addressing the limitations.
Chapter 2
Background

2.1 Introduction

ACO algorithms were first described as Ant System [2]. After that, many modifications and improvements were proposed. These ACO algorithms have been applied in different combinatorial optimization problems. This chapter describes the background necessary to understand the ACO algorithms. At first, combinatorial optimization problems are briefly described with their types of solution techniques. Ant System algorithm which is the first of its class of ACO metaheuristic, has been applied to solve the combinatorial optimization problems. This basic algorithm is described at first. Then the best performing ACO algorithm called Ant Colony System [22] is described in short. Our proposed algorithm uses this ant colony system as the basic framework of ACO.

2.2 Combinatorial Optimization Problem

Combinatorial optimization problems [4, 40] involve finding values for discrete variables such that the optimal solution with respect to a given objective function is found. A combinatorial optimization problem is either maximization or a minimization problem which is associated a set of problem instances. The term problem refers to the general question to be answered, usually having several parameters or variables with unspecified values. The term instance refers to a problem with specified values for all the parameters.

More formally, an instance of a combinatorial optimization problem $\Pi$ is a triple $(S, f, \Omega)$, where $S$ is the set of candidate solutions, $f$ is the objective function which assigns an objective function value $f(s)$ to each candidate solution $s \in S$, and $\Omega$ is a set of constraints. The solutions belonging to the set $\tilde{S} \subseteq S$ of candidate
solutions that satisfy the constraints $\Omega$ is called feasible solutions. The goal is to find a globally optimal feasible solution $s^*$. 

For minimization problems this consists in finding a solution $s^* \in \tilde{S}$ with minimum cost, that is, a solution such that $f(s^*) \leq f(s)$ for all $s \in \tilde{S}$; for maximization problems one searches for a solution with maximum objective value, that is, a solution with $f(s^*) \geq f(s)$ for all $s \in \tilde{S}$. It should be noted that an instance of a combinatorial optimization problem is typically not specified explicitly by enumerating all the candidate solutions (i.e., the set $S$) and the corresponding cost values, but is rather represented in a more concise mathematical form (e.g., shortest-path problems are typically defined by a weighted graph).

A straightforward approach to the solution of combinatorial optimization problems would be exhaustive search, that is, the enumeration of all possible solutions and the choice of the best one. Unfortunately, in most cases, such a naive approach becomes rapidly infeasible because the number of possible solutions grows exponentially with the instance size $n$, where the instance size can be given, for example, by the number of binary digits necessary to encode the instance.

For some combinatorial optimization problems, deep insight into the problem structure and the exploitation of problem-specific characteristics allow the definition of algorithms that find an optimal solution much quicker than exhaustive search does. In other cases, even the best algorithms of this kind cannot do much better than exhaustive search.

Finding an optimal solution of a combinatorial optimization problem is difficult. This difficulty can be measured by the worst-case complexity. Worst-case complexity can be explained as: a combinatorial optimization problem $\Pi$ is said to have worst-case complexity $O(g(n))$ if the best algorithm known for solving $\Pi$ finds an optimal solution to any instance of $\Pi$ having size $n$ in a computation time bounded from above by $\text{const} \cdot g(n)$.
A combinatorial optimization problem $\Pi$ is solvable in polynomial time if the maximum amount of computing time necessary to solve any instance of size $n$ of $\Pi$ is bounded from above by a polynomial in $n$. If $k$ is the largest exponent of such a polynomial, then the combinatorial optimization problem is said to be solvable in $O(n^k)$ time.

Although some important combinatorial optimization problems have been shown to be solvable in polynomial time, for the great majority of combinatorial problems no polynomial bound on the worst-case solution time could be found so far. For these problems, the run time of the best algorithms known increases exponentially with the instance size and, consequently, so does the time required to find an optimal solution. A notorious example of such a problem is the TSP.

An important theory that characterizes the difficulty of combinatorial problems is that of NP-completeness. This theory classifies combinatorial problems in two main classes: those that are known to be solvable in polynomial time, and those that are not. The first are said to be tractable, the latter intractable.

The theory of NP-completeness distinguishes between two classes of problems of particular interest: the class $P$ for which an algorithm outputs in polynomial time the correct answer ("yes" or "no"), and the class NP for which an algorithm exists that verifies for every instance, independently of the way it was generated, in polynomial time whether the answer "yes" is correct. It is clear that $P \subseteq NP$ while nothing can be said on the question whether $P = NP$ or not.

A polynomial time reduction is a procedure that transforms a problem into another one by a polynomial time algorithm. The interesting point is that if problem $\Pi_A$ can be solved in polynomial time and problem $\Pi_B$ can be transformed into $\Pi_A$ via a polynomial time reduction, then also the solution to $\Pi_B$ can be found in polynomial time.

A problem is said to be NP-hard, if every other problem in NP can be transformed to it by a polynomial time reduction. Therefore, an NP-hard problem is at least as hard
as any of the other problems in NP. However, NP-hard problems do not necessarily belong to NP. An NP-hard problem that is in NP is said to be NP-complete. Therefore, the NP-complete problems are the hardest problems in NP: if a polynomial time algorithm could be found for an NP-complete problem, then all problems in the NP-complete class (and consequently all the problems in NP) could be solved in polynomial time. Until today, a large number of problems have been proved to be NP-complete, including the TSP.

2.3 Solving Combinatorial Optimization Problems

Two classes of algorithms are available for the solution of combinatorial optimization problems:

- Exact algorithms
- Approximate algorithms.

2.3.1 Exact algorithms

Exact algorithms are guaranteed to find the optimal solution and to prove its optimality for every finite size instances of a combinatorial optimization problem within an instance-dependent run time. Exact algorithms need, in the worst case, exponential time to find the optimum in the case of NP-hard problems. The application of exact algorithms to NP-hard problems in practice also suffers from a strong rise in computation time when the problem size increases, and often their use becomes infeasible.

For example, the most effective exact algorithms are cutting-plane or facet-finding algorithms [30]. These algorithms are quite complex, with codes on the order of 10,000 lines. These algorithms are also very demanding of computer power. For example, the exact solution of a symmetric traveling salesman problem with 2392 cities was determined over a period of more than 27 hours on a powerful supercomputer. It took roughly 3-4 years of CPU time on a large network of computers to determine the exact solution of the 7397-city problem.
2.3.2 Approximate algorithms

In contrast, the approximate algorithms obtain good solutions but do not guarantee that optimal solutions will be found. These algorithms are usually very simple and have (relative) short running times. Some of the algorithms give solutions that in average differ only by a few percent from the optimal solution. Therefore, if a small deviation from optimum can be accepted, it may be appropriate to use an approximate algorithm. The class of approximate algorithms may be subdivided into the following three classes:

- Tour construction algorithms
- Tour improvement algorithms
- Composite algorithms.

2.3.2.1 Tour Construction Algorithms

A simple example of a tour construction algorithm is the so-called nearest neighbor algorithm. Start in an arbitrary node. As long as there are nodes that have not yet been visited, visit the nearest node that still has not appeared in the tour. Finally, return to the first node. This approach is simple, but often too greedy. The first distances in the construction process are reasonably short, whereas the distances at the end of the process usually will be rather long. A lot of other construction algorithms have been developed to remedy this problem.

2.3.2.2 Tour Improvement Algorithms

Tour improvement algorithms are also called local search methods [3, 22]. Local search starts from some initial solution and repeatedly tries to improve the current solution by local changes. The first step in applying local search is the definition of a neighborhood structure over the set of candidate solutions. The neighborhood structure defines for each current solution the set of possible solutions to which the local search algorithms can move.
One common way of defining neighborhoods is via k-exchange moves that exchange a set of k components of a solution with a different set of k components. Its most basic version is often called iterative improvement, or sometimes hill-climbing or gradient-descent for maximization or minimization problems, respectively.

The local search algorithm searches for an improved solution within the neighborhood of the current solution. If an improving solution is found, it replaces the current solution and the local search is continued. These steps are repeated until no improving solution is found in the neighborhood and the algorithm terminates in local optima. A disadvantage of tour improvement is that the algorithm may stuck at very poor-quality local optima.

2.3.2.3 Composite Algorithms

The composite algorithms combine the features of tour construction and tour improvement algorithms. Tour construction algorithms are too greedy and tour improvement algorithms stuck in local optima. Combining these two algorithms, the solution quality is increased in composite algorithms. ACO algorithms can be loosely classified as composite algorithms. Because ACO algorithms first construct tours using some probabilistic rules and then use local search to improve the solution quality.

2.4 ACO Metaheuristic

Combinatorial optimization problems are often easy to state but very difficult to solve. Many of the problems arising in applications are NP-hard, that is, it is strongly believed that they cannot be solved to optimality within polynomially bounded computation time. Hence, to practically solve large instances one often has to use approximate methods which return near-optimal solutions in a relatively short time. Algorithms of this type are loosely called heuristics. They often use some problem-specific knowledge to either build or improve solutions.
Recently, many researchers have focused their attention on a new class of algorithms, called metaheuristics. A metaheuristic is a set of algorithmic concepts that can be used to define heuristic methods applicable to a wide set of different problems. The use of metaheuristics has significantly increased the ability of finding very high-quality solutions to hard, practically relevant combinatorial optimization problems in a reasonable time. Metaheuristic can also be defined as a master strategy that guides and modifies other heuristics to produce solutions beyond those that are normally generated in a quest for local optimality.

A particularly successful metaheuristic is inspired by the behavior of real ants. Starting with AS, a number of algorithmic approaches based on the very same ideas were developed and applied with considerable success to a variety of combinatorial optimization problems from academic as well as from real-world applications. ACO is a metaheuristic framework which covers the algorithmic approach mentioned above. The ACO metaheuristic has been proposed as a common framework or the existing applications and algorithmic variants of a variety of ant algorithms. Algorithms that fit into the ACO metaheuristic framework are called ACO algorithms. The application of ACO is particularly interesting for:

1. NP-hard problems which cannot be efficiently solved by more traditional algorithms
2. Dynamic shortest-path problems in which some properties of the problem's graph representation change over time concurrently with the optimization process
3. Problems in which the computational architecture is spatially distributed.

Besides ACO metaheuristic, other metaheuristics exist including simulated annealing, tabu search, guided local search, iterated local search, greedy randomized adaptive search procedures, and evolutionary computation. Several characteristics make ACO a unique approach: it is a constructive, population-based metaheuristic which exploits an indirect form of memory of previous performance. This combination of characteristics is not found in any of the other metaheuristics.
2.5 Ant System-The Basic ACO Algorithm

Ant System (AS) is the basic ACO algorithm and it was the first algorithm proposed in the class of ACO algorithms in 1991. The main characteristics of this model are positive feedback, distributed computation, and the use of a constructive greedy heuristic. Positive feedback accounts for rapid discovery of good solutions, distributed computation avoids premature convergence, and the greedy heuristic helps find acceptable solutions in the early stages of the search process.

In AS, the search activities are distributed over so-called "ants," that is, agents with very simple basic capabilities which, to some extent, mimic the behavior of real ants. One of the problems studied by ethologists was to understand how almost blind animals like ants could manage to establish shortest route paths from their colony to feeding sources and back. It was found that the medium used to communicate information among individuals regarding paths, and used to decide where to go, consists of pheromone trails. A moving ant lays some pheromone (in varying quantities) on the ground, thus marking the path by a trail of this substance.

While an isolated ant moves essentially at random, an ant encountering a previously laid trail can detect it and decide with high probability to follow it, thus reinforcing the trail with its own pheromone. The collective behavior that emerges is a form of autocatalytic behavior where the more the ants following a trail, the more attractive that trail becomes for being followed. The process is thus characterized by a positive feedback loop, where the probability with which an ant chooses a path increases with the number of ants that previously chose the same path.
Fig 2.1: An example with real ants: (a) Ants follow a path between points A and E, (b) An obstacle is interposed, ants can choose to go around it following one of the two different paths with equal probability, (c) On the shorter path more pheromone is laid down.

Consider for example the experimental setting shown in Fig. 2.1. There is a path along which ants are walking (for example from food source A to the nest E, and vice versa, see Fig. 2.1.a). Suddenly an obstacle appears and the path is cut off. So at position B the ants walking from A to E (or at position D those walking in the opposite direction) have to decide whether to turn right or left (Fig. 2.1.b). The choice is influenced by the intensity of the pheromone trails left by preceding ants.

A higher level of pheromone on the right path gives an ant a stronger stimulus and thus a higher probability to turn right. The first ant reaching point B (or D) has the same probability to turn right or left (as there was no previous pheromone on the two alternative paths). Because path BCD is shorter than BFD, the first ant following it will reach D before the first ant following path BHD (Fig. 2.1.c). The result is that an ant returning from E to D will find a stronger trail on path DCB, caused by the half of all the ants that by chance decided to approach the obstacle via DCBA and by the already arrived ones coming via BCD; they will therefore prefer (in probability) path DCB to path DFB.

As a consequence, the number of ants following path BCD per unit of time will be higher than the number of ants following BFD. This causes the quantity of pheromone on the shorter path to grow faster than on the longer one, and therefore the probability
with which any single ant chooses the path to follow is quickly biased towards the shorter one. The final result is that very quickly all ants will choose the shorter path. The AS model is derived from the study of real ant colonies.

2.5.1 Artificial ants

The AS uses artificial ants (ants for short). These ants behave almost similarly with the real ants. But there are also some differences. The properties that are possessed by the artificial ants other than the real ants are:

- Artificial ants have some memory,
- They are not be completely blind,
- They live in an environment where time is discrete.

Consider the graph of Fig. 2.2.a, which is a possible AS interpretation of the situation of Fig. 2.1.b. Suppose that the distances between D and F, between B and F, and between B and D, via C, are equal to 2, and let C be positioned half the way between D and B (see Fig. 2.2.a). Now let us consider what happens at regular discretized intervals of time: \( t=0, 1, 2, \) and so on.

![Diagram showing the movement of ants from A to E through B, C, and D, with distances and ant counts at different times.](image)

Fig. 2.2: An example with artificial ants: (a) the initial graph with distances. (b) at time \( t=0 \) there is no trail on the graph edges; therefore, ants choose whether to turn right or left with equal probability. (c) At time \( t=1 \) trail is stronger on shorter edges, which are therefore, in the average, preferred by ants.
Suppose that 30 new ants come to B from A, and 30 to D from E at each time unit, that each ant walks at a speed of 1 per time unit, and that while walking an ant lays down at time t a pheromone trail of intensity 1, which, to make the example simpler, evaporates completely and instantaneously in the middle of the successive time interval \((t+1, t+2)\). At \(t=0\) there is no trail yet, but 30 ants are in B and 30 in D. Their choice about which way to go is completely random. Therefore, on the average 15 ants from each node will go toward F and 15 toward C (Fig. 2.2.b).

At \(t=1\) the 30 new ants that come to B from A find a trail of intensity 15 on the path that leads to F, laid by the 15 ants that went that way from B, and a trail of intensity 30 on the path to C, obtained as the sum of the trail laid by the 15 ants that went that way from B and by the 15 ants that reached B coming from D via C (Fig. 2.2.c). The probability of choosing a path is therefore biased, so that the expected number of ants going toward C will be the double of those going toward F: 20 versus 10 respectively. The same is true for the new 30 ants in D which came from E.

This process continues until all of the ants will eventually choose the shortest path. The idea is that if at a given point an ant has to choose among different paths, those which were heavily chosen by preceding ants (that is, those with a high trail level) are chosen with higher probability. Furthermore high trail levels are synonymous with short paths.

### 2.5.2 The AS Algorithm

The AS algorithm was the first ACO algorithm. Its performance was poor compared to the recent ACO algorithms. Besides, it didn't incorporate local search to improve the solutions. So, the basic AS algorithm was able to find good solutions only for very small sized problems. The AS was first applied to the traveling salesman problem (TSP). TSP can be divided into two classes: symmetric TSP or normally known as TSP and asymmetric TSP, known as ATSP.
TSP:
Let \( V = \{a, \ldots, z\} \) be a set of cities, \( A = \{(r, s) : r, s \in V\} \) be the edge set, and \( \delta(r, s) = \delta(s, r) \) be a cost measure associated with edge \((r, s) \in A\). The TSP is the problem of finding a minimal cost closed tour that visits each city once. In the case cities \( r \in V \) are given by their coordinates \((x_r, y_r)\), and \( \delta(r, s) \) is the Euclidean distance between \( r \) and \( s \), then we have an Euclidean TSP.

ATSP:
If \( \delta(r, s) \neq \delta(s, r) \) for at least one edge \((r, s)\), then the TSP becomes an asymmetric TSP (ATSP).

AS utilizes a graph representation which is the same as that of TSP augmented as follows: in addition to the cost measure \( \delta(r, s) \), each edge \((r, s)\) has also a desirability measure \( \tau(r, s) \), called pheromone, which is updated at run time by artificial ants (ants for short). When AS is applied to symmetric instances of the TSP \( \tau(r, s) = \tau(s, r) \), but when it is applied to asymmetric instances it is possible that \( \tau(r, s) \neq \tau(s, r) \).

Informally, AS works as follows. Each ant generates a complete tour by choosing the cities according to a probabilistic state transition rule: Ants prefer to move to cities which are connected by short edges with a high amount of pheromone. Once all ants have completed their tours a global pheromone updating rule (global updating rule, for short) is applied. A fraction of the pheromone evaporates on all edges (edges that are not refreshed become less desirable), and then each ant deposits an amount of pheromone on edges which belong to its tour in proportion to how short its tour was (in other words, edges which belong to many short tours are the edges which receive the greater amount of pheromone). The process is then iterated.

2.5.3 Tour Construction of AS

In AS, \( m \) artificial ants concurrently build a tour the TSP. Initially, ants are put on randomly chosen cities. At each construction step, ant \( k \) applies a probabilistic action
choice rule, called random proportional rule [2, 41], to decide which city to visit next. In particular, the probability with which ant k, currently at city i, chooses to go to city j is:

\[ p_{ij}^k = \begin{cases} \frac{\tau_{ij}^x \eta_{ij}^y}{\sum_{l\in N_i^k} \tau_{il}^a \eta_{il}^b}, & \text{if } j \in N_i^k \\ 0, & \text{otherwise} \end{cases} \tag{2.1} \]

where \( \eta_{ij} = 1/d_{ij} \) is a heuristic value that is available a priori, \( a \) and \( \beta \) are two parameters which determine the relative influence of the pheromone trail and the heuristic information, and \( N_i^k \) 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 \( N_i^k \) is 0. By this probabilistic rule, the probability of choosing a particular edge \((i, j)\) increases with the value of the associated pheromone trail \( \tau_{ij} \) and of the heuristic information value \( \eta_{ij} \). The role of the parameters \( a \) and \( \beta \) is as follows [2]:

- If \( a=0 \), 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 \( \beta=0 \), 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.

Each ant \( k \) maintains a memory \( M^k \) which contains the cities already visited, in the order they were visited. This memory is used to define the feasible neighborhood \( N_i^k \) in the construction rule given by equation (2.1). In addition, the memory \( M^k \) allows ant \( k \) both to compute the length of the tour \( T^k \) it generated and to retrace the path to deposit pheromone.
There are two different ways of implementing solution construction: parallel and sequential solution construction. In the parallel implementation, at each construction step the entire ants move from their current city to the next one, while in the sequential implementation an ant builds a complete tour before the next one starts to build another one. In the AS, both choices for the implementation of the tour construction are equivalent in the sense that they do not significantly influence the algorithm's behavior.

2.5.4 Update of Pheromone Trails in AS

After all the ants have constructed their tours, the pheromone trails are updated. This is done by first lowering the pheromone value on all edges by a constant factor, and then adding pheromone on the edges the ants have crossed in their tours. Pheromone evaporation is implemented by:

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

where $0 < \rho \leq 1$ is the pheromone evaporation rate. The parameter $\rho$ is used to avoid unlimited accumulation of the pheromone trails and it enables the algorithm to "forget" bad decisions previously taken. In fact, if an edge is not chosen by the ants, its associated pheromone value decreases exponentially in the number of iterations. After evaporation, all ants deposit pheromone on the edges they have crossed in their tour:

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

where $\Delta\tau_{ij}^k$ is the amount of pheromone ant $k$ deposits on the areas it has visited. It is defined as follows:

$$\Delta\tau_{ij}^k = \begin{cases} 1/C_k, & \text{if edge } (i, j) \text{ belongs to } T_k \\ 0, & \text{otherwise} \end{cases}$$
where \( C^k \), the length of the tour \( T^k \) built by the \( k \)-th ant, is computed as the sum of the lengths of the edges belonging to \( T^k \). By means of equation (2.4), the better an ant’s tour is, the more pheromone the edges belonging to this tour receives. In general, edges that are used by many ants and which are part of short tours, receive more pheromone and are therefore more likely to be chosen by ants in future iterations of the algorithm.

2.5.5 The Pseudo Code of AS Algorithm

The pseudo code of the basic AS algorithm is as follows:

```
Initialization
Loop
    Each ant is positioned on a starting node
    Loop
        Apply Tour Construction
        Apply Local Updating
        (Local Search was not applied in AS)
        Until all ants have built a complete solution
    Apply Global Updating
    Until End_condition
```

2.5.6 Problems of AS

Although AS was useful for discovering good or optimal solutions for small TSPs (up to 30 cities), the time required to find such results made it unfeasible for larger problems. Because of the poor performance of AS, it is not used commonly but it is still considered as the basic foundation of the Ant Algorithms. Among the ACO algorithms, two improvements and modifications of the AS are Ant Colony System (ACS) and Max-Min Ant System (MMAS). These two algorithms are the best performing ACO algorithms. And since our proposed thesis is based on ACS, the following sections briefly describe the basics of ACS.
2.6 Ant Colony System

Ant colony system (ACS) is built on the previous AS in the direction of improving efficiency of the algorithm. ACS outperforms other nature-inspired algorithms such as simulated annealing and evolutionary computation.

The main idea is that of having a set of agents, called ants, search in parallel for good solutions and cooperate through pheromone-mediated indirect and global communication. Informally, each ant constructs a solution in an iterative way: it adds new nodes to a partial solution by exploiting both information gained from past experience and a greedy heuristic. Memory takes the form of pheromone deposited by ants on the edges, while heuristic information is simply given by the edge's length.

2.6.1 Differences with AS

ACS differs from the previous AS in three main aspects [42]: (i) the state transition rule provides a direct way to balance between exploration of new edges and exploitation of a priori and accumulated knowledge about the problem, (ii) the global updating rule is applied only to edges which belong to the best ant tour, and (iii) while ants construct a solution a local pheromone updating rule (local updating rule, for short) is applied.
The steps [3] that ACO uses for solving combinatorial optimization problems are:

i. **Tour Construction**: There is several numbers of ants each of which constructs their tours. While constructing tours, ants choose the path based on the deposited pheromone and the problem-dependent parameter. The deposited pheromone is initialized during the initialization phase.

In ACS, when located at node $i$, ant $k$ moves to a node $j$ chosen according to the pseudorandom proportional rule, given by

$$j = \begin{cases} \arg \max_{i \in \mathbb{N}_t} \left\{ r_{ij} \eta_{ij}^{q} \right\}, & \text{if } q \leq q_0; \\ J, & \text{otherwise} \end{cases} \quad (2.5)$$
where $q$ is a random variable uniformly distributed in $[0, 1]$, $q_0 (0 \leq q_0 \leq 1)$ is a parameter, and $J$ is a random variable selected according to the probability distribution given by equation (2.1) (with $\alpha = 1$). In other words, with probability $q_0$ the ant makes the best possible move as indicated by the learned pheromone trails and the heuristic information (in this case, the ant is exploiting the learned knowledge), while with probability $(1 - q_0)$ it performs a biased exploration of the edges. Tuning the parameter $q_0$ allows modulation of the degree of exploration and the choice of whether to concentrate the search of the system around the best-so-far solution or to explore other tours.

ii. **Local Update**: Local update is applied to the edges which are used by the ants while constructing a solution. Local updating decreases some pheromone on the edges in which it is applied. It is done so that the edges will be less desirable for the future ants. Thus, the local updating makes the search for new, possibly better tours in the neighborhoods of the previous best tour.

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

$$
\tau_y \leftarrow (1 - \xi)\tau_y + \xi \tau_0
$$

(2.6)

where $\xi, 0 < \xi < 1$, 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. A good value for $\xi$, is 0.1, while a good value for $\tau_0$ was found to be $1/nC''_{nn}$, where $n$ is the number of nodes in the problem instance and $C''_{nn}$ is the length of a nearest-neighbor tour. The effect of the local updating rule is that each time an ant uses an edge $(i, j)$ its pheromone trail $\tau_y$ is reduced, so that the edge becomes less desirable for the following ants. In other words, this allows an increase in the exploration of edges 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).
iii. **Local Search:** The local search of ACO is used to reduce the chance of getting stuck in local optima and to optimize the solutions found by the local updating rule. Existing local search strategy cannot find optimum solution for larger problems. So, it is necessary to devise a new adaptive local search strategy that will increase the chance of finding optimum solution and reduce the required time.

iv. **Global Update:** A Global Updating Rule is used to deposit pheromone. Since it increases the pheromone of edges, more ants in the future will prefer these edges. So, this rule biases the ants to exploit the visited edges.

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:

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

(2.7)

where $\Delta \tau_{ij}^{bs} = 1/C^{bs}$. It is important to note that in ACS the pheromone trail update, both evaporation and new pheromone deposit, only applies to the edges of $T^{bs}$, not to all the edges as in AS. 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 $\rho$ represents pheromone evaporation; unlike AS's equations (2.3) and (2.4), in equation (2.7) the deposited pheromone is discounted by a factor $\rho$; this results in the new pheromone trail being a weighted average between the old pheromone value and the amount of pheromone deposited.
2.7 Features of ACO Algorithm

Features of the ACO algorithms are [43]:

- Flexible: The colony can respond to internal and external challenges
- Robust: Tasks are completed even if some individuals fail
- Decentralized: There is no central controller in the colony
- Self-organized: Paths to solutions are emergent rather than predefined

It was found that the media used to communicate among individuals information regarding paths and used to decide where to go consists of pheromone trails. A moving ant lays some pheromone (in varying quantities) on the ground, thus marking the path it follows by a trail of this substance. While an isolated ant moves essentially at random, an ant encountering a previously laid trail can detect it and decide with high probability to follow it, thus reinforcing the trail with its own pheromone.

The collective behavior that emerges is a form of autocatalytic behavior where the more the ants are following a trail, the more attractive that trail becomes for being followed. The process is thus characterized by a positive feedback loop, where the probability with which an ant chooses a path increases with the number of ants that previously chose the same path.

An autocatalytic, i.e. positive feedback, process is a process that reinforces itself, in a way that causes very rapid convergence and, if no limitation mechanism exists, leads to explosion. Ants are also given a heuristic to guide the early stages of the computational process, when experience hasn't yet accumulated into the problem structure. This heuristic automatically loses importance as the experience gained by ants, and memorized in the problem representation, increases.
2.8 Local Search

Local search is a general approach for finding high-quality solutions to hard combinatorial optimization problems in reasonable time. It is based on the iterative exploration of solutions trying to improve the current solution by local changes. The types of local changes that may be applied to a solution are defined by a neighborhood structure.

A neighborhood structure is a function \( N : S \mapsto 2^S \) that assigns a set of neighbors \( N(s) \subseteq S \) to every \( s \in S \). \( N(s) \) is also called the neighborhood of \( s \).

Once ants have completed their solution construction, the solutions are taken to their local optimum by the application of a local search routine. Then pheromones are updated on the edges of the locally optimized solutions. Local search methods that were used in ACO algorithms are: 2-opt, 2.5-opt and 3-opt.

2-opt move

The 2-opt algorithm is a special case of the \( \lambda \)-opt algorithm [15], where in each step \( \lambda \) links of the current tour are replaced by \( \lambda \) links in such a way that a shorter tour is achieved. In other words, in each step a shorter tour is obtained by deleting \( \lambda \) links and putting the resulting paths together in a new way, possibly reversing one or more of them.

The 2-opt neighborhood consists of those tours that can be obtained from a tour \( s \) by replacing two of its edges. This local search strategy starts with a given tour and replaces 2 links of the tour with 2 other links in such a way that the new tour length is shorter. Searching is continued in this way until no more improvements are possible. The figure 2.4 below gives an example of one specific 2-exchange: the pair of edges \((b, c)\) and \((a, f)\) is removed and replaced by the pair \((a, c)\) and \((b, f)\).
2.5-opt move

2.5-opt is a local search algorithm that includes a strongly restricted version of a 3-opt move on top of a 2-opt local search. When checking for an improving 2-opt move, it is also checked whether inserting the city between a node $i$ and its successor, as illustrated in fig 2.5, results in an improved tour. 2.5-opt leads only to a small, constant overhead in computation time over that required by a 2-opt local search but, as experimental results show, it leads to significantly better tours. However, the tour quality returned by 2.5-opt is still significantly worse than that of 3-opt.

3-opt move

The 3-opt neighborhood consists of those tours that can be obtained from a tour $s$ by replacing three of its edges. The removal of three edges results in three partial tours that can be recombined into a full tour in eight different ways. The figure 2.6 below gives one particular example of a 3-opt exchange move.
Problems of Local Search

The local search methods that were used had to predefine the number of edges to exchange. As a result, the time complexity grew with the size of the given problem. Besides, fixed number of edge exchanges did not produce good solutions.

2.9 Conclusion

AS is the predecessor of ACO algorithms. After AS, several works have been reported among which ACS is the best-performing one. Since our LSACO uses the ACS as the guideline, this chapter describes both the AS and the ACS algorithms. ACO algorithms have been used to solve several combinatorial optimization problems. ACO algorithms use pheromone model to guide the search and local search strategy to improve the solution quality. But the pheromone model and local search that are used are not efficient for solving large problems. The existing ACO algorithms get stuck in local optima when they are applied in large problem instances and their time complexity is also increased.
3.1 Introduction

ACO has been applied successfully to a large number of difficult combinatorial optimization problems including traveling salesman problems [27], quadratic assignment problems [15], and scheduling problems [6, 8, 9], as well as to dynamic routing problems [44] in telecommunication networks. Starting from the AS, different improvements and modifications of it have been proposed till now. But all the existing ACO algorithms have their shortcomings which have been described in chapter 1. The problems were mainly the stagnation behavior or the local optima problem and the higher time complexity to solve larger problems. The proposed algorithm addresses these problems. The previous chapter deals with the basics of ACO necessary to understand the background of the proposed algorithm. This chapter describes our proposed algorithm. The proposed algorithm differs with the existing algorithms in several aspects. It applies new updating rules, different tour construction techniques and an adaptive local search method. These strategies make our proposed Local Search based ACO (LSACO) algorithm a promising one.

3.2 Overview of LSACO Algorithm

The proposed LSACO algorithm differs with the existing ACO algorithms in many aspects. LSACO will first map the problem into a graph like the other ACO algorithms. The pheromone in the edges of the graph will be then initialized based on the problem. Ants in the proposed algorithm will use a new state transition rule based on the pheromone in an edge and the distance of that edge in constructing tours.

The pheromone level in edges will be changed by using a new local updating rule. Unlike conventional ACO, the proposed algorithm will use a variable edge changing search strategy for finding good quality solution by exploiting and exploring the search space. The search of an edge for entering a tour will be limited to an
appropriate number of nearest neighbors. Pheromone is then updated on the tour that will found after applying the search strategy using a new global updating rule.

Figure 3.1 shows the flowchart of the LSACO Algorithm. The proposed algorithm makes it possible to find optimal solutions to large-scale problems, in reasonable running times. Although this algorithm is approximate but it can produce optimal solutions in high frequency. It can produce optimal solutions for many large problems whereas other approximate algorithms cannot.
3.3 New Tour Construction Phase

The proposed LSACO constructs tours or create solutions in different but effective way to improve the solution quality in a less amount of time. Initially, LSACO places m ants on randomly chosen nodes. In AS, the number of ants was equal to the number of nodes in the given problem. But when the problem size grows large, it becomes infeasible to use large number of ants. Moreover, limiting the number of ants from 10 to 20 does not decrease the quality of solution. So, LSACO uses approximately 15 numbers of ants because it will be used to solve very large problem instances which cannot be solved by the existing ACO algorithms.

Pheromone initialization plays an important role in solution quality [45]. In MMAS, pheromone is initialized to a maximum value so as to increase the exploration rate at the start of the search. In AS and other ACO algorithms, pheromone is initialized to a lower value so as to reduce the exploration rate at the start of the search. But in LSACO, pheromone is initialized to \(1/C^n\) where \(C^n\) is the length of a nearest-neighbor tour of problem size \(n\). This initial pheromone value is not too low or too high. As a result, at the start of the search, all edges will be good candidates for exploration but not for too explorative. Thus, this pheromone value avoids early local optima and also avoids low convergence.

The ants will choose the next state/node from the nodes that are not visited yet. A state transition rule which is a probability distribution rule based on heuristic information about the problem will be used to choose the next node. This rule is used in ant colony system which is given as follows:

\[
j = \begin{cases} \arg \max_{v \in N_i^k} \left\{ \tau \left[ \eta_{mv} \right]^\beta \right\} & \text{if } q \le q_0 \text{ (exploitation)} \\
J_i & \text{otherwise (biased exploration)} \end{cases}
\]

(3.1)

where \(N_i^k\) is the feasible neighborhood of ant \(k\) when being at node \(i\), that is, the set of nodes that ant \(k\) has not visited yet, \(q\) is a random variable uniformly distributed in
\( q \in [0, 1], q_0 (0 \leq q_0 \leq 1) \) is a parameter, and \( J \) is a random variable selected according to the probability distribution of AS given by the following equation:

\[
P^k_j = \begin{cases} 
\frac{[\tau^a_{ij}]^\alpha [\eta^\beta_{ij}]}{\sum_{u \in N_i^k} [\tau^a_{iu}]^\alpha [\eta^\beta_{iu}]} & , \text{if } j \in N_i^k \\
0, & \text{otherwise}
\end{cases}
\]

(H3.2)

Here, \( \eta_j = 1/d_j \) is a heuristic value that is available a priori where \( d_j \) is the cost of the edge \((i, j)\). \( \alpha \) and \( \beta \) are two parameters which determine the relative influence of the pheromone trail and the heuristic information. The probability of choosing a node outside \( N_i^k \) is 0. By this probabilistic rule, the probability of choosing a particular edge \((i, j)\) increases with the value of the associated pheromone trail \( \tau^k_j \) and of the heuristic information value \( \eta_j \).

As we know from chapter 2 that \( \alpha \geq 0 \) is the relative importance of the trail and \( \beta \geq 0 \) is the relative importance of the visibility. If \( \alpha = 0 \), the closest nodes are more likely to be selected. If \( \beta = 0 \), only pheromone is used, without any heuristic bias. It generally leads to rather poor results. When \( \beta = 0 \) is used with \( \alpha > 1 \), it leads to the rapid emergence of a local optima, that is, a situation in which all the ants follow the same path and construct the same tour.

In existing ACO algorithms, \( \alpha = 1 \) was used to increase the chance of convergence. But in LSACO, \( \alpha = 0.1 \) is used to increase the exploration and decrease the exploitation rate. Because LSACO uses adaptive local search that exploits the search space to find optimal solution quickly. So, to discover the promising areas of the search space, it is very necessary to give importance to exploration in the tour construction phase.

LSACO also makes use of the candidate list in this construction phase for problems larger than 1000 nodes. Candidate list is computed by the nearest neighbor heuristic.
It is a list of preferred nodes to be visited. Candidate list is pre-computed and remains the same throughout the algorithm. Without candidate lists, the complexity of LSACO algorithm would be $O(n^2)$ where $n$ is the number of nodes. For large problem instances, it is very infeasible. When LSACO is used with candidate list, its complexity nearly reduces to $O(n)$ which is very effective in solving large problems in reasonable time.

In LSACO, for problem size greater than 1000 nodes, candidate lists keep 50 preferred nodes to be visited for each node. It is generally ordered by increasing distances. LSACO first considers the nodes belonging to the candidate list to move to. Only if none of the nodes in the candidate list can be visited then it considers the rest of the nodes. Thus LSACO’s construction phase is a nice balance between exploration and exploitation of search space and its time complexity is also feasible for finding good quality solutions for big problem instances in less time.

### 3.4 New Local Pheromone Update

Local pheromone updating means to decrease pheromone on the edges of a graph. It is done so as to simulate the evaporation of pheromone in real world. Pheromone evaporation decreases the attraction towards an edge. In AS, pheromone was decreased on all edges by a constant factor. This was done as follows:

$$\tau_{ij} \leftarrow (1 - \rho)\tau_{ij}, \forall (i, j) \in L \quad (3.3)$$

where $0 < \rho \leq 1$ is the pheromone evaporation rate. The more the value of $\rho$, the more the pheromone evaporation rate. Pheromone is evaporated so that the ants will follow new edges to explore for better solution. Since pheromone is decreased in all edges by AS and other ACO algorithms, these algorithms cannot be applied in large problems because of $O(n^2)$ complexity. In ACS, the pheromone is decreased by all ants immediately after having crossed an edge $(i, j)$ during the tour construction.
In LSACO, pheromone is not decreased in all edges. Although like ACS, LSACO allows only the ants to decrease pheromone, but LSACO uses different pheromone evaporation rule. In ACS, a small amount of pheromone was also deposited while evaporation during local updating. But LSACO only evaporates a small amount of pheromone. This is done to reduce the desirability of the visited edges by the ants in each iteration so that all the ants do not follow the same path. Since pheromone of all the edges are not updated, LSACO is very effective for solving large problems. The local pheromone updating rule of LSACO is as follows:

\[ \tau_y \leftarrow (1 - \xi)\tau_y \]  

(3.4)

where \( \xi \), \( 0 < \xi < 1 \) is the pheromone decay parameter. This local updating rule reduces the pheromone of the visited edges in each iteration so that the edges become less desirable for the next iteration. This will allow the exploration of unvisited edges. As a result, LSACO’s chance of getting stuck in local optima is greatly reduced and the time required is linear with problem size, i.e., \( O(n) \) where \( n \) is the number of nodes of the given problem.

3.5 Adaptive Local Search

3.5.1 Existing Local Search Methods

As we know from the previous chapter, local search methods are used to improve the tours found by the ants. When ants have completed their solution construction, local search is applied to take the solutions to their local optimum. Then pheromone is updated on the edges of the locally optimized solutions. Local search methods that are generally used in ACO algorithms are 2-opt and 3-opt.

2-opt and 3-opt local search methods can be stated as \( \delta \)-opt for simplicity. In \( \delta \)-opt local search, \( \delta \) edges of the current tour are replaced by \( \delta \) edges in such a way that a shorter tour is achieved. In other words, in each step a shorter tour is tried to obtain by deleting \( \delta \) edges and putting the resulting paths together in a new way.
Disadvantages of Existing Local Search Methods:

These existing local search algorithms have two disadvantages such as:

- Although larger the value of \( \delta \), better the possibility of finding an optimum solution. But when \( \delta \) increases, the number of operations to check the tour also increases. As a result, the algorithm will suffer from time complexity for larger problems. In a naive implementation, the operations of a \( \delta \)-exchange have a time complexity of \( O(n^\delta) \). Although any value for \( \delta \) can be used, but to keep the time complexity low, only \( \delta=2 \) and \( \delta=3 \) are used.

- The existing local search algorithms are not adaptive. That is, they use fixed value of \( \delta \) exchange. Since \( \delta \) is fixed, the solution obtained by local search may not be optimal. Because, it is not possible to know in advance what value of \( \delta \) to use to achieve the best compromise between running time and quality of solution.

3.5.2 Proposed Adaptive Local Search

The proposed algorithm uses an adaptive local search with new control parameters that not only improves the solution quality but also reduces the required time. The proposed LSACO changes the value of \( \delta \) during the runtime of the algorithm deciding in each step what the value of \( \delta \) should be. At each iteration, the increasing values of \( \delta \) are examined whether an exchange of \( \delta \) edges may result in a better solution. That is, when \( \delta \) exchanges yield better tour, then \( \delta+1 \) exchange is considered.

Given a feasible tour, the local search method repeatedly performs exchanges that reduce the length of the current tour, until a tour is reached for which no exchange yields an improvement. This process is applied after the tour construction and local pheromone updating phase occurs. Fig 3.2 shows the flowchart of the adaptive local search method.

The local search starts with \( \delta=2 \). Then it is checked and it is also checked for \( \delta>2 \).
These exchanges are chosen in such a way that a feasible tour may be formed at any stage of the process. If the exploration succeeds in finding a new shorter tour, then the actual tour is replaced with the new tour.

In LSACO, the ants first construct tours. Then the tour with the minimum cost is chosen. Suppose, $T_{min}$ is the tour with the minimum cost. Then at each iteration step the searching attempts to find two sets of links, $X = \{x_1, x_2, \ldots, x_n\}$ and $Y = \{y_1, y_2, \ldots, y_n\}$, such that, if the links of X are deleted from $T_{min}$ and replaced by the links of Y, the result is a better tour. Figure 3.3 illustrates a sample edge exchanging process.
The search for better tours is stopped when the current tour is similar to a previous tour or when all the alternatives are searched.

Modifications to Improve Performance:

To reduce the number of comparisons and thus to reduce time complexity and to improve the performance of the algorithm, several measures have been taken in LSACO. These new improvements make the LSACO a promising algorithm. The major new steps that are incorporated in this new algorithm are given below:

- To speed up the computation time, the concept of don't look bit [46] is used here. Suppose, $x_t = (t_1, t_2)$. Starting from $t_t$, there may be several choices for $x_t$. If for a given choice of $t_t$, the algorithm previously failed to find an improvement, and if $t_t$'s tour neighbors have not changed since that time, then it is unlikely that an improving move can be made if the algorithm again looks at $t_t$. To incorporate this technique, each node has a don't look bit, which initially is 0. The bit for node $t_t$ is set to 1 whenever a search for an improving move with $t_t$ fails, and it is set to 0 whenever an improving move is made in which it is an end node of one of the its edges. In considering candidates for $t_t$ all nodes whose don't look bit is 1 are ignored. This is done in maintaining a

Fig 3.3: Edge exchanging process
queue of nodes whose bits are zero. Thus running time is significantly reduced.

- The existing ACO algorithms use a nearest neighbor heuristic to compute the candidate edges to work with. The heuristic rule is built on the assumption that the shorter an edge is, the greater is the probability that it belongs to an optimal tour. This heuristic rule thus directs the search against short tours and reduces the search effort substantially.

Our proposed LSACO algorithm uses a dynamic ordering of the candidates to speed up the search even more. Each time a shorter tour is found, all edges shared by this new tour and the previous shortest tour become the first candidate edges among the list of candidate edges.

- Backtracking was required in the existing local searches of ACO algorithms. In the proposed LSACO, since adaptive local search is used, backtracking is not required. As a result, the algorithm is simplified and the runtime is reduced a lot.

- The new edge must have a common endpoint with the old edge. That is, $x_i$ and $y_i$ must have a common endpoint and so must $y_i$ and $x_{i+1}$. This condition helps to maintain the chain of links. Fig 3.4 illustrates this condition.

![Fig 3.4: Sharing common endpoints](image-url)
The new tour that will be obtained by replacing edges must also be a closed tour.

Every resulting final tour after exchanging edges must be better than the original one. Otherwise, the original tour is not replaced.

Sets $X$ and $Y$ must be disjoint so as to reduce running time. Figure 3.5 shows that sets $X = \{x_1, x_2, x_3\}$ and $Y = \{y_1, y_2, y_3\}$ are disjoint.

If there is more than one edge to select as a new edge, then an untried edge is chosen. In figure 3.6, $x_1$ and $x_2$ are part of the original tour. Edges $a$, $b$ and $c$ are candidate edges to replace edge $x_2$. If $c$ was already tried in previous iterations, then $a$ or $b$ will be checked. If there are more than one alternative for an edge, the one where the cost is minimum, is chosen. This reduces the running time.

In 2-opt or 3-opt strategy, the algorithm continues its steps by adding fixed number of potential exchanges in order to find an even shorter tour. When no more exchanges are possible, the search stops and the current tour $T$ is replaced by the most advantageous tour. This results in more running time.
But in LSACO, a tour $T$ is replaced by a shorter tour as soon as an improvement is found. As a result, time complexity is reduced.

- If an improvement of the tour is found, all the other edges are marked as untried alternatives and is tried for $\delta + 1$ edge.

- Replaced edges must not break the closed tour.

- To reduce search time, replaced edges will not be added or added edges will not be deleted. Figure 3.7 shows such an example.

![Fig 3.7: Replaced edges will not be added or added edges will not be deleted.](image)

- New edges will be chosen such that cost is reduced in the new tour.

![Fig 3.8: Cost(T2)<Cost(T1)](image)

- The search for a link to enter the tour is limited to maximum 5 nearest neighbors and the search starts from $\delta=2$ edges. If $\delta>5$ is considered, the time complexity increases very much for larger instances but the solution quality remains the same.

- The search for improvements is stopped if the current tour is the same as a previous solution tour. It saves a lot of running time and it does not affect
quality of solutions. If a tour is the same as a previous solution tour, there is no point in attempting to improve it further. The time needed to check that no more improvements are possible is therefore saved.

- The proposed LSACO uses maximum 5-opt move. As a result, it broadens the search and increases the algorithm’s ability to find good tours. Although 5-opt move requires more computation, but due to the adaptive measures and the use of above mentioned techniques, run time is significantly reduced.

3.6 New Global Updating Rule

Global updating is used to deposit pheromone. Deposited pheromone will attract more ants to exploit the visited edges. So, exploitation of the search will be used. LSACO updates the pheromone globally in a new way. In LSACO, exploration is used both in the local pheromone updating and in local search phase. In global updating phase, LSACO exploits the search space in an effective way than the existing ACO algorithms.

In AS, all the ants deposit pheromone on the edges they cross in their tour. The deposited pheromone was dependent of the quality of the tour of an ant. The better the tour, the more pheromone was deposited by an ant. As a result, the edges that are shorter and chosen by many ants receive more pheromone and therefore they are more likely to be chosen by the ants in the future iterations of the algorithm and thus it becomes prone to be stuck in local optima. In AS, the global updating rule was implemented as follows:

\[
\tau_{ij}^k \leftarrow \tau_{ij} + \Delta \tau_{ij}^k, \quad \forall (i,j) \in L, \quad \forall k \leq M
\]  

(3.5)

\[
\Delta \tau_{ij}^k = \begin{cases} 
\frac{1}{C^k}, & \text{if edge } (i,j) \text{ belongs to } T^k \\
0, & \text{otherwise}
\end{cases}
\]  

(3.6)
where $\Delta r^k_{ij}$ is the amount of pheromone ant $k$ deposits on the areas it has visited and $C^k$, the length of the tour $T^k$ built by the $k$-th ant, is computed as the sum of the lengths of the edges belonging to $T^k$.

In ACS, pheromone deposition is always performed in the best-so-far tour. But when pheromone updates are always performed by the best-so-far ant, the search focuses very quickly around best-so-far tour. This results in local optima. Other ACO algorithms also suffer from local optima problem.

To reduce computational complexity, LSACO applies global updating rule only by a single ant instead of all the ants as in AS. This single ant may be the iteration-best ant or the best-so-far ant. As we know that if the best-so-far ant is allowed to deposit pheromone each time, the search falls in local optima. And if the iteration-best ant is allowed to deposit pheromone each time, then the number of edges that receive pheromone is larger and the search is less directed. So, LSACO applies both the best-so-far and iteration-best update rules alternatively. The global updating rule of LSACO is given below:

$$
\tau_{ij} \leftarrow \tau_{ij} + \Delta r'_{ij},
$$

(3.7)

where $\Delta r'_{ij} = 1/C^{bs}$ or $\Delta r'_{ij} = 1/C^{ib}$. $C^{bs}$ is the length of the best-so-far tour and $C^{ib}$ is the length of the iteration-best tour. Since only single ant is allowed to deposit pheromone in each iteration, the computational complexity is only $O(n)$. And also the LSACO is less prone to local optima.

### 3.7 Time Complexity of LSACO

Each tour construction has complexity of $O(n^2)$, which would lead to substantial run times for larger instances. So, like ACO algorithms, LSACO makes use of a nearest neighbor heuristic to decrease the initial search space. This is done by comprising a fixed number of nearest neighbors for each node in order of increasing distances. This list of nodes is called candidate list. When constructing a tour an ant chooses the next
node among those of the candidate list, if possible. Only if all the members of the candidate list of a node have already been visited, one of the remaining nodes is chosen. So, the time complexity reduces to $O(n)$.

In ACO algorithms, pheromone trails are stored in a matrix with $O(n^2)$ entries. All the entries of this matrix are updated at each iteration. This is a very expensive operation for large problems. So, LSACO updates pheromone only to the edges connecting a node $i$ to nodes belonging to $i$'s candidate list. Hence, the pheromone trails can be updated in $O(n)$.

Time complexity of 2-opt and 3-opt local search methods are approximately $O(n^{2.2})$. Since our adaptive local search reduces the number of comparisons and uses different approaches to reduce running time, its time complexity reduces to $O(n)$.

So, the average time complexity of our proposed LSACO algorithm is $O(n)$.

### 3.8 Differences with Existing Works

Our proposed LSACO algorithm differs from the existing ACO algorithms in many aspects. First, pheromone is initialized in different way. This new pheromone initialization avoids early local optima and also avoids low convergence. Second, it applies new state transition rule to construct tours.

Third, both the local and global pheromone updating strategy is different from the other existing ACO algorithms. In local pheromone updating, pheromone is not decreased in all edges. LSACO allows only the ants to decrease pheromone instead of updating in all edges. So, computational complexity of LSACO is lower than the AS.

Fourth, it applies adaptive local search. The existing local search algorithms are not adaptive. That is, they use fixed value of edge exchange. Since the number of edges is fixed, the solution obtained by local search may not be optimal because it is not possible to know in advance what number of edge exchange to use to achieve the best compromise between running time and quality of solution.
Fifth, to speed up the computation time and to find good quality solutions, several other techniques have been applied in LSACO. For example, in adaptive local search, the concept of don’t look bit is used to avoid fruitless search. Sixth, our proposed LSACO algorithm uses a dynamic ordering of the candidates to speed up the search even more. It means that each time a shorter tour is found, all edges shared by this new tour and the previous shortest tour become the first two candidate edges among the list of candidate edges.

Seventh, our proposed LSACO algorithm uses maximum 5-opt move. So, LSACO widens the search and increases the probability of finding optimum solutions. Finally, LSACO also updates the pheromone globally in a new way. LSACO applies global updating rule only by a single ant instead of all the ants as in AS. This single ant may be the iteration-best ant or the best-so-far ant.

Due to these differences with the existing ACO works, LSACO finds good solutions in a reasonable amount of time. Its computational complexity is also significantly low than the other ACO algorithms. As a result, LSACO can be successfully applied in larger problem instances which were not possible with the existing ACO algorithms.

3.9 Conclusion

The existing ACO algorithms get stuck in local optima when applied to large problems. The time complexity of the existing ACO algorithms is also high. The proposed LSACO algorithm addresses the problems of existing ACO algorithms. LSACO adopts several changes and modifications to improve the performance in solving large combinatorial optimization problems. The proposed LSACO differs in many aspects with the existing ACO algorithms.
Chapter 4

Experimental Study

4.1 Introduction

This chapter evaluates the performance of LSACO on several well-known benchmark problems such as symmetric traveling salesman problems [22, 49], asymmetric traveling salesman problems [22, 49] and Hamiltonian cycle problems [47]. These problems have been used to experiment the nature and performance of the existing ACO algorithms. This chapter also describes the format of the datasets that were taken and the source of the datasets. Experiments were done to evaluate the performance of LSACO and to compare it with the existing ACO algorithms.

4.2 Description of Data Set

The choice of the test problems was dictated by published results found in the literature of the existing ACO algorithms. The test problems can be found in TSPLIB. TSPLIB is a library of sample instances for the TSP, asymmetric TSP, Hamiltonian problems and related problems from various sources and of various types. The TSPLIB problems can be obtained from [47] and [48].

4.2.1 Symmetric TSP

A salesman is required to visit each of n given cities once and only once, starting from any city and returning to the original place of departure and the distance of the travel should be minimum. The distances between any pair of cities are assumed to be known by the salesman. Distance can be replaced by another notion, such as time or money. In the following the term 'cost' is used to represent any such notion. This problem, the traveling salesman problem (TSP), is one of the most widely studied problems in combinatorial optimization. The problem is easy to state, but hard to solve. Mathematically, the problem may be stated as follows:
Given a cost matrix \( C = (c_{ij}) \), where \( c_{ij} \) represents the cost of going from city \( i \) to city \( j \), \((i, j = 1, \ldots, n)\), find a permutation \((i_1, i_2, i_3, \ldots, i_n)\) of the integers from 1 through \( n \) that minimizes the quantity \( c_{i_1i_2} + c_{i_2i_3} + c_{i_3i_4} + \cdots + c_{i_{n-1}i_1} \).

The types of TSP problems that were considered here are drilling problem, bridge tournament problem, clustered random problem, programmed logic array and city problems. Many of these symmetric problems are used to evaluate the performance of LSACO.

4.2.2 Asymmetric TSP

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. In this case, the distance from node \( i \) to node \( j \) and the distance from node \( j \) to node \( i \) may be different. The types of ATSP problems that were considered here are stacker crane application and city problems.

4.2.3 Hamiltonian Cycle Problem

Let \( G \) be a finite graph with \( V(G) \) the set of vertices and \( E(G) \) the set of edges. A Hamiltonian cycle \( c \) of \( G \) is a cycle that goes through every vertex exactly once. The Hamiltonian cycle problem (HCP) asks whether a given graph \( G \) has a Hamiltonian cycle. The Hamiltonian path problem for graph \( G \) is equivalent to the Hamiltonian cycle problem in a graph \( H \) obtained from \( G \) by adding a new vertex and connecting it to all vertices of \( G \). At present, the TSPLIB includes 9 Hamiltonian cycle problems ranging in size from 1000 to 5000 nodes. Every instance of the problems contains a Hamiltonian cycle.
4.3 Types of Data Formats Used

Each data file consists of a specification part and of a data part. The specification part contains information on the file format and on its contents. The data part contains explicit data.

4.3.1 The Specification Part

All entries in this section are of the form `<keyword>:<value>`, where `<keyword>` denotes an alphanumerical keyword and `<value>` denotes alphanumerical or numerical data, respectively. The following keywords are used:

i. **NAME: <string>**  
   Identifies the data file.

ii. **TYPE: <string>**  
    Specifies the type of the data. Possible types are
    a. TSP: Data for a Symmetric Traveling Salesman Problem
    b. ATSP: Data for an Asymmetric Traveling Salesman Problem
    c. HCP: Hamiltonian Cycle Problem data
    d. TOUR: A collection of tours

iii. **COMMENT: <string>**  
    Additional comments are given here.

iv. **DIMENSION: <integer>**  
   For a TSP or ATSP, the dimension is the number of its nodes. For a TOUR file it is the dimension of the corresponding problem.

v. **EDGE_WEIGHT_TYPE: <string>**  
   Specifies how the edge weights (or distances) are given. The values are:
   a. EXPLICIT: Weights are listed explicitly in the corresponding problem
b. EUC_2D: Weights are Euclidean distances in 2-D

c. CEIL_2D: Weights are Euclidean distances in 2-D rounded up

d. GEO: Weights are geographical distances

e. ATT: Special distance function for problems att48 and att532

vi. EDGE_WEIGHT_FORMAT: <string>
Describes the format of the edge weights if they are given explicitly. The values are:

a. FUNCTION: Weights are given by a function

b. FULL_MATRIX: Weights are given by a full matrix

c. UPPER_ROW: Upper triangular matrix (row-wise without diagonal entries)

d. LOWER_ROW: Lower triangular matrix (row-wise without diagonal entries)

e. UPPER_DIAG_ROW: Upper triangular matrix (row-wise including diagonal entries)

f. LOWER_DIAG_ROW: Lower triangular matrix (row-wise including diagonal entries)

vii. EDGE_DATA_FORMAT: <string>
Describes the format in which the edges of a graph are given, if the graph is not complete. The values are:

a. EDGE_LIST: The graph is given by an edge list

b. ADJ_LIST: The graph is given as an adjacency list

viii. NODECOORDTYPE: <string>
Specifies whether coordinates are associated with each node. The values are:

a. TWOD_COORDS: Nodes are specified by coordinates in 2-D

b. NO_COORDS: The nodes do not have associated coordinates

The default value is NO_COORDS.
ix. DISPLAY_DATA_TYPE: <string>
Specifies how a graphical display of the nodes can be obtained. The values are:

a. COORD_DISPLAY: Display is generated from the node coordinates
b. TWOD_DISPLAY: Explicit coordinates in 2-D are given
c. NO_DISPLAY: No graphical display is possible

The default values is COORD_DISPLAY if node coordinates are specified and NO_DISPLAY otherwise.

x. EOF:
Terminates the input data.

4.3.2 The Data Part

Depending on the choice of specifications some additional data may be required. These data are given in corresponding data sections following the specification part. Each data section begins with the corresponding keyword. The length of the section is either implicitly known from the format specification, or the section is terminated by an appropriate end-of-section identifier.

i. NODE_COORD_SECTION:
Node coordinates are given in this section. Each line is of the form:

<integer> <real> <real>

if NODE_COORD_TYPE is TWOD_COORDS.
The integers give the number of the respective nodes. The real numbers give the associated coordinates.

ii. EDGE_DATA_SECTION:
Edges of a graph are specified in either of the two formats allowed in the EDGE_DATA_FORMAT entry. If the type is EDGE_LIST, then the edges are given as a sequence of lines of the form:

<integer> <integer>
each entry giving the terminal nodes of some edge. The list is terminated by a -1. If the type is ADJ_LIST, the section consists of a list of adjacency lists for nodes. The adjacency list of a node $x$ is specified as

$$<\text{integer}> <\text{integer}> ... <\text{integer}> -1$$

where the first integer gives the number of node $x$ and the following integers (terminated by -1) the numbers of nodes adjacent to $x$. The list of adjacency lists is terminated by an additional -1.

iii. DISPLAY_DATA_SECTION:
If DISPLAY_DATA_TYPE is TWOD_DISPLAY, the 2-dimensional coordinates from which a display can be generated are given in the form (per line)

$$<\text{integer}> <\text{real}> <\text{real}>$$

The integers specify the respective nodes and the real numbers give the associated coordinates.

iv. TOUR_SECTION:
A collection of tours is specified in this section. Each tour is given by a list of integers giving the sequence in which the nodes are visited in this tour. Every such tour is terminated by a -1. An additional -1 terminates this section.

v. EDGE_WEIGHT_SECTION:
The edge weights are given in the format specified by the EDGE_WEIGHT_FORMAT entry. At present, all explicit data is integral and is given in one of the (self-explanatory) matrix formats with implicitly known lengths.
4.4 Experimental Setup

The datasets may be in different forms like Euclidean distance, geographical distance or other forms. The distances between the nodes are processed to take it into a 2-dimensional matrix form. For each dataset, 15 trials were taken to compute the result. The experiments have been computed on a Pentium 4 machine of 1.8 GHz processor speed. The parameters considered here are:

- $\alpha$: relative importance of pheromone trail, $0 < \alpha < 1$
- $\beta$: relative importance of the distance or heuristic information, $\beta > 0$
- $\eta$: heuristic value
- $m$: the number of ants
- $q$: a random number uniformly distributed in $[0 .. 1]$  
- $q_0$: a parameter $[0 \leq q_0 \leq 1]$
- $\tau_0$: initial pheromone
- $\xi$: pheromone decay parameter, $0 < \xi < 1$

The heuristic value $\eta_{ij}$ is set to $\eta_{ij} = 1/d_{ij}$, where $d_{ij}$ is the distance between the nodes $i$ and $j$. The initial pheromone value set to $\tau_0 = 1/C^m$, where $C^m$ is the tour length produced by the nearest neighbor heuristic and $n$ is the number of nodes. During the tour construction the ants use a candidate list of size 20. And the adaptive local search uses a candidate list of size 50. All the parameter values presented here were found to be optimum for yielding better solutions quickly.

The results are compared with the optimum result that is given in the TSPLIB [47, 48]. The best result is computed for each data set and the percentage error is computed as $\left(\frac{\text{best} - \text{optimum}}{\text{optimum}} \times 100\right)$. Here, optimum is the length of the shortest possible tour found in [47]. Best means the best length distance found over 15 trials by our proposed LSACO algorithm.
4.5 Experimental Results and Analysis

Tables 4.1-4.9 show the results of LSACO over 15 independent trials on several different problems. The time is given in seconds. The averaged results are computed over 15 trials. It can be observed from tables 4.1-4.9 that our proposed LSACO produces very good results in a very short time. For example, for pla7397 dataset of symmetric traveling salesman problem, the best length produced by the LSACO algorithm is found to be optimal.

In all the experiments of the following sections, the numeric parameters are set to \( \alpha = 0.1, \beta = 2, \theta = 0.9 \) and \( \zeta = 0.1 \). The number of ants is set to 10 to 20 depending on the problems. For small problems, \( m \) is set to 20 and for large problems \( m \) is set to 10. For problem dimension less than 1000, candidate list is not used. For problem dimension more than 1000, candidate list 50 is used during adaptive local search. These values help getting near optimum results in a reasonable time.

Table 4.1: Performance of LSACO on some problems of Symmetric TSP over 15 trials (Edge weight is in Geographical distance)

<table>
<thead>
<tr>
<th>Name</th>
<th>Dimension</th>
<th>Type</th>
<th>Optimum</th>
<th>Best Length</th>
<th>% Error</th>
<th>Best Time (Seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>bayg29</td>
<td>29</td>
<td>GEO</td>
<td>1610</td>
<td>1610</td>
<td>0.0</td>
<td>0</td>
</tr>
<tr>
<td>Bays29</td>
<td>29</td>
<td>GEO</td>
<td>2020</td>
<td>2020</td>
<td>0.0</td>
<td>0</td>
</tr>
<tr>
<td>gr96</td>
<td>96</td>
<td>GEO</td>
<td>55209</td>
<td>55209</td>
<td>0.0</td>
<td>0</td>
</tr>
<tr>
<td>gr137</td>
<td>137</td>
<td>GEO</td>
<td>69853</td>
<td>69853</td>
<td>0.0</td>
<td>0</td>
</tr>
<tr>
<td>gr202</td>
<td>202</td>
<td>GEO</td>
<td>40160</td>
<td>40160</td>
<td>0.0</td>
<td>0</td>
</tr>
<tr>
<td>gr229</td>
<td>229</td>
<td>GEO</td>
<td>134602</td>
<td>134602</td>
<td>0.0</td>
<td>1</td>
</tr>
<tr>
<td>ali535</td>
<td>535</td>
<td>GEO</td>
<td>202309</td>
<td>202309</td>
<td>0.0</td>
<td>1</td>
</tr>
<tr>
<td>gr666</td>
<td>666</td>
<td>GEO</td>
<td>294358</td>
<td>294358</td>
<td>0.0</td>
<td>4</td>
</tr>
</tbody>
</table>

Table 4.1 shows the performance of LSACO on some problems of symmetric TSP. The results were taken over 15 trials. The edge weights of these problems are given in geometric distances. In geographical problems [47], the nodes correspond to points on the earth and the distance between two points is their distance on the idealized sphere with radius 6378.388 kilometers. The node coordinates give the geographical latitude
and longitude of the corresponding point on the earth. Latitude and longitude are
given in the form DDD.MM where DDD are the degrees and MM the minutes.
Positive latitude is assumed to be “North”, negative latitude means “South”. Positive
longitude means “East”, negative longitude is assumed to be “West”. First the input is
converted to geographical latitude and longitude given in radians. The distance
between two different nodes in kilometers is then computed from these values.

Table 4.2 shows the performance of LSACO on some symmetric TSP problems of
edge weights given in matrix form. LSACO produces optimum results for all these
problems. Time required to find the best solution is also very low. Table 4.3 shows
the performance of LSACO on some symmetric TSP problems of edge weights given
in special form. The datasets that are used here are att48 and att532.

Table 4.2: Performance of LSACO on some problems of Symmetric TSP over 15
trials (Edge weight is in matrix form)

<table>
<thead>
<tr>
<th>Name</th>
<th>Dimension</th>
<th>Type</th>
<th>Optimum</th>
<th>Best Length</th>
<th>% Error</th>
<th>Best Time (Seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Swiss42</td>
<td>42</td>
<td>MATRIX</td>
<td>1273</td>
<td>1273</td>
<td>0.0</td>
<td>0</td>
</tr>
<tr>
<td>gr48</td>
<td>48</td>
<td>MATRIX</td>
<td>5046</td>
<td>5046</td>
<td>0.0</td>
<td>0</td>
</tr>
<tr>
<td>brazil58</td>
<td>58</td>
<td>MATRIX</td>
<td>25395</td>
<td>25395</td>
<td>0.0</td>
<td>0</td>
</tr>
<tr>
<td>gr120</td>
<td>120</td>
<td>MATRIX</td>
<td>6942</td>
<td>6942</td>
<td>0.0</td>
<td>0</td>
</tr>
<tr>
<td>si175</td>
<td>175</td>
<td>MATRIX</td>
<td>21407</td>
<td>21407</td>
<td>0.0</td>
<td>0</td>
</tr>
<tr>
<td>brg180</td>
<td>180</td>
<td>MATRIX</td>
<td>1950</td>
<td>1950</td>
<td>0.0</td>
<td>0</td>
</tr>
<tr>
<td>si535</td>
<td>535</td>
<td>MATRIX</td>
<td>48450</td>
<td>48450</td>
<td>0.0</td>
<td>1</td>
</tr>
<tr>
<td>pa561</td>
<td>561</td>
<td>MATRIX</td>
<td>2763</td>
<td>2763</td>
<td>0.0</td>
<td>1</td>
</tr>
<tr>
<td>si1032</td>
<td>1032</td>
<td>MATRIX</td>
<td>92650</td>
<td>92650</td>
<td>0.0</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 4.3: Performance of LSACO on some problems of Symmetric TSP over 15
trials (Edge weight is in special form)

<table>
<thead>
<tr>
<th>Name</th>
<th>Dimension</th>
<th>Type</th>
<th>Optimum</th>
<th>Best Length</th>
<th>% Error</th>
<th>Best Time (Seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>att48</td>
<td>48</td>
<td>ATT</td>
<td>10628</td>
<td>10628</td>
<td>0.0</td>
<td>0</td>
</tr>
<tr>
<td>att532</td>
<td>532</td>
<td>ATT</td>
<td>27686</td>
<td>27686</td>
<td>0.0</td>
<td>1</td>
</tr>
</tbody>
</table>
Table 4.4 shows some symmetric TSP problems whose edge weights are given in Euclidean 2-dimensional format. The largest problem of this type that is used here is fnl4461. Its dimension or the number of nodes is 4461. LSACO found the optimal solution for this dataset in 41 seconds.

**Table 4.4: Performance of LSACO on some problems of Symmetric TSP over 15 trials (Edge weight is in Euclidean distances in 2-D)**

<table>
<thead>
<tr>
<th>Name</th>
<th>Dimension</th>
<th>Type</th>
<th>Optimum</th>
<th>Best Length</th>
<th>% Error</th>
<th>Best Time (Seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>eil51</td>
<td>51</td>
<td>EUC_2D</td>
<td>426</td>
<td>426</td>
<td>0.0</td>
<td>0</td>
</tr>
<tr>
<td>eil76</td>
<td>76</td>
<td>EUC_2D</td>
<td>538</td>
<td>538</td>
<td>0.0</td>
<td>0</td>
</tr>
<tr>
<td>kroA100</td>
<td>100</td>
<td>EUC_2D</td>
<td>21282</td>
<td>21282</td>
<td>0.0</td>
<td>0</td>
</tr>
<tr>
<td>rat195</td>
<td>195</td>
<td>EUC_2D</td>
<td>2323</td>
<td>2323</td>
<td>0.0</td>
<td>0</td>
</tr>
<tr>
<td>d198</td>
<td>198</td>
<td>EUC_2D</td>
<td>15780</td>
<td>15780</td>
<td>0.0</td>
<td>0</td>
</tr>
<tr>
<td>a280</td>
<td>280</td>
<td>EUC_2D</td>
<td>2579</td>
<td>2579</td>
<td>0.0</td>
<td>0</td>
</tr>
<tr>
<td>lin318</td>
<td>318</td>
<td>EUC_2D</td>
<td>42029</td>
<td>42029</td>
<td>0.0</td>
<td>0</td>
</tr>
<tr>
<td>Linhp318</td>
<td>318</td>
<td>EUC_2D</td>
<td>41345</td>
<td>41345</td>
<td>0.0</td>
<td>0</td>
</tr>
<tr>
<td>fl417</td>
<td>417</td>
<td>EUC_2D</td>
<td>11861</td>
<td>11861</td>
<td>0.0</td>
<td>0</td>
</tr>
<tr>
<td>pr439</td>
<td>439</td>
<td>EUC_2D</td>
<td>107217</td>
<td>107217</td>
<td>0.0</td>
<td>1</td>
</tr>
<tr>
<td>pcb442</td>
<td>442</td>
<td>EUC_2D</td>
<td>50778</td>
<td>50778</td>
<td>0.0</td>
<td>0</td>
</tr>
<tr>
<td>rat575</td>
<td>575</td>
<td>EUC_2D</td>
<td>6773</td>
<td>6773</td>
<td>0.0</td>
<td>1</td>
</tr>
<tr>
<td>rat783</td>
<td>783</td>
<td>EUC_2D</td>
<td>8806</td>
<td>8806</td>
<td>0.0</td>
<td>0</td>
</tr>
<tr>
<td>pcb1173</td>
<td>1173</td>
<td>EUC_2D</td>
<td>56892</td>
<td>56892</td>
<td>0.0</td>
<td>1</td>
</tr>
<tr>
<td>u1432</td>
<td>1432</td>
<td>EUC_2D</td>
<td>152970</td>
<td>152970</td>
<td>0.0</td>
<td>4</td>
</tr>
<tr>
<td>d1655</td>
<td>1655</td>
<td>EUC_2D</td>
<td>62128</td>
<td>62128</td>
<td>0.0</td>
<td>5</td>
</tr>
<tr>
<td>u1817</td>
<td>1817</td>
<td>EUC_2D</td>
<td>57201</td>
<td>57201</td>
<td>0.0</td>
<td>6</td>
</tr>
<tr>
<td>rl1889</td>
<td>1889</td>
<td>EUC_2D</td>
<td>316536</td>
<td>316536</td>
<td>0.0</td>
<td>3</td>
</tr>
<tr>
<td>pcb3038</td>
<td>3038</td>
<td>EUC_2D</td>
<td>137694</td>
<td>137694</td>
<td>0.0</td>
<td>19</td>
</tr>
<tr>
<td>fnl4461</td>
<td>4461</td>
<td>EUC_2D</td>
<td>182566</td>
<td>182566</td>
<td>0.0</td>
<td>41</td>
</tr>
</tbody>
</table>

The edge weight type CEIL_2D requires that the 2-dimensional Euclidean distances are rounded up to the next integer. Table 4.5 shows this type of input data types.

**Table 4.5: Performance of LSACO on some problems of Symmetric TSP**

(Edge weight is in Euclidean distances in 2-D)

<table>
<thead>
<tr>
<th>Name</th>
<th>Dimension</th>
<th>Type</th>
<th>Optimum</th>
<th>Best Length</th>
<th>% Error</th>
<th>Best Time (Seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>dsj1000</td>
<td>1000</td>
<td>CEIL_2D</td>
<td>18659688</td>
<td>18659688</td>
<td>0.0</td>
<td>9</td>
</tr>
<tr>
<td>pla7397</td>
<td>7397</td>
<td>CEIL_2D</td>
<td>23260728</td>
<td>23260728</td>
<td>0.0</td>
<td>185</td>
</tr>
</tbody>
</table>
Table 4.6: Performance of LSACO on some problems of Asymmetric TSP over 15 trials (Edge weight is in matrix form)

<table>
<thead>
<tr>
<th>Name</th>
<th>Dimension</th>
<th>Type</th>
<th>Optimum</th>
<th>Best Length</th>
<th>%Error</th>
<th>Best Time (Seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td>P43</td>
<td>43</td>
<td>MATRIX</td>
<td>5620</td>
<td>5620</td>
<td>0.0</td>
<td>0</td>
</tr>
<tr>
<td>Ry48p</td>
<td>48</td>
<td>MATRIX</td>
<td>14422</td>
<td>14422</td>
<td>0.0</td>
<td>0</td>
</tr>
<tr>
<td>Ft53</td>
<td>53</td>
<td>MATRIX</td>
<td>6905</td>
<td>6905</td>
<td>0.0</td>
<td>0</td>
</tr>
<tr>
<td>Ft70</td>
<td>70</td>
<td>MATRIX</td>
<td>38673</td>
<td>38673</td>
<td>0.0</td>
<td>0</td>
</tr>
<tr>
<td>Kro124</td>
<td>124</td>
<td>MATRIX</td>
<td>1776</td>
<td>1776</td>
<td>0.0</td>
<td>0</td>
</tr>
<tr>
<td>Ftv150</td>
<td>150</td>
<td>MATRIX</td>
<td>1530</td>
<td>1530</td>
<td>0.0</td>
<td>0</td>
</tr>
<tr>
<td>Ftv170</td>
<td>170</td>
<td>MATRIX</td>
<td>1613</td>
<td>1613</td>
<td>0.0</td>
<td>1</td>
</tr>
<tr>
<td>Rbg323</td>
<td>323</td>
<td>MATRIX</td>
<td>1326</td>
<td>1326</td>
<td>0.0</td>
<td>3</td>
</tr>
<tr>
<td>Rbg358</td>
<td>358</td>
<td>MATRIX</td>
<td>1163</td>
<td>1163</td>
<td>0.0</td>
<td>4</td>
</tr>
<tr>
<td>Rbg403</td>
<td>403</td>
<td>MATRIX</td>
<td>2465</td>
<td>2465</td>
<td>0.0</td>
<td>2</td>
</tr>
<tr>
<td>Rbg443</td>
<td>443</td>
<td>MATRIX</td>
<td>2720</td>
<td>2720</td>
<td>0.0</td>
<td>1</td>
</tr>
</tbody>
</table>

Table 4.6 shows the performance of our LSACO algorithm for asymmetric TSP problems. It can be observed from the tables 4.1-4.6 that LSACO can find optimal results for problem size up to 10,000 nodes. Table 4.7 shows the results of LSACO on all the Hamiltonian cycle problems found in [47]. The existing ACO algorithms can also find good results for these Hamiltonian cycle problems. But our proposed LSACO finds Hamiltonian cycles in all the trials. That is, the success rate of LSACO is 100%.

Table 4.7: Performance of LSACO for Hamiltonian Cycle Problems (over 15 trials)

<table>
<thead>
<tr>
<th>Name</th>
<th>Dimension</th>
<th>Min. Time (Seconds)</th>
<th>Avg. Time (Seconds)</th>
<th>Success</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alb1000</td>
<td>1000</td>
<td>0.0</td>
<td>0.1</td>
<td>15/15</td>
</tr>
<tr>
<td>Alb2000</td>
<td>2000</td>
<td>0.1</td>
<td>0.1</td>
<td>15/15</td>
</tr>
<tr>
<td>Alb3000a</td>
<td>3000</td>
<td>0.1</td>
<td>0.1</td>
<td>15/15</td>
</tr>
<tr>
<td>Alb3000b</td>
<td>3000</td>
<td>0.1</td>
<td>0.1</td>
<td>15/15</td>
</tr>
<tr>
<td>Alb3000c</td>
<td>3000</td>
<td>0.1</td>
<td>0.1</td>
<td>15/15</td>
</tr>
<tr>
<td>Alb3000d</td>
<td>3000</td>
<td>0.1</td>
<td>0.1</td>
<td>15/15</td>
</tr>
<tr>
<td>Alb3000e</td>
<td>3000</td>
<td>0.1</td>
<td>0.1</td>
<td>15/15</td>
</tr>
<tr>
<td>Alb4000</td>
<td>4000</td>
<td>0.1</td>
<td>0.1</td>
<td>15/15</td>
</tr>
<tr>
<td>Alb5000</td>
<td>5000</td>
<td>0.2</td>
<td>0.2</td>
<td>15/15</td>
</tr>
</tbody>
</table>
Table 4.8: Experimental Results of LSACO on some larger TSP problems (over 15 trials)

<table>
<thead>
<tr>
<th>Problems</th>
<th>Dimension</th>
<th>Optimum</th>
<th>Best (Error %)</th>
<th>Average (Error %)</th>
</tr>
</thead>
<tbody>
<tr>
<td>D1655</td>
<td>1655</td>
<td>62128</td>
<td>62128 (0.00%)</td>
<td>62129 (0.00%)</td>
</tr>
<tr>
<td>U1817</td>
<td>1817</td>
<td>57201</td>
<td>57201 (0.00%)</td>
<td>57228 (0.05%)</td>
</tr>
<tr>
<td>D2103</td>
<td>2103</td>
<td>80450</td>
<td>80450 (0.00%)</td>
<td>80473 (0.03%)</td>
</tr>
<tr>
<td>Pr2392</td>
<td>2392</td>
<td>378032</td>
<td>378032 (0.00%)</td>
<td>378057 (0.01%)</td>
</tr>
<tr>
<td>pcb3038</td>
<td>3038</td>
<td>137694</td>
<td>137694 (0.00%)</td>
<td>137712 (0.01%)</td>
</tr>
<tr>
<td>Fl3795</td>
<td>3795</td>
<td>28772</td>
<td>28772 (0.00%)</td>
<td>28781 (0.03%)</td>
</tr>
<tr>
<td>fnl4461</td>
<td>4461</td>
<td>182566</td>
<td>182566 (0.00%)</td>
<td>182569 (0.002%)</td>
</tr>
<tr>
<td>Rl5915</td>
<td>5915</td>
<td>565530</td>
<td>565530 (0.00%)</td>
<td>565693 (0.03%)</td>
</tr>
<tr>
<td>pla7397</td>
<td>7397</td>
<td>23260728</td>
<td>23260728 (0.00%)</td>
<td>23265210 (0.02%)</td>
</tr>
<tr>
<td>Rl11849</td>
<td>11849</td>
<td>923288</td>
<td>923362 (0.01%)</td>
<td>923437 (0.02%)</td>
</tr>
<tr>
<td>D15112</td>
<td>15112</td>
<td>1573084</td>
<td>1573282 (0.01%)</td>
<td>1573362 (0.02%)</td>
</tr>
</tbody>
</table>

Fig 4.1: Time requirement of LSACO with respect to problem size

Table 4.8 shows the performance of LSACO on some large problems of TSP. LSACO finds very near optimum results for problem size of 15 thousands. And the average result and the percentage of error rate are very impressive for these large problems. Fig. 4.1 shows that LSACO finds optimum solutions nearly in $O(n)$ time.
4.6 Comparison

This section compares experimental results of LSACO with the best-performing ACO algorithms [3] which are ACS [22] and MMAS [34]. ACS-3-opt [22], MMAS-2-opt [34] and MMAS-3-opt [34] are the ACO algorithms when they incorporate local search. ACS-3-opt, MMAS-2-opt and MMAS-3-opt perform better than ACS and MMAS. Our proposed LSACO compares its results with those of ACS-3-opt, MMAS-2-opt and MMAS-3-opt.

Tables 4.9-4.13 shows the comparison results between LSACO and other best-performing ACO algorithms. All these results were obtained over 15 independent trials. Table 4.9 and 4.10 shows the comparison between LSACO and ACS-3-opt on some well-known symmetric and asymmetric TSP problems. LSACO obtains optimum results every time whereas ACS-3-opt obtains good results for small problems like d198 and ry48p. But when the problem size increases, the error rate of ACS-3-opt also increases.

Table 4.9: Comparison between LSACO and ACS-3-opt on some symmetric TSP problems (over 15 trials)

<table>
<thead>
<tr>
<th>Problem Name</th>
<th>ACS-3-opt</th>
<th>LSACO</th>
<th>% Error (Best Length)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Best Length</td>
<td>Average Length</td>
<td>Best Length</td>
</tr>
<tr>
<td>D198</td>
<td>15780</td>
<td>15782</td>
<td>15780</td>
</tr>
<tr>
<td>Pcb442</td>
<td>50782</td>
<td>50792</td>
<td>50778</td>
</tr>
<tr>
<td>Att532</td>
<td>27693</td>
<td>27718</td>
<td>27686</td>
</tr>
<tr>
<td>Rat783</td>
<td>8818</td>
<td>8838</td>
<td>8806</td>
</tr>
<tr>
<td>fl1577</td>
<td>22352</td>
<td>22484</td>
<td>22249</td>
</tr>
</tbody>
</table>

The performance of ACS-3-opt and LSACO on the asymmetric problems is nearly the same. Because the asymmetric TSP problems contain small sized problems. Although
for these small problems LSACO and ACS-3-opt obtains the same quality solution, LSACO requires less time to find the optimum.

**Table 4.10:** Comparison between LSACO and ACS-3-opt on some asymmetric TSP problems (over 15 trials)

<table>
<thead>
<tr>
<th>Problem Name</th>
<th>ACS-3-opt</th>
<th>LSACO</th>
<th>Optimum</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Avg. (length)</td>
<td>Avg. (sec)</td>
<td>%Error</td>
</tr>
<tr>
<td>Ry48p</td>
<td>14422</td>
<td>1</td>
<td>0.00%</td>
</tr>
<tr>
<td>Ft70</td>
<td>38679</td>
<td>0</td>
<td>0.02%</td>
</tr>
<tr>
<td>Kro124p</td>
<td>36230</td>
<td>2</td>
<td>0.00%</td>
</tr>
<tr>
<td>Ftv170</td>
<td>2755</td>
<td>4</td>
<td>0.00%</td>
</tr>
</tbody>
</table>

**Table 4.11:** Comparison of LSACO with MMAS on symmetric problems of TSP (over 15 trials)

<table>
<thead>
<tr>
<th>Problem Name</th>
<th>MMAS</th>
<th>LSACO</th>
<th>Optimum</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Best Length</td>
<td>%Error</td>
<td>Best Length</td>
</tr>
<tr>
<td>D198</td>
<td>15790</td>
<td>0.06%</td>
<td>15780</td>
</tr>
<tr>
<td>Lin318</td>
<td>42225</td>
<td>0.46%</td>
<td>42029</td>
</tr>
<tr>
<td>Pcb442</td>
<td>51305</td>
<td>1.03%</td>
<td>50778</td>
</tr>
<tr>
<td>Att532</td>
<td>28101</td>
<td>1.49%</td>
<td>27686</td>
</tr>
<tr>
<td>Rat783</td>
<td>9010</td>
<td>2.31%</td>
<td>8806</td>
</tr>
<tr>
<td>U1060</td>
<td>225997</td>
<td>0.84%</td>
<td>224094</td>
</tr>
<tr>
<td>Pcb1173</td>
<td>57898</td>
<td>1.76%</td>
<td>56892</td>
</tr>
<tr>
<td>D1291</td>
<td>52202</td>
<td>2.75%</td>
<td>50801</td>
</tr>
<tr>
<td>Fl1577</td>
<td>23109</td>
<td>3.86%</td>
<td>22249</td>
</tr>
</tbody>
</table>

**Table 4.12:** Comparison of LSACO with MMAS-2-opt on symmetric problems of TSP (Over 15 trials)

<table>
<thead>
<tr>
<th>Problem Name</th>
<th>MMAS-2-opt</th>
<th>LSACO</th>
<th>Optimum</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Best Length</td>
<td>%Error</td>
<td>Best Length</td>
</tr>
<tr>
<td>Kroa100</td>
<td>21282</td>
<td>0.00%</td>
<td>21282</td>
</tr>
<tr>
<td>D198</td>
<td>15786</td>
<td>0.04%</td>
<td>15780</td>
</tr>
<tr>
<td>Lin318</td>
<td>42195</td>
<td>0.39%</td>
<td>42029</td>
</tr>
<tr>
<td>Pcb442</td>
<td>51212</td>
<td>0.85%</td>
<td>50778</td>
</tr>
<tr>
<td>Att532</td>
<td>27911</td>
<td>0.81%</td>
<td>27686</td>
</tr>
<tr>
<td>Rat783</td>
<td>8976</td>
<td>1.93%</td>
<td>8806</td>
</tr>
</tbody>
</table>
Table 4.13: Comparison of LSACO with MMAS-3-opt on asymmetric problems of TSP (Over 15 trials)

<table>
<thead>
<tr>
<th>Problem Name</th>
<th>Best Length</th>
<th>%Error</th>
<th>Best Length</th>
<th>%Error</th>
<th>Optimum</th>
</tr>
</thead>
<tbody>
<tr>
<td>P43</td>
<td>5623.8</td>
<td>0.07%</td>
<td>5621</td>
<td>0.02%</td>
<td>5620</td>
</tr>
<tr>
<td>Ry48p</td>
<td>14494</td>
<td>0.50%</td>
<td>14422</td>
<td>0.00%</td>
<td>14422</td>
</tr>
<tr>
<td>Ft70</td>
<td>38707</td>
<td>0.09%</td>
<td>38673</td>
<td>0.00%</td>
<td>38673</td>
</tr>
<tr>
<td>Kro124p</td>
<td>36655</td>
<td>1.17%</td>
<td>36230</td>
<td>0.00%</td>
<td>36230</td>
</tr>
<tr>
<td>Ftv170</td>
<td>2807</td>
<td>1.89%</td>
<td>2755</td>
<td>0.00%</td>
<td>2755</td>
</tr>
</tbody>
</table>

Tables 4.11-4.13 show the experimental results of MMAS, MMAS-2-opt, MMAS-3-opt [50] and our proposed LSACO. MMAS performs poor when applied to problems size more than 100. But our proposed LSACO obtains optimum result every time. It can be observed from these tables that our proposed LSACO performs better than all these ACO algorithms and this LSACO can even perform well for problems of size more than 10 thousand nodes.

Fig 4.2: Comparison of error rates of LSACO and other algorithms with respect to problem dimension

Fig. 4.2 shows the error rate of LSACO and other algorithms. The error is computed with respect to the best length found in any of the 15 trials. For small problem sizes, ACS-3-opt and MMAS-3-opt can find near optimum solutions. When the problem size increases, the performance of MMAS-3-opt and ACS-3-opt decreases gradually. The proposed LSACO algorithm can find very good optimum solutions even for large
problem instances. This figure also shows that MMAS-3-opt produces inferior solutions compared to ACS-3-opt. But our proposed LSACO algorithm finds best results among these algorithms.

Fig. 4.3: Comparison of required time to find the optimum solution

Fig. 4.3 shows the best time of LSACO and other algorithms to produce the optimum result among 15 trials. ACS-3-opt produces good quality solutions more quickly than MMAS-3-opt. It can be also observed from this figure that LSACO requires less and nearly linear time to produce the optimum solution.

Based on the above comparisons, it is clear that LSACO performed better than other algorithms in most cases. Although such comparisons may not be entirely fair due to different experimental setups, we have tried our best to make our experimental setup as close to the previous ones as possible.

4.7 Discussions

This section briefly explains why the performance of LSACO is better than the existing ACO algorithms. There are some major differences that might contribute to better performance by LSACO in comparison with other existing ACO algorithms. The first and the most important reason is that LSACO uses adaptive local search to improve the solution quality. Since the local search used by the existing ACO algorithm uses fixed number of edges to exchange, they get stuck in local optima
when applied to large problems. To speed up the adaptive local search, LSACO uses the concept of don’t look bit to avoid fruitless search and thus reduces the time to find better solutions. Our proposed LSACO also uses a dynamic ordering of the candidates to speed up the search even more.

Second, LSACO uses different exploration and exploitation phase compared to the other ACO algorithms. LSACO initializes pheromone in different way by setting it to a medium value so as to balance exploration and exploitation at the start of the algorithm. Third, the state transition rule that is used by the LSACO helps to avoid early stage of local optima.

Fourth, the local and global pheromone updating strategies that are used by the proposed LSACO are different from the other existing ACO algorithms. In LSACO, pheromone is not decreased in all edges in local pheromone updating. Only the ants are allowed to decrease pheromone. Moreover, in LSACO global updating rule is used by only a single ant to increase pheromone.

Thus, LSACO requires less time in all the phases throughout the algorithm compared to the existing ACO algorithms. The solution quality is also better due to the balanced exploration and exploitation of the search space. Thus, LSACO shows a great stability in finding good quality solutions in a reasonable time. So, LSACO can be very good alternative for the existing ACO algorithms for its capability of applying to larger problems.
5.1 Conclusive Remarks

ACO algorithms have been introduced to the swarm intelligence community for nearly a decade. Since then several ACO algorithms have been proposed and they were successfully applied to a large number of difficult combinatorial optimization problems. But all the existing ACO algorithms were applied to small problem instances. For larger problem instances, their performance degrades in finding good quality solutions. Their time requirement also increases exponentially with the problem size. This thesis proposes a new adaptive local search based ACO algorithm that addresses these issues.

Our new algorithm LSACO adopts several modifications and enhancements to increase the accuracy of the solution and to decrease the time complexity. The LSACO algorithm is described in chapter 3. LSACO applies new adaptive local search to avoid local optima and thus to obtain optimum solution in a reasonable time. The pheromone updating rules are applied in such a way so as to balance the exploration and exploitation of the search phase. Several other strategies have been adopted to increase the overall performance of the LSACO algorithm.

Extensive experiments have been carried out in this thesis to evaluate how well LSACO performed on different problems in comparison with other existing ACO algorithms. These results are shown in chapter 4 of this thesis. In almost all cases, LSACO outperformed the others. The results have showed clearly the advantages of LSACO. For example, LSACO’s solution quality was better than the best-performing ACO algorithm ACS. The time required to find the optimum solutions was also very impressive for all the problems.
Although LSACO has performed very well for almost all problems we tested, our experimental study appeared to have revealed a weakness of LSACO in dealing with the very large problems containing nodes more than fifteen thousand. In these large problems, LSACO produced a small amount of error. This error was generated because of getting stuck in local optima for very large problems. However, the results from LSACO were actually better than the existing ACO algorithms.

5.2 Recommendations for Future Research

There are many ways in which the research work may be enhanced or expanded. It would be interesting in the future to analyze LSACO further and identify its strength and weakness. Potential hybridization between LSACO and other evolutionary algorithms would also be an interesting future research topic. The other possible fields in which the LSACO can be applied in near future are as follows:

- To increase the number of edges that is to be exchanged while not increasing the time complexity.
- To combine other strategies to get even better results for very large problems.
- To apply LSACO in new applications those have not been applied yet.
- To parallelize the whole algorithm using multiple processors.
- The proposed LSACO can be applied more efficiently than the existing genetic i.e., evolutionary programming in various optimization problems.

So, the proposed LSACO algorithm may be very useful for the researchers in doing experiments in different fields.
REFERENCES


