A Metaheuristic Ant Colony Optimization Algorithm for Symmetric and Asymmetric Traveling Salesman Problems

A thesis submitted for the degree of Doctor of Philosophy

by

Lilysuriazna Binti Raya

Department of Mathematics

Brunel University London

March 2018

Abstract

This research addresses solving two types of Travelling Salesman Problems (TSP) which are the symmetric TSP (STSP) and the asymmetric TSP (ATSP). The TSP is a problem of finding a minimal length closed tour that visits each city once. If the distances between each pair of cities are the same in both directions, the problem is a STSP, otherwise, it is an ATSP.

In this thesis, a new metaheuristic algorithm which is based on Ant Colony Optimization (ACO) is proposed to solve these problems. The key idea is to enhance the ability of exploration and exploitation by incorporating a metaheuristic approach with an exact method. A new strategy for creating 'Intelligent Ants' is introduced to construct the solution tours. This strategy aims at reducing the computational time by heuristically fixing part of the solution tour and improving the accuracy of the solutions through the usage of a solver, specifically for large size instances. Moreover, this proposed algorithm employs new ways of depositing and evaporating pheromone. A different approach of global updating of pheromone is proposed in which the pheromone is deposited only on the edges belonging to the colony-best ant and evaporated only on the edges belonging to the colony-worst ant that are not in the colony-best ant.

Additionally, the parameters of the proposed algorithm which include the number of colonies, the number of ants in each colony, the relative influence of the pheromone trail α and the pheromone evaporation rate ρ are expressed as a function of the problem size. Comparisons with other sets of parameter values suggested in the literature have been investigated which illustrate the advantages of the choice of the proposed parameter settings.

Further, in order to evaluate the performance of the proposed algorithm, a set of standard benchmark problems from the TSPLIB with up to 442 cities were solved and the results obtained were compared with other approaches from the literature. The proposed algorithm has proven to be competitive and shows better performance in 63% of the 16 algorithms in terms of solution quality and obtained the optimum solutions in 70% of the 33 instances, proving that it is a good alternative approach to solve these hard combinatorial optimisation problems.

Acknowledgements

"In the name of Allah, the Most Gracious and the Most Merciful"

Alhamdulillah, all praises to Allah for the strengths and His blessing in completing this thesis.

Firstly, I would like to express my sincere appreciation and gratitude to my supervisor Dr Cormac A. Lucas for his guidance, useful advice, and encouragement. His invaluable help of constructive comments and suggestions throughout the experimental and thesis works have contributed to the success of this research. Not forgetting Dr Paresh Date and Dr Diana Roman for their insightful feedback and suggestions to widen the research perspectives.

I am also deeply thankful to the Brunel Graduate School for their support and help towards my postgraduate affairs. My acknowledgement also goes to all the technicians and office staffs of Department of Mathematics, Brunel University London especially to Mr Nalin Soni for their co-operation.

I wish to express my unqualified thanks to my parents, Hj Raya Bin Hj Yunus and Hjh Noorizan Binti Mohd Saad for their tremendous encouragements and prayers that always guided me to the right way. I could never have accomplished this dissertation without your love, support and understanding. To my brother, sister and other family members: thank you for your eternal love and support.

Sincere thanks to my beloved friends who have been my cheerleaders! Your kind words and encouragement have carried me through this journey and your hugs have kept me sane.

Last but not least, special appreciation and gratefulness to my home country, Malaysia and Majlis Amanah Rakyat (MARA) for the financial support throughout my PhD studies.

Thank you so much!

Declaration

I, Lilysuriazna Binti Raya, here declare that the work presented in his thesis was carried out by myself at Brunel University London except where references are cited, and no part of this work has been previously submitted to Brunel University London, nor any other academic institution, for admission to a higher degree.

Lilysuriazna Binti Raya

March 2018

List of Figures

List of Tables

List of Algorithms

Acronyms

ABC	Artificial Bee Colony
ABO	African Buffalo Optimization
ACO	Ant Colony Optimization
ACS	Ant Colony System
AEAS	Annealing Elitist Ant System
AS	Ant System
ATSP	Asymmetric Traveling Salesman Problem
B&B	Branch-and-bound
B&C	Branch-and-cut
BA	Bat Algorithm
BCO	Bee Colony Optimization
BWAS	Best-Worst Ant System
COP	Combinatorial Optimization Problem
EAS	Elitist Ant System
FLC	Fuzzy Logic Controller
GA	Genetic Algorithm
GVNS	Guided Variable Neighbourhood Search
IA	Intelligent Ants
IBA	Improved Bat Algorithm
IGA	Improved Genetic Algorithm
IP	Integer Programming
LP	Linear Programming

MACO	Modified Ant Colony Optimization (The Proposed Algorithm)
MACOd	Modified Ant Colony Optimization using Dorigo's Parameter Value
MIP	Mixed Integer Programming
MMAS	Max-Min Ant System
mTSP	Multiple Traveling Salesman Problem
NN	Nearest-neighbour
NN2-opt	Nearest-neighbour with 2-opt
PS-ACO	Particle Swarm-Ant Colony Optimization
PSO	Particle Swarm Optimization
RAI	Randomized Arbitrary Insertion
R-AS	Rank-based Ant System
SA	Simulated Annealing
SEE	Pheromone Correction Strategy
SI	Swarm Intelligence
SOQ	Sequential Ordering Problem
SSA	Swarm Simulated Annealing
STSP	Symmetric Traveling Salesman Problem
TS	Tabu Search
TSP	Traveling Salesman Problem

Contents

Chapter 1

Introduction

An optimization problem is defined by a set of decision variables, an objective function and a possible set of constraints. It is a process of minimizing or maximizing the objective function by finding values of the decision variables that satisfy the set of constraints. Let f be an objective function, X a feasible set and $\mathbb{R}$ a solution space. The optimization problem is formulated as:

$$\max/\min \ \{f(x)|x \in X, X \subseteq \mathbb{R}\} \tag{1.1}$$

If $x \in X$ then x is called a feasible solution. Otherwise, x is an infeasible solution. In general, optimization models are classified according to characteristics of the decision variables, constraints or objective function. If the classification is based on the nature of the equations for the objective function and the constraints, the optimization problem is classified as linear or nonlinear problems. On the other hand, if the classification is based on types of variables, the optimization problems are classified as continuous optimization or discrete optimization problems. Further, if the solution space is discrete, then the problem is a combinatorial optimization problem (COP). The COPs are easy to state but difficult to solve (Osman & Kelly, 1996). Additionally, many COPs can be formulated as mathematical programming problems.

Meanwhile, the optimization problem is a linear programming problem (LP) if the objective function is a linear function in the form of (Dantzig, 1963):

$$c_1x_1 + c_2x_2 + \cdots + c_nx_n \tag{1.2}$$

for some $c_i \in \mathbb{R}$, $i = 1, \dots, n$. Besides, the feasible region is the solution set to a finite number of linear inequality or equality constraints associated with some linear combination of the decision variables in the form:

$$a_1x_1 + a_2x_2 + \cdots + a_nx_n \begin{Bmatrix} \leq \\ = \\ \geq \end{Bmatrix} b \tag{1.3}$$

In other words, a feasible solution is a solution for which all the constraints are satisfied while the feasible region is the set containing all these feasible solutions.

Mathematically, these linear programming models can be defined as a method of finding the maximum or minimum value of the objective function satisfying a set of linear constraints. In particular, it is a process of finding the values of decision variables x_j that maximize or minimize the objective function. It can be written as follows:

$$\text{max/min} \sum_{j=1}^{n} c_j x_j \tag{1.4}$$

subject to

$$\sum_{j=1}^{n} a_{ij} x_j \begin{Bmatrix} \leq \\ = \\ \geq \end{Bmatrix} b_i \qquad i = 1, \dots, m \tag{1.5}$$

$$x_j \geq 0 \qquad j = 1, \dots, n \tag{1.6}$$

where c_j, a_{ij} and b_i are constants.

Since the decision variables that satisfy all the linear constraints of the problem are called a feasible solution, then x is a feasible solution if $x \in X$. Otherwise, x is an infeasible solution.

Furthermore, if some or all of the variables in 1.6 are restricted to integer values, then, the LP is an *integer programming* (IP). Three types of IP models are:

- Pure integer programming : if all the variables are integers.
- Mixed integer programming : if only some of the variables are restricted to be integer values.
- Binary integer programming : all the variables are binary (restricted to the values of 0 or 1).

Hence, if there are no integer variables, then it is an LP, if no continuous variables are present, then it is a pure IP, and if both integer and continuous variables are present, then it is a mixed integer programming (Kolman & Beck, 1995).

An example of a widely studied combinatorial optimization problem is the Traveling Salesman Problem (TSP). This research is aimed on finding the optimal solution or an approximate solution to two types of TSPs. In the next section, there are definitions, classifications, and formulations for the TSPs.

1.1 Classification of the Travelling Salesman Problem

The TSP is defined by having a set of cities where the distances between each city pair is known. The problem is to plan a route that visits each city once and ends where it starts.

The simplicity of its formulation has led to numerous remarkable applications in many areas such as vehicle routing (Bektas, 2006), data transmission in a computer network (Ali & Kamoun, 1993), scheduling (Bigras, Gamache, & Savard, 2008) and (Baez, Angel-Bello, & Alvarez, 2016), air traffic management (Furini, Persiani, & Toth, 2016), printed-circuit-boards manufacturing (Fujimura, Fujiwaki, Kwaw, & Tokataka, 2001) , robot navigation (Barral, Perrin, Dombre, & Liegeois, 1999) and data partitioning (Cheng, Lee, & Wong, 2002). In these instances, the main goal is to find the optimal tour when the cost or distance between each location is known. In brief, it is a process of determining an order of how each location is visited once and the total cost incurred or the total distance travelled is a minimum.

The general form of the TSP was first studied by mathematicians starting in the 1920's in Vienna, notably by Karl Menger (Applegate, Bixby, Chvatal, & Cook, 1998). The TSP then was studied by Princeton's mathematical community during the 1930's. In the 1940's, Merrill Flood publicised the name TSP within the mathematical community on mass (Lawler, Lenstra, Kan, & Shmoys, 1985).

The TSP can be classified into different classes according to the properties of the cost matrix or the type of graph. The common classification of TSP is symmetric TSP (STSP), asymmetric TSP (ATSP), and mult TSP (mTSP).

1.1.1 Asymmetric Traveling Salesman Problem

The asymmetric TSP is defined as the problem of finding a minimal length closed tour that visits each city once. The distances between each pair of cities are not necessarily the same in both directions. In general, paths may not even exist in both directions.

Consider a graph $G = (V, E)$, where $V = \{v_1, \dots, v_n\}$ is a set of n cities, and $E = \{(i, j): i, j \in V\}$ is the set of arcs. Let x_{ij} be a decision variable and $C = (c_{ij})$ is a cost matrix associated with edge $(i, j) \in E$. The formulation of the asymmetric TSP can be stated as follows (Punnen, 2007):

$$\text{minimize} \sum_{\substack{i,j \in E \\ i \neq j}} c_{ij} x_{ij} \tag{1.7}$$

subject to

$$\sum_{i=1}^{n} x_{ij} = 1 \qquad j = 1, \dots, n \tag{1.8}$$

$$\sum_{j=1}^{n} x_{ij} = 1 \qquad i = 1, \dots, n \tag{1.9}$$

$$+ \text{ subtour elimination constraints,} \tag{1.10}$$

$$x_{ij} = 0 \text{ or } 1, \qquad (i, j) \in E \tag{1.11}$$

Constraints 1.8 and 1.9 are the in-degree and out-degree for each vertex, which ensures that each vertex leaves and enters each node exactly once; constraints 1.10 ensure that the tour is connected with no subtour while constraints 1.11 is the integrality constraint.

1.1.2 Symmetric Traveling Salesman Problem

The symmetric TSP is a particular case of the asymmetric TSP (Punnen, 2007) . The symmetric TSP implies that the distance between two cities is the same in each direction. Additionally, this symmetry halves the number of feasible solutions.

Consider the graph $G = (V, E)$ defined previously in Section 1.1.1. The formulation of the symmetric TSP is given by (Punnen, 2007):

$$\text{minimize} \sum_{\substack{i,j \in E \\ i < j}} c_{ij} x_{ij} \tag{1.12}$$

subject to

$$\sum_{i<k} x_{ik} + \sum_{j>k} x_{kj} = 2 \qquad i,j,k = 1,\dots,n \tag{1.13}$$

$$+ \text{subtour elimination constraints,} \tag{1.14}$$

$$x_{ij} = 0 \text{ or } 1, \qquad i,j \in E \tag{1.15}$$

Constraints 1.13 are referred to as degree constraints, enforcing that the tour uses exactly two of the edges incident on each node; constraints 1.14 and 1.15 are subtour elimination constraints and integrality constraints,respectively.

If the distances c_{ij} satisfy the triangle inequality $c_{ij} + c_{jk} \geq c_{ik}$ for all distinct cities i,j,k then, the symmetric TSP is a metric TSP. The triangle inequality ensures that a direct path between two vertices is at least as short as any indirect path. If the distances c_{ij} satisfies the Euclidean norm and obeys the triangle inequality, then the metric TSP is a Euclidean TSP.

1.1.1.1 A Euclidean TSP

An instance of a Euclidean TSP is a complete graph G with its vertices in a Euclidean space. The distance of any two vertices $v_i = (x_i, y_i)$ and $v_j = (x_j, y_j)$ in a Euclidean TSP is given by $d(v_i, v_j) = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$. In other words, the cost is the length of the straight line between the two points.

1.1.3 The Multi Traveling Salesman Problem

The Multi Traveling Salesman Problem (mTSP) is a generalization of the TSP which consists of a set of n nodes (cities) and a set of m salesmen located at a single depot node. The remaining cities that are to be visited are called *intermediate nodes*. Therefore, it is a problem of determining a set of routes for m salesmen who start and end at the depot or home city such that each *intermediate node* is visited exactly once and the total cost of visiting all nodes is minimized (Bektas, 2006). Note that the cost metric can be defined in terms of distance, time, etc. Some possible variations of the problem are:

- Single depot : In the single depot case, all salesman start from and end their tours at a single central depot.

- Multiple depots: In the multiple depot cases with a number of salesmen located at each, the final destination of the salesman can either be their original depot or any depot with the restriction that, at each depot, the initial number of the salesman remains the same after all the travel.
- Number of salesmen: The number of salesman appearing in the problem may be a bounded variable or fixed a priori.
- Fixed charges: If the number of salesman in the problem is a bounded variable, the usage of each salesman in the solution usually has an associated fixed cost. In this case, the minimization of this bounded variable may also be of concern.
- Time windows: Certain cities need to be visited in specific time periods, named as time windows. This extension of the mTSP is referred to as mTSP with Time Windows (mTSPTW).
- Other restrictions: These additional constraints may consist of bounds on the number of nodes each salesman visits, the maximum or minimum distance or travelling duration a salesman travels or other special restrictions.

1.2 Different Formulations of the TSP

The standard formulation for the TSP was suggested by Dantzig, Fulkerson and Johnson (G.B.Dantzig, Fulkerson, & Johnson, 1954) in 1954. Dantzig, Fulkerson and Johnson formulated the problem as a zero-one linear program involving $O(n^2)$ variables and $O(2^n)$ linear constraints (Padberg & Sung, 1991). Since their formulation has an exponentially large number of constraints, many researchers have proposed alternative formulations of the TSP that involve only a polynomial number of constraints. For the sake of brevity, only three are mentioned here which are Miller, Tucker and Zemlin (Miller, Tucker, & Zemlin, 1960), Gavish and Graves (Gavish & Graves, 1978) and Claus (Claus, 1984).

1.2.1 The Dantzig-Fulkerson-Johnson Formulation

The Dantzig-Fulkerson-Johnson (DFJ) formulation associates a binary variable x_{ij} with each edge (i, j) which is equal to 1 if and only if the edge appears in the optimal tour. They have

formulated the subtour elimination constraints 1.10 and 1.14 as follows (G.B.Dantzig, Fulkerson, & Johnson, 1954):

$$\sum_{i,j\in S} x_{ij} \leq |S| - 1 \qquad \forall\, S \subset V\,, S \neq \emptyset \tag{1.16}$$

The subtour elimination constraints 1.16 are derived based on the fact that the number of nodes for every tour or subtour must be equal to the number of arcs. Thus, to form a tour without forming a subtour, for every subset that consists of 2 or $n-1$ cities , the number of arcs must be less than the number of cities, where S is a nonempty proper subset of all nodes V, and $|S|$ is the number of cities in S. Otherwise, constraints 1.16 are violated since its left-hand side value would be equal to $|S|$ and its right-hand side value equal to $|S|-1$. Moreover, constraints 1.16 are only valid for $2 \leq |S| \leq n-2$ due to constraints 1.8,1.9 and 1.13.

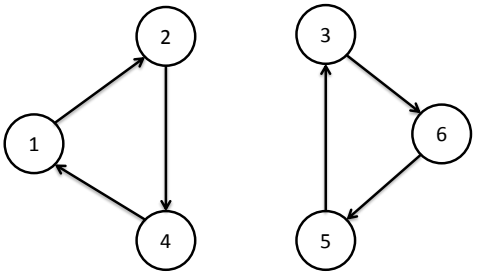


Figure 1.1: Illustrative example of DFJ subtour elimination constraints

Consider the subtour $1 \to 2 \to 4 \to 1$ in Figure 1.1 with $V = \{1,2,3,4,5,6\}$ and $S = \{1,2,4\}$. This subtour could not be satisfied since $\sum_{i,j\in S} x_{ij} \nleq |S| - 1$, that is

$$\sum_{i,j\in S} x_{ij} = \quad x_{12} + x_{24} + x_{41} = 3 \text{ and } |S| - 1 = 2$$

Thus the corresponding constraints 1.16 would be violated.

Likewise, the subtour $3 \to 6 \to 5 \to 3$ in Figure 1.1 with $S = \{3,5,6\}$ would also violate constraints 1.16 since

$$\sum_{i,j\in S} x_{ij} = \quad x_{36} + x_{65} + x_{53} = 3 \text{ and } |S| - 1 = 2.$$

1.2.2 The Miller-Tucker-Zemlin Formulation

Alternatively, Miller, Tucker and Zemlin (MTZ) proposed another formulation for a more general TSP using a polynomial number of subtour elimination constraints by introducing additional continuous variables u_i namely node potentials. The MTZ subtour elimination constraints for 1.10 and 1.14 are then stated as (Miller, Tucker, & Zemlin, 1960) :

$$u_i - u_j + nx_{ij} \leq n - 1 \quad \forall i \neq j = 2, \dots, n \tag{1.17}$$

The subtour elimination constraints 1.17 indicate the order of the corresponding node in the tour. The formulation involves $O(n^2)$ variables and $O(n^2)$ constraints (Oncan, Altinel, & Laporte, 2009).

The indices u_i represent the position of node $i \geq 2$ in the tour assuming that the tour starts at node 1. Hence, node 1 has position 1. The constraints defining node potential prevent subtours by providing a simple contradiction upon surrogating the last set of constraints in 1.17 over nodes in any subtour that does not involve the node 1.

For example, consider the subtour $3 \rightarrow 6 \rightarrow 5 \rightarrow 3$ in Figure 1.1 with $x_{36} = x_{65} = x_{53} = 1$ and $u_3 = 1, u_6 = 2$ and $u_5 = 3$. From constraints 1.17, three corresponding constraints are:

$$u_3 - u_6 + 6x_{36} = 1 - 2 + 6(1) = 5 \leq 5 \quad \text{(satisfied)}$$

$$u_6 - u_5 + 6x_{65} = 2 - 3 + 6(1) = 5 \leq 5 \quad \text{(satisfied)}$$

$$u_5 - u_3 + 6x_{53} = 3 - 1 + 6(1) = 8 > 5 \quad \text{(violated)}$$

It can be seen clearly that the subtour violates the last constraint.

1.2.3 The Gavish and Graves Formulation

In 1978, Gavish and Graves (G&G) proposed a single commodity flow problem that extends the MTZ formulation of TSP. Both MTZ and G&G use the same number of variables but differ in their constraints set. Imagine that the salesman carries $n - 1$ units of commodity when he leaves node 1, and delivers 1 unit of this commodity to each other node. Let the x_{ij} be defined as in DFJ, G&G uses additional flow variables y_{ij} indicating the amount of flow

on arc (i,j). The G&G formulation of the subtour elimination constraints are stated as (Gavish & Graves, 1978):

$$\sum_{\substack{j=1 \\ j\neq i}}^{n} y_{ij} - \sum_{\substack{j=2 \\ j\neq i}}^{n} y_{ji} = 1 \qquad i = 2, \dots, n \tag{1.18}$$

$$0 \leq y_{ij} \leq (n-1)x_{ij} \qquad 1 \leq i,j \leq n, \quad j \neq i \tag{1.19}$$

$$x_{ij} = 0 \text{ or } 1, \qquad i,j \in E \tag{1.20}$$

For fixed values of the variables x_{ij} , the two sets of constraints 1.18 and 1.19 yield a network flow problem in which the variables y_{ij} take integer values. Constraints 1.18 guarantee that each client receives one unit of flow; constraints 1.19 ensure that the flow on each arc leaving the nodes does not exceed a specified capacity. The G&G formulation has $O(n^2)$ variables and $O(n^2)$ constraints (Oncan, Altinel, & Laporte, 2009) .

1.2.4 The Claus Formulations

Moreover, Claus presented a multi-commodity flow formulation with $n-1$ commodities, 1 unit of each for each customer. This formulation introduces non-negative continuous variables w_{ij}^k representing the amount of the kth commodity passing directly from node i to node j. The formulation is as follows (Claus, 1984):

$$\sum_{i=1}^{n} w_{ij}^k - \sum_{i=2}^{n} w_{ji}^k = 0 \qquad j,k = 2, \dots, n \ , j \neq k \tag{1.21}$$

$$\sum_{i=2}^{n} w_{1i}^k = 1 \qquad k = 2, \dots, n \tag{1.22}$$

$$\sum_{i=1}^{n} w_{ik}^k = 1 \qquad k = 2, \dots, n \tag{1.23}$$

$$0 \leq w_{ij}^k \leq x_{ij}, \qquad i,j = 1, \dots, n; \quad i \neq j; \quad k = 2, \dots, n \tag{1.24}$$

$$x_{ij} = 0 \text{ or } 1, \qquad i,j \in E \qquad 1.25$$

Constraints 1.21 ensure that a commodity leaves a vertex if it is not its final destination; constraints 1.22 guarantee that each commodity leaves the depot and is delivered to a vertex; constraints 1.23 ensure that each vertex gets exactly one commodity. The coupling constraints 1.24 indicate that flow of any commodity is allowed on the arc (i,j) only if that arc is included in the solution. In other words, variable w_{ij}^{k} is equal to 1 if and only if the commodity going from vertex 1 to vertex k flows on arc (i,j).

The Claus formulation involves $O(n^3)$ variables and $O(n^3)$ constraints (Oncan, Altinel, & Laporte, 2009).

1.3 Solution Methods to the Travelling Salesman Problem

The TSP is a typical example of hard combinatorial optimization problems that have intrigued mathematicians and computer scientists for years. Traditional optimization techniques such as linear programming, non-linear programming and dynamic programming have frequently been used for solving the TSP. However, these algorithms are computationally expensive since they examine potentially every feasible solution in order to identify the optimum solution. For this reason, heuristic or approximate algorithms are often preferred to exact algorithms for solving the large TSPs in practice.

1.3.1 Exact Algorithms

These types of algorithms theoretically are guaranteed to find an optimal solution although they can only be successfully used for modestly sized problem instances. Such methods include brute-force, cutting planes, branch-and-bound, branch-and-cut, and dynamic programming.

1.3.1.1 Brute-force Method

A brute force method is a straightforward approach directly based on the problem's statement and definitions of the concepts involved. The brute-force method for solving the TSP would be to check every possible route and take the shortest one as the answer. If there are N cities

to choose from, then there will be $(n-1)!$ maximum possible Hamiltonian cycles for an asymmetric TSP (directed graph) and $(n-1)!/2$ maximum possible Hamiltonian cycles for a symmetric TSP (undirected graph).

However, the factorial function $(n-1)!$ grows larger when n becomes larger as it needs to generate $(n-1)!$ number of permutations for all the *n* cities. Hence, the time needed for solving the TSP grows enormously even for modern computers.

1.3.1.2 Branch-and-bound

The branch-and-bound (B&B) was first proposed by (Land & Doig, 1960) in the 1960s and is typically used to solve discrete optimisation problems. The underlying idea of a B&B algorithm lies in finding an optimal solution and proving its optimality by successive partitioning of the feasible set. The word 'branch' in the B&B refers to the partitioning process while the word 'bound' refers to the process of defining lower bounds for subproblems. In other words, it is a method of searching for an optimal solution based on the successive partitioning of the solution space.

The solution of a problem with a B&B algorithm is usually described as a search through a search tree in which the root node corresponds to the original problem and each other node is a subproblem of the original problem. For most cases, a feasible solution to the problem is produced in advance using a heuristic and the value thereof is used as the current best solution or incumbent. In each iteration of a B&B algorithm, a node is selected for exploration from the list of live nodes corresponding to unexplored feasible subproblems using some selection strategy. It is then followed by the branching strategy in which two or more children of the selected node are constructed through the addition of constraints to the subproblem of the node. In this way, the subspace is subdivided into smaller subspaces. For each of these subproblems, the bound for the node is calculated, and this value is used to discard certain subproblems from further consideration. When there are no more unexplored parts of the solution space left, the search in the whole B&B tree terminates and the optimal solution is the value of the current best solution.
In general, the essential components of the B&B for a discrete optimisation problem are:

- A relaxation of the original problem: Relaxation means some or all constraints are dropped which result in additional feasible solutions. A good relaxation is the one

that gives a strong lower bound and is easy to solve. A common form of relaxation is linear relaxation, which removes the requirement that the variables be integers (Miller S. J., 2017). A Lagrangian relaxation removes complicating constraints by adding them to the objective function, assigned with a Lagrangian multiplier (Williams, 2006). The bounds calculated by Lagrangian relaxation are tight yet computationally demanding.

- The quality of the upper bound and lower bound: Upper bound is the value of the best feasible solution or incumbent. Lower bound is the value of the objective function to the current node, which is not possible to reach any successor node with a smaller value than this lower bound in case the current node is expanded further. If the upper bound of any subproblem is larger than the incumbent, the subproblem is added to the list of active nodes to be expanded further during the search. If not, then it will be fathomed.
- The branching rule: The branching rule determines how children (newly generated nodes) are generated from a subproblem. A good branching rule is one that generates few successors of a node of the search tree, and generates strongly constrained subproblems (Balas & Toth, 1983). Some of the most common variable selection strategies for MIP are the most infeasible rule, strong branching and pseudo-cost branching.
- The subproblem (node) selection rule: The subproblem selection rule determines the order in which unexplored subproblems are selected for exploration. Three popular ways of subproblem selection are breadth-first, depth-first search and heuristic search.

In addition to these, a key element to a good B&B algorithm is a low initial incumbent solution to facilitate fathoming of the nodes as early as possible. For this reason, the initial feasible solution is normally obtained by a heuristic or metaheuristic. If no such heuristic exists, the initial value of the incumbent is set to infinity. Besides, pruning or fathoming is significant in minimising the number of nodes generated in the B&B search tree by removing regions of the search space that cannot lead to a better solution. Three types of pruning are:

(i) Pruning by infeasibility: The subproblem has no feasible linear programming solution, and further partitioning would not lead to feasibility again.

(ii) Pruning by optimality: The subproblem has an integer optimum, and further partitioning would not improve the solution.

(iii) Pruning by bound: The upper bound of the subproblem optimum is less than or equal to the lower bound of the original problem.

In summary, the efficiency of the B&B method is dependent on the sharpness of the bound and the effort involved in computing the bound. A good upper bound is significant to keep the size of the B&B tree as small as possible (Raidl & Puchinger, 2008). Therefore, a heuristic or metaheuristic is usually applied at some nodes in the search tree (Raidl & Puchinger, 2008). Also note that although a feasible solution is often found early in the B&B search tree, the confirmation of optimality requires longer CPU time to be proved (Dowsland, 2014).

1.3.1.3 Cutting Plane

This method solves optimization problems through a series of relaxations whose feasible sets are progressively tightened through the addition of valid inequalities. These valid inequalities are called cuts. The added cuts will not affect the original problem but will affect the relaxed problem by increasing the chance of finding a solution. This method can be applied to optimization problems by iteratively solving the relaxation problem with cuts as additional constraints until the solution at the current relaxation problem equals the current upper bound or incumbent. Here, the value of the incumbent is the optimal solution to the problem (Gomory, 1958).

1.3.1.4 Branch-and-cut

The branch-and-cut (B&C) algorithm incorporates the cutting plane method in the B&B algorithm. The cutting plane can be applied at the root node or at every node in the search tree. This will produce a smaller sized tree (Mitchell J. E., 2011), (Williams, 2006). The B&C adds cutting planes to a tight relaxed subproblem by deleting a set of solutions for the relaxed subproblem. The deleted solutions are not feasible to the unrelaxed subproblem (Hoffman & Padberg, 1991).

In particular, the B&C algorithm combines the advantages of the B&B and cutting plane methods. The enumeration benefits from the cutting plane, where the lower bound

obtained from the enumeration tree is better than the bound obtained from the B&B tree (Naddef & Rinaldi, 2002). Conversely, the cutting plane benefits from the enumeration, where the separation algorithm can be more active when it is used with branching (Naddef & Rinaldi, 2002). In addition, the B&C reduces the size of the search tree which helps to increase the size of the solvable instances. However, it also increases the time at each node in the search tree (Hoffman & Padberg, 1991).

1.3.1.5 Other Approaches

There are also other exact methods in the literature to solve the TSP such as:

- Cut-and-solve: Cut-and-solve uses a search path instead of the search tree. At each node in the search path, it solves two easy subproblems which are a relaxed problem and a spare problem. The advantages of this are that the iterative search strategy will not choose the wrong branch since it has only one branch, and the memory requirements are minimal (Climer & Zhang, 2006).
- Column Generation: This technique adds extra variables to the problem to avoid an excessive number of constraints (Williams, 2006). It is also considered as a dual of the cutting plane approach (Raidl & Puchinger, 2008).
- Dynamic Programming: This is an enumerative method with a divide-and-conquer-strategy with four elements: stages, states, decisions and policies (Dowsland, 2014) . The problem is handled in smaller parts in a sequential way so that small subproblems are solved first before their solutions are stored for future reference. Likewise, larger subproblems are solved by a recursion formula from the smaller ones. In other words, the next stage is calculated from the previous stage in a bottom-up manner.

1.3.2 Heuristic Algorithms

In practice, not all problems can be solved by exact algorithms due to many reasons such as a problem with a large number of constraints or huge search space. These types of problem will have an enormous feasible solution space and hence the optimal solution will be difficult to find (Michalewicz & Fogel, 2004). For these reasons, other solution methods are needed such as heuristics or metaheuristics.

Heuristic algorithms can be described as a trial and error approach when finding an optimal solution is impractical. Though there is no proven guarantee of the solution quality, some heuristic algorithms perform very well in practice and produce high-quality solutions in reasonable time. Theoretically, it is a procedure that produces a good feasible solution but not necessarily optimal (Hillier & Lieberman, 2010). This technique produces an approximate solution without guarantee of optimality. Heuristics such as local search can find near-optimal solutions within reasonable running times. It begins with an initial solution searching in its neighbourhood to find a solution that is better than the current one until no better solution is found.

The performance of an approximate algorithm can be measured by the running time and the quality of the solution (Aarts & Lenstra, 2003). The running time is given by the CPU time while the solution quality is measured by calculating the ratio between the value of the final solution obtained by the heuristic and the optimal solution obtained from the literature (Aarts & Lenstra, 2003). Unfortunately, each heuristic is designed for a specific type of problem and treats only that problem, efficiently (Michalewicz & Fogel, 2004).

Rego and Clover in (Rego & Glover, 2007) classified heuristics for the TSP into two classes, *tour construction procedures* and *tour improvement procedures.* The tour construction procedure builds an initial solution while the tour improvement procedure starts from an initial solution and seeks a better one by iteratively searching the neighbourhood (Rego & Glover, 2007). In accordance to (Rego & Glover, 2007), the heuristics in this thesis are divided into two groups which are constructive heuristics and improvement heuristics.

1.3.2.1 Constructive Heuristics

These heuristics focus on constructing a feasible solution with the main interest on the cost and not an improvement (Silberholz & Golden, 2010). The tour construction heuristic starts with an empty solution and repeatedly, tries to extend the current solution until a complete solution is obtained. Generally, tour construction heuristics can be split into three phases which are initialization, selection and insertion. The initialization phase determines the choice of the initial sub-cycle or starting point while the selection phase specifies a criterion of choosing the next nodes to be added to the current solution. The insertion phase decides the position of the new selected nodes into the current solution (Cordeau J.-F. , Laporte,

Savelsbergh, & Vigo, 2007) . The most commonly use constructive heuristics are nearest-neighbour and insertion heuristics.

1.3.2.1.1 Nearest-neighbour

This heuristic is obtained by applying the greedy approach to the TSP provided that the tour being constructed grows in a connected way. This procedure starts from a randomly chosen initial city, finds the closest unvisited city to the current city and inserts that city at the end of the current partial solution (Rader, 2010). The same procedure is recursively applied until all vertices have been included in the tour and no subtours exist. Typically, the same process is repeated with each city selected as the initial one and the best among the n tours generated in this process is selected as the output of this algorithm. The node selection procedure of this algorithm is illustrated in Figure 1.2.

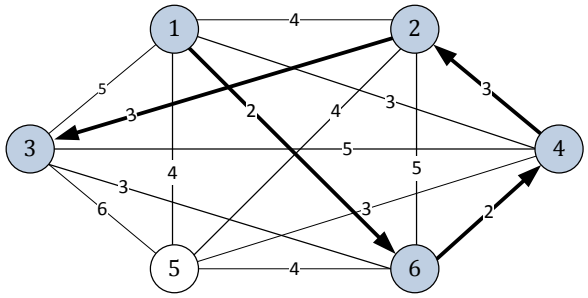


Figure 1.2: Illustration of the nearest-neighbour selection

Suppose that the current partial sequence of a six-city TSP is $1 \rightarrow 6 \rightarrow 4 \rightarrow 2$; from city 2, the distances to other unvisited cities are

$$d_{23} = 3 \quad d_{25} = 4$$

Since city 3 is the closest to city 2, city 3 would be added to the sequence, yielding

$$1 \rightarrow 6 \rightarrow 4 \rightarrow 2 \rightarrow 3.$$

1.3.2.1.2 Insertion Heuristic

The insertion heuristic is based on the insertion of a node in a particular tour. There are several variations depending on rules and criteria for the insertion phase such as cheapest-insertion and farthest-insertion.

1.3.2.1.2.1 Cheapest-insertion Heuristics

The cheapest-insertion heuristic maintains a subtour until it becomes a tour. In each step, it simultaneously determines which unvisited node should be added next and wherein the subtour it should be inserted to achieve the smallest increase in the subtour length. Given a subtour S, it finds an arc (i,j) and a city r not in S such that the index $c_{ir} + c_{rj} - c_{ij}$ is minimal and then inserts r between i and j (Rader, 2010). The selection process is repeated until a tour is obtained. Further, this procedure can also be repeated with each city as the initial city and the best of the tours obtained taken as the output. The cheapest-insertion method is illustrated in Figure 1.3.

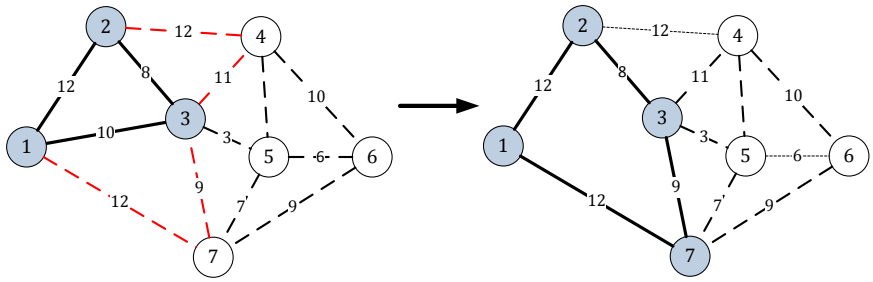


Figure 1.3: Illustration of the cheapest-insertion heuristic

From subtour $(1,2,3)$, two possible nodes to be inserted are node 4 and 7.

If node 4 is inserted, the tour cost increase is

$$c_{24} + c_{43} - c_{23} = 12 + 11 - 8 = 15$$

If node 7 is inserted, the tour cost increase is

$$c_{17} + c_{73} - c_{13} = 12 + 9 - 10 = 11$$

So, node 7 is to be inserted, which leads to a new loop $(1,2,3,7)$.

1.3.2.1.2.2 Farthest-insertion Heuristics

This heuristic begins with a tour that visits the two cities which are farthest apart. The next city to be inserted is the farthest node from any node in the current tour in such a way that the increase in the subtour length is minimized (Golden, Bodin, Doyle, & Jr, 1980). The farthest-insertion method is illustrated in Figure 1.4.

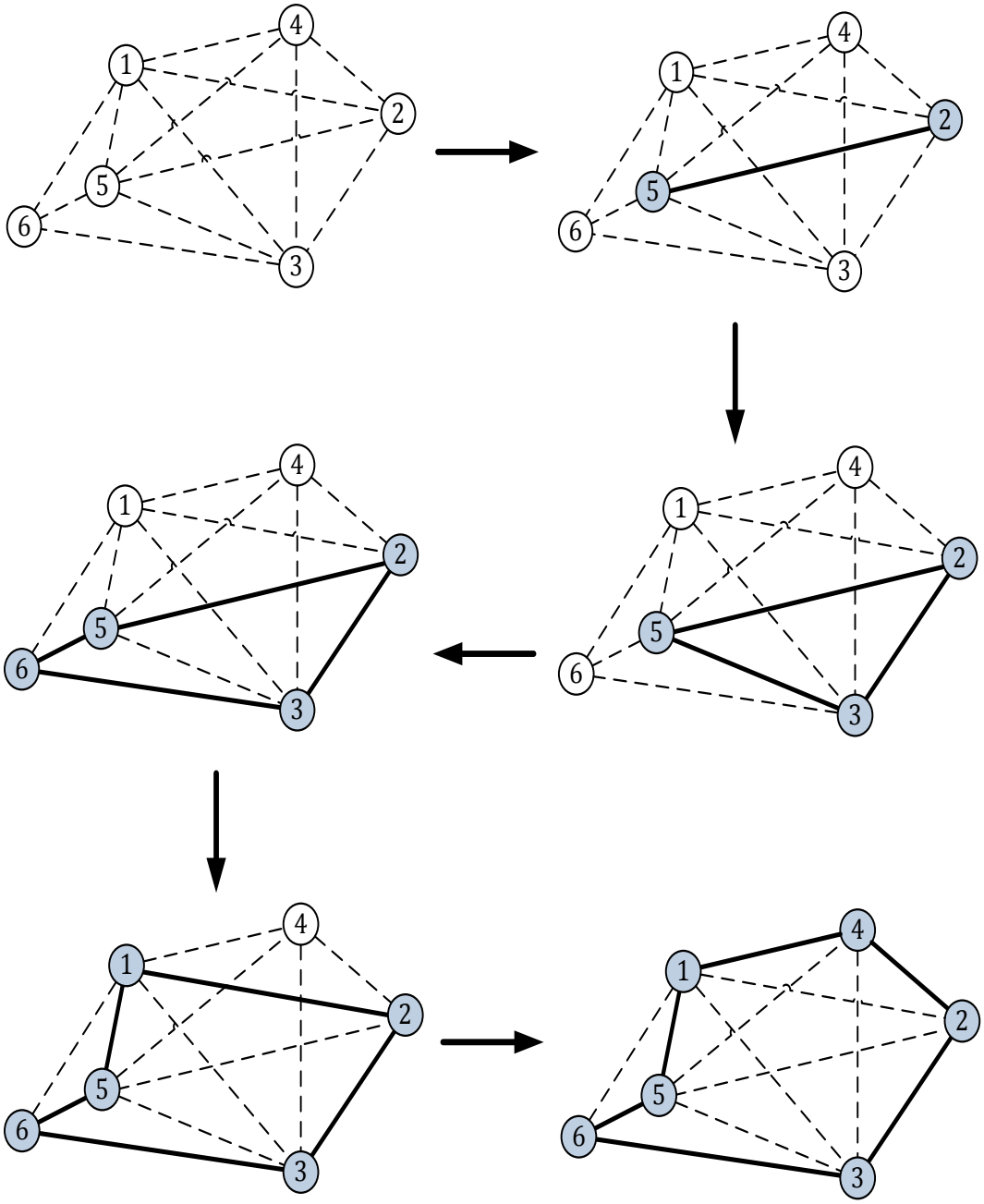


Figure 1.4: Illustration of the farthest-insertion heuristic

1.3.2.2 Improvement Heuristics

In contrast to tour construction algorithms, improvement heuristics are often used to improve initial solutions generated by other heuristics. Starting from an initial solution, an improvement heuristic applies simple modifications such as exchanges of edges to obtain new solutions of possibly better cost. If an improving solution is found, it then becomes the current solution and the process iterates. Otherwise a local minimum has been identified. The most common ways to improve an initial tour generated by construction heuristics for TSP are 2-opt and 3-opt local searches.

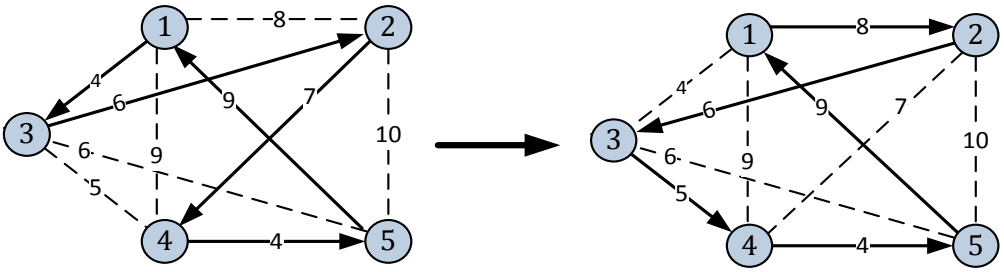
1.3.2.2.1 2-opt

The 2-opt is a simple local search algorithm that works by doing small changes on a tour and then checking if the solution quality improves. The 2-opt algorithm removes two edges from the tour, creating two new subtours, then reconnecting them in a new different way so that it forms a correct tour, only if the sum of the length of the newly arranged edges is less than the sum of the length of the deleted edges (Reinelt, 1994). This is frequently referred to as a 2-opt move and this process is repeated until no further improvement can be obtained. The 2-opt move is illustrated in Figure 1.5.

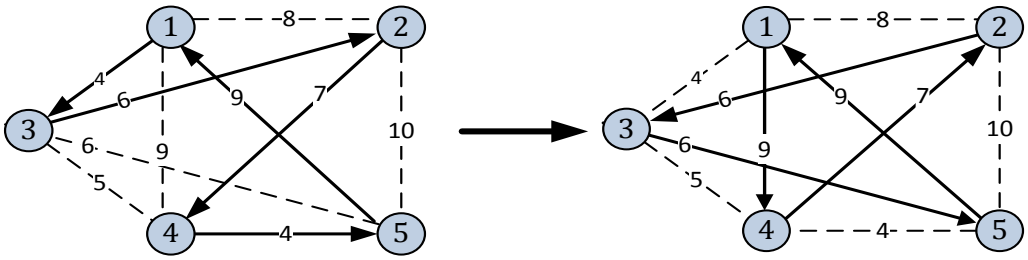
Consider a tour $1-3-2-4-5-1$ with the length of 30 units as shown in Figure 1.5. , possible edges to be changed are:

- Iteration 1 : Edges $(1,3),(2,4)$ with $(1,2),(3,4)$
- Iteration 2 : Edges $(1,3),(4,5)$ with $(1,4),(3,5)$
- Iteration 3 : Edges $(3,2),(4,5)$ with $(3,4),(2,5)$
- Iteration 4 : Edges $(3,2),(5,1)$ with $(3,5),(2,1)$

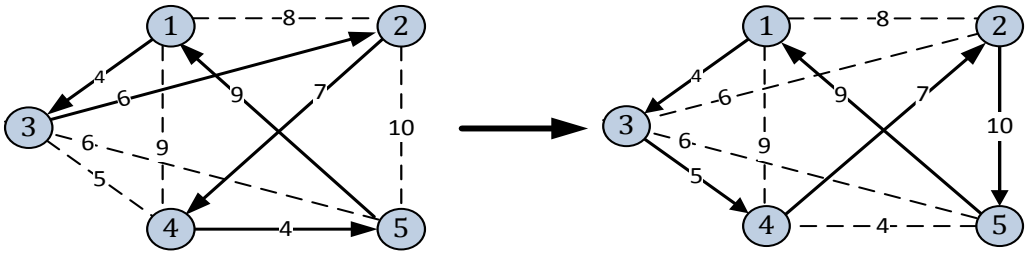
In iterations 1, 2 and 3, 2-opt is not preserved since the resultant tours have longer tours than the tour before the swap. However, replacing edges $(3,2),(5,1)$ with $(3,5),(2,1)$ in iteration 4 decreases the tour length by 1 unit thus this 2-opt move is accepted.



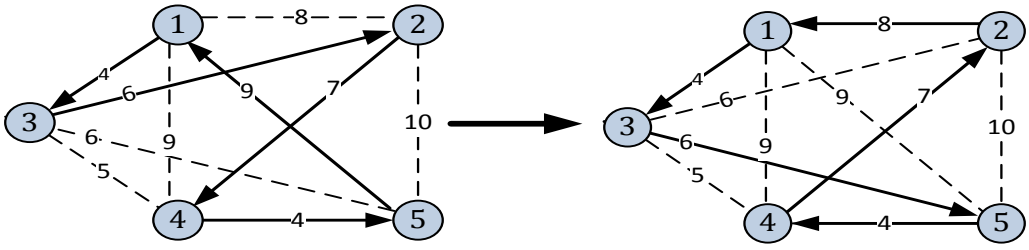
Iteration 1: swap (1,3), (2,4) with (1,2), (3,4) ; $l_{after} = 32 > l_{before} = 30$



Iteration 2: swap (1,3), (4,5) with (1,4), (3,5) ; $l_{after} = 37 > l_{before} = 30$



Iteration 3: swap (3,2), (4,5) with (3,4), (2,5) ; $l_{after} = 35 > l_{before} = 30$



Iteration 4: swap (3,2), (5,1) with (3,5), (2,1) ; $l_{after} = 29 < l_{before} = 30$ (new tour)

Figure 1.5: Illustration of the 2-opt move

1.3.2.2.2 3-opt

The 3-opt algorithm works analogously to the 2-opt heuristic, but instead of removing two edges, the exchange removes three edges from the current solution to replace them with three new edges not previously included in the current solution (Rego & Glover, 2007).

A 3-opt move can be seen as two or three 2-opt moves. So, if a tour is 3-optimal, it is also 2-optimal (Helsgaun, 2000). A 3-opt exchange provides better solutions than 2-opt exchange, but it is significantly slower in CPU time. The 3-opt move is illustrated as in Figure 1.6.

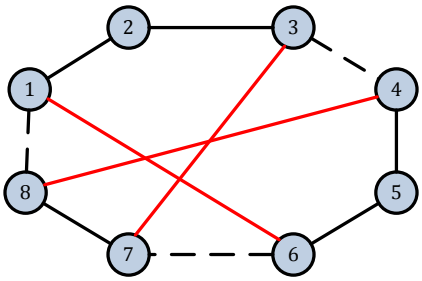


Figure 1.6: Illustration of the 3-opt interchange

1.3.3 Metaheuristics

Metaheuristics are considered a powerful approach applied to solving difficult combinatorial optimization problems and can achieve good results (Gendreau & Potvin, 2005). It consists of heuristics that are based on some metaheuristic rules.

The motivation behind the metaheuristics approach is to explore the search space in an effective and efficient way (Blum & Roli, 2008). Examples of metaheuristics are simulated annealing (Nikolaev & Jacobson, 2010), tabu search (Glover & Laguna, 1997), genetic algorithms (Davis, 1991) and ant colony optimization (Dorigo & Stutzle, 2010), etc.

The formal definition of metaheuristics can be described as an iterative master process that guides and modifies the operations of a subordinate heuristic by combining intelligently different concepts for exploring and exploiting the search space to produce near-optimal solutions. By this combination, metaheuristics aim to improve the quality of solutions compared with classical heuristics but with better computing time (Laporte, Gendreau,

Potvin, & Semet, 2000). A high or low-level procedure or a simple local search, or a construction method are examples of the subordinate heuristics.

Two basic classes of metaheuristics are local search metaheuristics and evolutionary algorithms. The local search approach finds good solutions by iteratively making changes to a current solution. In each step of the search, the current solution is replaced by another solution found in the neighbourhood, normally the best solution of that neighbourhood. However, local search metaheuristics easily get trapped in local optima and do not guarantee global optimum solutions. Thus, many metaheuristics were proposed to improve local search heuristics in order to find better solutions including simulated annealing (Kirkpatrick, Gelatt, & Vecchi, 1983), tabu search (Fred Glover, 1986) and ACO (Dorigo, Maniezzo, & Colorni, 1991).

In general, metaheuristics can be compared based on the following properties which would guarantee both their practical and theoretical aspect (Hansen, Mladenovic, Brimberg, & Moreno, 2010):

1. Simplicity: the metaheuristic should be founded on a simple and clear principle which is applicable for a wide variety of problems.
2. Precision: the steps of metaheuristics should be expressed in precise mathematical terms.
3. Coherence: all the steps of metaheuristics for a specific problem should logically follow the metaheuristic's principle.
4. Effectiveness: the metaheuristic for a specific problem should find a good or an approximate solution in a reasonable computational time.
5. Efficiency: the metaheuristics for a specific problem should find optimal or near-optimal solutions for most realistic instances.
6. Robustness: the metaheuristics performance should be consistent for different instances not only for specific instances.
7. User-friendliness: the metaheuristics should be easy to understand and easy to implement, which best implies that the metaheuristics have as few parameters as possible and ideally none.

8. Generality: the metaheuristic should lead to good results for several types of problems.
9. Interactivity: the metaheuristic should enable the user to improve the resolution process.
10. Multiplicity: the metaheuristic should be able to present some near-optimal solutions.

Metaheuristic algorithms can be classified into many categories such as nature-inspired and non-nature-inspired, population-based and single-solution based, single neighbourhood and various neighbourhood structure and, memory usage and memory-less methods (Birattari, Paquete, Stutzle, & Varrentrapp, 2001). The most common classification of metaheuristics is single solution-based and population-based metaheuristics (BoussaiD, Lepagnot, & Siarry, 2013) . In this thesis, metaheuristics are classified based on the number of solutions used by a metaheuristic at any time as follows (Sorensen & Glover, 2016):

1. Local Search Based Metaheuristics: These types of metaheuristics iteratively make small changes to a single solution (Sorensen & Glover, 2016). For example, simulated annealing and tabu search.
2. Population-based: These algorithms iteratively combine solutions into new ones (Sorensen & Glover, 2016). Such algorithms are genetic algorithms and ant colony optimization.
3. Hybrid Metaheuristics: The combination of metaheuristics and the techniques of optimization. (Sorensen & Glover, 2016).

1.3.3.1 Local Search Based Metaheuristics

1.3.3.1.1 Simulated Annealing

Simulated Annealing (SA) is inspired by the analogy to a physical annealing process. It provides a means to escape the local optimum by accepting moves that are not necessarily better than the current objective function value in order to find the global optimum (Nikolaev & Jacobson, 2010). In practice, there are four components to any SA algorithm for combinatorial search which are a brief problem representation, a neighbourhood function, a transition model and a cooling schedule (Aarts, Korst, & Laarhoven, 1997).

Let z denote the objective function value for the current trial solution, z_1 is the objective function value for the current candidate to be the next trial solution, and ζ is a parameter used to determine acceptance of the candidate to be the next trial solution if the candidate value is not better than the value of the current solution (Hillier & Lieberman, 2010). The SA algorithm starts with an initial solution in the feasible region and uses move selection rules to move to the next trial solution. Assuming the objective is maximization of the objective function, the move selection rule is:

- accept movement to the next trial solution if its value is better than the current solution;
- otherwise, if no better solution is found in the neighbourhood of the current solution
  - move to the immediate neighbour only if a random number ξ generated from a uniform distribution (0,1) is less than the probability of acceptance

$$\text{Prob}\{\text{acceptance}\} = e^{x} \quad \text{where} \quad x = \frac{z_i - z}{\zeta}$$

  - otherwise, keep the current solution.

During the search, the value of ζ gradually decreases and each value of ζ can be used with a determined number of iterations. When the desired number of iterations has been performed at the smallest value of ζ in the temperature schedule, the process is terminated and the best trial solution found at any iteration is accepted as the final solution (Hillier & Lieberman, 2010).

The SA is different from local search in three aspects (Michalewicz & Fogel, 2004):

1. How the procedures stop: The SA is executed until the stopping conditions are satisfied while the local search is executed until no improvement is found.
2. The way the SA moves: it not only moves to better solutions but also accepts solutions based on the current temperature ζ.
3. The value of the parameter ζ in the SA is updated periodically during the search: this parameter value influences the outcome of the SA.

The probabilistic rule used by the SA to moves between the candidate solutions accepts the neighbourhood as the new current position if the solution found is better than the current one.

Otherwise, either accept this new solution anyway or continues the search again in the same neighbourhood for another solution (Michalewicz & Fogel, 2004).

In general, the stopping conditions of the SA can be the maximum number of iterations, or the maximum number of iterations when none of the immediate neighbours of the current trial solution is accepted (Hillier & Lieberman, 2010).

The SA approach has been applied to the TSP by a number of researchers including Bonomi and Lutton (Bonomi & Lutton, 1984), Rossier, Troyon and Liebling (Rossier, Troyon, & Liebling, 1986), Golden and Skiscim (Golden & Skiscim, 1986) and Nahar, Sahni and Shragowitz (Nahar, Sahni, & Shragowitz, 1986), with a different degree of success. However, the SA has proved to be less efficient to solving the TSP than heuristic methods that have more knowledge about the problem (Coppin, 2004).

1.3.3.1.2 Tabu Search

Tabu Search (TS) is considered an extension of a classical local search with the addition of short-term memory (Gendreau & Potvin, 2010). The first proposition for the TS method was made in 1986 by Glover (Fred Glover, 1986) and it has been applied to a wide-ranging number of applications such as vehicle routing (Cordeau & Laporte, 2005), machine scheduling (Taillard, 1990), the maximum clique problem (Gendreau, Soriano, & Salvail, 1993) and the quadratic assignment problem (Skorin-Kapov, 1990) .

The basic elements of TS are the search space, the neighbourhood structure, the short-term tabu lists, and aspiration criteria. The search space is the space of all possible solutions that can be considered during the search (Gendreau & Potvin, 2010) . At each iteration of TS, a local transformation is applied to the current solution defining a set of neighbouring solutions in the search area (Gendreau & Potvin, 2010). Aspiration criteria state that the tabus can be ignored if there is no chance of cycling. In other words, it accepts movements to tabu moves if it produces solutions that are better than the current solution (Gendreau & Potvin, 2010).

The TS uses a local search procedure to find a local optimum and then moves to any point in the neighbourhood. If a better solution is found in the neighbourhood of the current trial solution, the local search procedure is applied again to find a new local optimum (Hillier & Lieberman, 2010). While classical local search methods stop when they encounter a local

optimum with regard to the modifications they allow, the TS continues moving to the best non-improving solution it can find.

To prevent a repetition of the same local optimum, the tabu list will record each move in tabu moves, so that a tabu list is updated during the running of the algorithm (Glover & Laguna, 1997). For the best use of memory in the tabu search, the tabu list should be used efficiently. There are three basic principles to managing the tabu list (Hertz, Taillard, & Werra, 1997):

- Size of the tabu list : A list that is too short may not prevent cycling, whilst a long list may expand the search and increase the number of visited solutions. Though, it is often difficult to determine the size of a tabu list that prevents cycling and does not excessively restrict the search for all instances of a given size. Hence, an effective way of avoiding this is to vary the size of the tabu list (Hertz, Taillard, & Werra, 1997) .
- Intensification of the search : Defined as exploring a portion of the promising neighbourhood more thoroughly, implying that the portion of the neighbourhood contains very good solutions (Hillier & Lieberman, 2010). In order to intensify the search, the size of the tabu list should be decreased for a small number of iterations (Hertz, Taillard, & Werra, 1997).
- Diversification : Indicates searching into previously unexplored areas of the neighbourhood (Hillier & Lieberman, 2010). A common way to diversify the search is to randomly execute several random restarts (Hertz, Taillard, & Werra, 1997).

Different stopping criteria such as a fixed number of iterations, a fixed number of consecutive iterations if there is no improvement in the best objective function value , or a fixed amount of CPU time can be used for the TS termination criteria (Hillier & Lieberman, 2010). For more information on TS and its application to TSP see (Knox, 1994) , (Gendreau, 2003), (Gendreau & Potvin, 2010) and (Basu, 2012).

1.3.3.1.3 Variable Neighbourhood Search

The variable neighbourhood search (VNS) was suggested by Mladenovic and Hansen in 1997 (Mladenovic & Hansen, 1997). Its basic idea is to escape from local optima trap by changing

the neighbourhood structure. It explores distant neighbourhoods of the current incumbent solution, and moves from there to a new one if and only if the improvement was made. For more information on VNS and its application to TSP see (Hansen & Mladenovic, 2001) and (Hansen, Mladenovic, Brimberg, & Moreno, 2010).

1.3.3.1.4 Reactive Bone Algorithm

The reactive bone algorithm (RBA) is a population-based method proposed by (Darani, Dolatnejad, & Yousefikhoshbakht, 2015) to solve the TSP. It is a modification of a Bone-Route method which was first introduced for solving the vehicle routing problem by Tarantilis and Kiranoudis in 2002 (Tarantilis & Kiranoudis, 2002). The Bone-Route algorithm constructs a new solution out of sequences of nodes or bones of the previous solutions. The construction of the new solution is based on the Adaptive Memory concept introduced in (Rochat & Taillard, 1995) which describes a pool of good solutions that are dynamically updated throughout the solution search process. Some components of these solutions are extracted from the pool periodically and combined to construct a new solution. The extraction criteria of the Bone-Route algorithm are:

1. Bone length – the number of nodes that must compose a bone.
2. Bone frequency – the number of stored routes in the pool that must include a bone in their routes.

However, the RBA has made two main modifications to the Bone-Route approach which are:

1. Value of the bone size systematically changes during run time.
2. Bone frequency maximum- the maximum number of stored routes in the pool that must include a bone.

Since the bone size and the bone frequency express the degree of similarity among the new solution and the previously stored solutions in the pool, the RBA modification ensures that the new solution is more similar to other solutions in the pool, whenever their value is high.

1.3.3.2 Population-Based Metaheuristics

1.3.3.2.1 Genetic Algorithms

Genetic algorithms (GAs) were first used by John Holland in 1975 (Holland, 1992). Holland's GA was inspired by the natural evolution cycle such as restricted alphabet genotype, pairwise parent selection, mating and mutation. In the GAs, bit strings play the role of the chromosome, with individual bit sets playing the role of genes. In other words, the genotype-phenotype mapping in GAs relies on a bit partitioning into bit sets. To recombine strings, Holland used the idea of genetic *crossover* and *mutation* (Reeves C. R., 1997) . Crossover is defined as the substitution of some genes from one parent with parallel genes in the other parent (Reeves C. R., 1997). Mutation means random changes in the specific genes in the phenotype (Mitchell M. , 1998). There are two strategies to generate offspring; the first one is to use crossover and mutation while the second strategy uses either crossover or mutation (Reeves C. , 2003).

The GAs are usually started with a population of feasible trial solutions. A random process is used to select some of the feasible solutions from the population to become parents and then randomly pair up the best parents to produce new feasible solutions (*children*) (Hillier & Lieberman, 2010). If an infeasible solution (*miscarriage*) is obtained, this process is repeated until a feasible solution is found. The stopping conditions could be the number of iterations or CPU time. The outline of a typical GA is given as follows (Mitchell M. , 1998):

1. Randomly initialize a population;
2. Calculate the fitness function of each individual in the population;
3. Select a pair of parent chromosomes based on the fitness value to create a new generation by applying mutation, or crossover;
4. Substitute the current population with the new population.
5. If the stopping condition is satisfied, stop. Otherwise, go to step 2.

In most cases, the performance of the GA is dependent on the genetic operators used. The four main operators for the GAs are given below (Affenzeller, Winkler, Wagner, & Beham, 2009):

- Parent Selection: selecting an individual from the population to be a parent based on its fitness. The selection process can be done through various methods such as proportional selection, linear-rank selection or tournament selection.
- Crossover: The process of joining two individuals (parents) to produce two new individuals (offspring). There are a number of crossover techniques including single point crossover and multiple point crossovers. Single point crossover divides the chromosome of each parent into two parts (head and tail) by using a random cut. The tail of the first parent connects with the head of the second parent and the tail of the second parent connects with the head of the first parent to produce two new offspring.
- Mutation: It allows an undirected change to the area of the search by randomly replacing one value of a gene (bit) in a specific position, but it happens randomly with very low frequency.
- Replacement: This operator decides which newly generated individual will be chosen to be a member of the new generation. There are many strategies for the replacement mechanism such as generational replacement, elitism and tournament replacement.

Two main concerns to be considered when implementing GAs are the size of the population and the technique used to choose the individuals (Reeves C. , 2003). The initial population is usually assumed to be random while the size of the population is normally chosen based on the required level of efficiency and effectiveness (Reeves C. , 2003). Since principally the GAs could run forever, a stopping condition is needed in practice. Common stopping conditions could be a number of fitness evaluations or computer run time, etc. For more information on GAs and its application see (Mitchell M. , 1998), (Taiwo, Mayowa, & Ruka, 2013) and (Sastry, Goldberg, & Kendall, 2014).

1.3.3.2.2 Particle Swarm Optimization

The particle swarm optimization (PSO) is an evolutionary algorithm inspired by a social behavior such as bird flocking and fish schooling introduced by Eberhart and Kennedy in 1995 (Kennedy & Eberhart, 1995). In PSO, each intelligent individual searching for an optimal position is called a particle. Each particle represents a candidate solution that can be evaluated by a preset evaluation function. Flying in a multidimensional search space, a particle changes its velocity dynamically based on its own flying experience and the flying experience of its colleagues.

The PSO algorithm begins with randomly initialising a swarm of particles, then iteratively adjusts the flying trajectory of each particle toward its personal best position called local optimum and toward the best particle of swarm or global optimum, and finally achieves an optimal solution.

The initial position vectors $x_i(0)$ and velocity vectors $v_i(0)$ are randomly selected over the search space. Then these particles evolve all through space according to two essential reasoning capabilities: a memory of their own best position and knowledge of the global or their neighbourhood's best. The evolution for each particle i is given by

$$x_i(t+1) = x_i(t) + v_i(t+1) \tag{1.26}$$

where $x_i(t)$ and $v_i(t)$ are the position and the velocity of particle i at time t , respectively.

The PSO algorithm is influenced by a number of control parameters such as swarm size, neighbourhood size, inertia weight, acceleration coefficients, and number of iterations. Hence, implementing a PSO algorithm requires a careful selection of these parameters.

1.3.3.2.3 Bat Algorithm

The bat algorithm (BA) is an algorithm based on the echolocation behaviour of bats (Yang, 2010). The echolocation is the use of sound waves and echoes to determine the location of objects in space. Bats use echolocation to navigate, detect prey and avoid obstacles in the dark by emitting a loud sound pulse and listen for the echo that bounces back from the surrounding objects. The sound pulses and the signal bandwith varies in properties and can be correlated with their hunting strategies, depending on the species.

The BA is formulated by idealizing the echolocation behaviour of bats as follows:

1. Bats use echolocation to sense distance and differentiate between food, prey and background barriers.
2. Bats fly randomly with velocity v_i at position x_i with a fixed frequency f_{min}, varying wavelength λ and loudness A_0 to search for prey. They can automatically adjust the wavelength (or frequency) of their emitted pulses and adjust the rate of pulse emission $r \in [0,1]$ depending on the proximity of their target;

3. Although the loudness can vary in many ways, they assume that the loudness varies from a large (positive) A_0 to a minimum constant value A_{min}.

1.3.3.2.4 Other Population-Based Metaheuristics Approaches

- Artificial Bee Colony

  Artificial Bee Colony (ABC) was introduced by Dervis Karaboga in 2005, motivated by the intelligent behaviour of honey bees. In the ABC system, the colony of artificial bees consists of three different groups namely employed, onlooker and scout (Karaboga, 2005). The employee bees work on the collection of food to the hive at a specific food source. The onlooker bees patrol the employees to verify when a specific food source is not worth anymore while the scout bees are responsible for looking for new food sources locations. In this algorithm, a food source denotes a possible solution to the optimization problem, and the quality of the food source is defined by the cost of the objective function on that position. Therefore, the ABC system combines local search methods, carried out by employed and onlooker bees, with global search methods, managed by onlookers and scouts, attempting to balance exploration and exploitation process.

- African Buffalo Optimization

  The African Buffalo (ABO) Optimization was proposed by Odili, Kahar and Anwar in (Odili, Kahar, & Anwar, 2015). The ABO algorithm models the three characteristic behaviours of the African buffalos that enable their search for pastures. These characteristics include extensive memory capacity, cooperative cum communicative ability and democratic nature borne out of extreme intelligence. The extensive memory capacity enables the buffalos to keep track of their routes through thousands of kilometres in the African landscape. Furthermore, the ‘waaa’ sound is an alarm call used to tell the herd to keep moving if the present location is unfavourable, lacks pasture or is dangerous. In other instances, the same ‘waaa’ sound is used to invite other buffalos to come to the aid of other animals in danger. On the other hand, the ‘maaa’ vocalizations are used to signal to the buffalo herd to stay on to exploit the present location as it holds promise of good grazing pastures and is safe. The third attribute of the buffalos is their democratic nature borne out of extreme intelligence. In

cases there are opposing calls by members of the herd, the buffalos have a unique way of doing an 'election' where the majority decision determines the next line of action. These three characteristics mark out African buffalos as one of the most organized and successful herbivores of all time (Odili, Kahar, & Anwar, 2015).

1.3.3.2.5 Ant Colony Optimization

The Ant Colony Optimization (ACO) is part of swarm intelligence and imitates the behaviour of ants during the process of moving food from the source to the colony by using shortest routes (Dorigo & Stutzle, 2004). It uses artificial ants to find solutions to combinatorial optimization problems.

The artificial ants use heuristic information and pheromone values for guiding the search process. The heuristic information that is normally available for many problems, together with a stochastic component in the ACO lead the ants towards more promising solutions. This stochastic component allows the ants to build a wide variety of different solutions and explore a larger number of solutions.

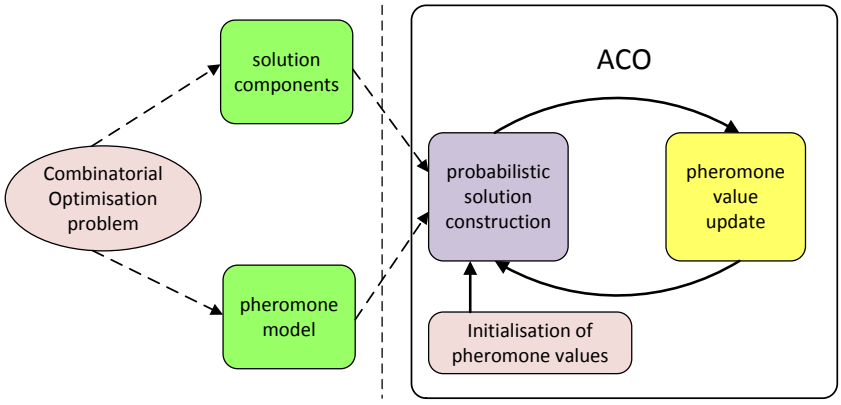


Figure 1.7: Basic principle of ACO metaheuristic (Blum C. , 2005)

The basic steps in solving an optimization problem using the metaheuristic of ACO are shown in Figure 1.7. For a given combinatorial optimization problem, a finite set C of solution components has to be derived to construct solutions to the COP. The pheromone model, which is a set of pheromone values T, has to be defined afterwards. This set of values is used to parameterize the probabilistic model. The pheromone values $\tau_i \in$ T are usually

associated with solution components. Broadly speaking, the ACO approach normally solves an optimization problem by repeating the following two steps as a loop (Blum C. , 2005):

- Candidate solutions are constructed using the pheromone model which represents a parameterized probability distribution over the solution space;
- The candidate solutions are further employed to modify the pheromone values which are deemed to bias the future sampling toward high-quality solutions.

In general, the ACO metaheuristic contains three main procedures (Dorigo & Stutzle, 2004) which are *Construct Solutions*, *Update Pheromones* and *Daemon Actions*.

Construct Solutions is the procedure where an ant constructively builds a solution of the considered problem by moving through neighbour nodes of the problem construction's graph. They move by applying a stochastic local decision policy that makes use of pheromone trails and heuristic information. After the ants have completed their solutions, they evaluate the quality of their solution, which will be used in the *Update Pheromone* procedure to decide how much pheromone to deposit.

*Update Pheromon*e is the procedure by which the pheromone trail values are updated based on the latest experience of the colony. The update phases consist of decreasing and increasing the pheromone intensity on the trails. Pheromone evaporation is applied to decrease pheromone values to encourage exploration and prevents stagnation whilst pheromone deposit is adapted to increase the pheromone values that belong to good solutions the ants have generated. The amount of pheromone deposited strongly depends on the quality of the particular solution that each path found. Hence, the intensity of pheromone will be biased towards the best solutions found so far.

On the other hand, *Daemon Actions* is an optional procedure where an additional enhancement to the original solution or a centralized action is implemented that cannot be done by a single ant. For example, the use of local search methods or to lay extra pheromone on the best solution found so far. The pseudo code of the standard ACO algorithm is shown in Figure 1.8.

For more information on the ACO and some of its applications see (Dorigo & Caro, 1999), (Cordon, Herrera, & Stutzle, 2002), (Dorigo & Stutzle, 2004), (Dorigo, Birattari, & Stutzle, 2006) and (López-Ibáñez, Stützle, & Dorigo, 2016).

Algorithm 1.1: Ant Colony Optimization

```
Set parameters, initialize pheromone trials
    while (termination condition not met) do
        Solution Construction Phase
        Pheromone Update Phase
        Daemon Actions Phase //optional
    end-while
return best solution
```

Figure 1.8: Pseudo-code of the ACO metaheuristic

1.3.3.3 Hybrid Metaheuristics

A hybrid metaheuristic is defined as a combination of a metaheuristic with other metaheuristics, or with parts of other metaheuristics, or with other optimization techniques (Blum & Roli, 2008). The hybrid optimizers are effective in reducing the CPU time and improving the quality of the solution (Riadl & Puchinger, 2008). The hybrid metaheuristics take advantage of the strengths of each of the individual algorithms and synergy to produce a more effective hybrid system (Raidl, Puchinger, & Blum, 2010).

Different classifications for the hybrid metaheuristics can be found in the literature. Blum & Roli (Blum & Roli, 2008) classified hybrid metaheuristic approaches as either collaborative combinations or integrative combinations. Collaborative combinations exchange the information between optimization techniques that executed sequentially, intertwined or in parallel (Puchinger & Raidl, 2005). Integrative combinations mean that one technique is a subordinate of another technique (Puchinger & Raidl, 2005).

An example of a hybrid metaheuristic is the hybridization of metaheuristic with B&B. This integrative combination can be done in two different ways (Blum & Roli, 2008):

- the use of B&B within a metaheuristic such as ACO to improve the efficiency of the metaheuristic search process.
- the use of the metaheuristic within B&B to reduce the CPU time and minimize the search tree in B&B.

Alternatively,Raidl et al. in (Raidl, Puchinger, & Blum, 2010) classify a hybrid metaheuristic according to the following criteria:

1. Hybridized algorithms: such as a mixture of parts of some metaheuristic strategies, or combination of metaheuristics with general techniques from operational research and artificial intelligence.
2. Level of hybridization: the strength of the hybridization; high-level combinations (no direct relationship of the internal workings of the algorithms) and low-level combinations (strongly dependent on each other).
3. Order of execution: in batch execution, algorithms are performed in sequential order while in an intertwined or parallel way, information is exchanged in a bidirectional way.
4. Control strategy: integrative (one is part of the other algorithm); or collaborative (each one of them is not part of the other, but they swap information).

For more reviews on the hybrid metaheuristics see (Cotta-Porras, 1998), (Dumitrescu & Stutzle, 2003), (Raidl, 2006) , (Blum, Roli, & Sampels, 2008) , (Blum, Puchinger, Raidl, & Roli, 2010) and (Ting, Yang, Cheng, & Huang, 2015).

1.3.4 Other Approaches

There are also other metaheuristic approaches in the literature use to solve the TSP such as:

- Artificial Neural Network :

  An artificial neural network or ANN is a computing system whose central theme is borrowed from the analogy of biological neural networks (Mehrotra, Mohan, & Ranka, 2000). ANN is also reffered to as artificial neural systems, neural nets or parallel distributed processing systems. The essential element of this paradigm is the novel structure of the information processing system. In a neural network, each node performs some simple computations and each connection conveys a signal from one to another, labeled by a number called the *connection strength* or *weight* indicating the extent to whch a signal is amplified or diminished by a connection. One of the most significant attributes of a neural network is its ability to learn by interacting with its environmnet or with information sources (Hassoun, 1995). An ANN is configured for a specific application, such as pattern recognition (Basu, Bhattacharyya, & Kim, 2010) or

data classification through a learning process (Gu, Liu, Li, & Yuan-Yuan Huang, 2008).

- Artificial Immune System :

  The Artificial Immune System (AIS) is inspired by the natural immune system for solving computational problems. Three immunological principles primarily used in AIS are the clonal selection principle, the negative selection mechanism and the immune network theory (Malim & Halim, 2012).

1.4 Survey on the Traveling Salesman Problems Solution Methods

According to (Alejandro Rodríguez; Rubén Ruiz, 2010), the methods designed for symmetric TSP instances may not be adapted easily to solve asymmetric TSP. This might be the reasons why the literature is rich for symmetric TSPs as well as for asymmetric TSPs but slightly poor for both symmetric and asymmetric TSP. Nevertheless, this section provides a brief overview of related works for comparison with the proposed algorithm according to the type of TSP considered. Some papers that address the solutions to the symmetric TSP are:

- Crowder and Padberg in (Crowder & Padberg, 1980) reported the optimal solutions for 10 large-scale symmetric travelling salesman problems with a size between 48 and 318 cities using a cutting-plane approach coupled with branch-and-bound. In their algorithm, variables are fixed at either zero or one to reduce the size of the problems. On average, this fixing strategy has reduced the number of variables obtained by 5% of the original number of variables. Further, this algorithm has shown impressive results when applied to large-scale zero-one linear programming problems with a number of zero-one variables between 33 and 2750 (Crowder, Johnson, & Padberg, 1983).
- A paper by Pasti and Castro (Pasti & Castro, 2006) in 2006 presented a metaheuristic approach for solving the TSP based on a neural network and artificial immune system called RABNET-TSP. This hybrid algorithm has a single-layer self-organizing neural network architecture with a learning procedure aimed at locating one network cell at each position of a city of the TSP instance to be solved. A modification to the RABNET-TSP was proposed by Masutti and Castro in (A.S.Masutti & Castro, 2009) to improve the efficacy and the computational time of the algorithm. The modified RABNET-TSP is applied to several STSP benchmark problems and the results

obtained are compared with the original RABNET-TSP and other algorithms from the literature. Despite competitive results reported, greater computational time is required to solve these problems.

- In 2011, Chen and Chien (Chen & Chien, 2011) described a hybrid of four algorithms; GA , SA , ACS and particle swarm optimization (PSO) to solve the TSP. This algorithm is called the genetic simulated annealing ant colony system with particle swarm optimization techniques or GSA-ACS-PSO. The ACS is used to generate the initial population of the genetic algorithm while the SA played the role of a mutation in the GA. The authors claimed that the GSA-ACS-PSO average solution and the percentage deviation of the average solution to the optimal solution results on 25 STSP benchmark problems are better than those existing algorithms. However, the performance of the GSA-ACS-PSO decreases as the size of instances increases, in particular for instances with more than 100 cities.

- Deng et al. (Deng, et al., 2012) . This paper introduced a novel two-stage hybrid swarm intelligence optimization algorithm or GA-PSO-ACO for solving the TSP. The GA-PSO-ACO is based on GAs, PSO and ACO, and is divided into two stages. In the first stage, the GA and PSO are used to obtain a series of sub-optimal solutions to adjust the initial pheromone value in the ACO. In the second stage, the algorithm employs the advantages of the parallel, positive feedback and high-accuracy of solution to accomplish solving the whole problem. The numerical results on 35 STSP benchmark problems showed that the solution qualities of the GA-PSO-ACO are better than the TS, GAs, PSO, ACO and PS-ACO but with longer CPU time. Besides, the performance of the GA-PSO-ACO becomes worse as the size of instances increased.

- Tuba and Jovanovic (Tuba & Jovanovic, 2013) in 2003 suggested a pheromone correction strategy (SEE) that was based on the analysis of properties of the best-found tour to ACO algorithm which only activated when the search algorithm has started to stagnate. This strategy adds a new heuristic for determining the undesirability of edges belonging to the tour and significantly decreases their pheromone values. Computational experiments on 11 TSP benchmark problems from the TSPLIB library with up to 200 cities showed that this algorithm is more efficient than the basic ACO and the particle swarm optimization.

- In 2015, Gunduz et al. proposed a new hierarchic approach known as ACO-ABC that uses patch construction and improvement heuristics to solve the TSP in (Gunduz, Kiran, & Ozceylan, 2015). The ACO is used as the path constructor and the results obtained are improved using the artificial bee colony (ABC). The computational experiments conducted on 10 STSP benchmark problems showed that the ACO-ABC produces better quality solutions with less computational time than the individual approaches of ACO and ABC. However, based on the optimal solutions stated in their paper, only 2 instances were solved to optimality out of the 10 test instances.
- Mahi et al. (Mahi, Baykan, & Kodaz, 2015) in 2015 recommended a new hybrid algorithm based on PSO, ACO and 3-opt for solving small TSP instances called PSO-ACO-3opt. The PSO is used to optimize the values of two main parameters of ACO that affect the performance of the ACO algorithm, and the 3-opt is used to escape from the local optima found by the ACO algorithm. The PSO-ACO-3opt was tested on 10 STSP benchmark problems with up to 200 cities. Although the computational results showed that the performance of the PSO-ACO-3opt is better or similar to other methods compared such as RABNET-TSP and ACO-ABC, only 50% of the test instances were solved to optimality.
- In 2016, inspired by the learning ability of the ACO algorithm, Wang et al. (Wang, Lin, Zhong, & Zhang, 2016) suggested a swarm SA (SSA) to improve the efficiency of the SA algorithm for TSP. The SSA employs a swarm of agents running SA algorithm collaboratively and stores learned knowledge in a pheromone matrix. The pheromone matrix is then used to guide the generation of candidate solutions. The comparative experiments showed that the SSA has better performance than GSA-ACS-PSO (Chen & Chien, 2011) and GA-PSO-ACO (Deng, et al., 2012). Despite the good quality solutions, the SSA shows rather poor performance for instances larger than 200 cities.
- Yousefikhoshbakht et al. (Yousefikhoshbakht, Malekzadeh, & Sedighpour, 2016) in 2016 presented an algorithm called REACSGA for solving the TSP. REACSGA is a reactive bone algorithm that employs the ACS for generating initial diversified solutions and modified GA improvement procedure for improving the initial solutions. The proposed algorithm was tested on 19 TSP benchmark problems involving 24 to 318 cities. The computational results showed that the REACSGA

yields better solutions than the genetic algorithm, ACS, GSA-ACS-PSO (Chen & Chien, 2011), PSO (Zhong, Zhang, & Chen, 2007) and bee colony optimization (BCO) (Wong, Low, & Chong, 2008). Even so, the solutions qualities of the REACSGA are decreased for instances with 100 or larger cities.

- In 2016, Mohsen (Mohsen, 2016) described a new hybridize metaheuristic algorithm termed annealing elitist ant system with mutation operator or AEAS to solve the TSP. This algorithm integrates the advantages of ACO, SA, mutation operator and local search. The SA and mutation operation were used to increase the ant's population diversity while the local search helps to exploit the current search area efficiently. The simulation results reported on 24 STSP benchmark problems showed that the AEAS outperformed GSA-ACS-PSO (Chen & Chien, 2011), SSA (Wang, Lin, Zhong, & Zhang, 2016), REACSGA (Yousefikhoshbakht, Malekzadeh, & Sedighpour, 2016) and PSO-ACO-3opt (Mahi, Baykan, & Kodaz, 2015).

The following works discusses the solutions to asymmetric TSPs:

- In 2001, Burke et. al (Burke, Cowling, & Keuthen, 2001) presented a Guided Variable Neighbourhood Search (GVNS) approach which embedded an exact algorithm into a local search heuristic in order to exhaustively search promising regions of the solution space. The GVNS was applied on TSPLIB instances with sizes ranging from 17 to 458 cities. Despite being capable of improving the results obtained by its constituent heuristic, the number of optimal solutions obtained is considered low with only 11%.
- In 2005, Brest and Zerovnik (Brest & Zerovnik, 2005) proposed a heuristic approach based on the arbitrary insertion algorithm or a relaxation of the cheapest insertion algorithm known as Randomized Arbitrary Insertion or RAI. The numerical results showed that the RAI found the optimal solutions in 85% of the test instances.
- Abdoun et al (Abdoun, Tajani, Abouchabaka, & Khatir, 2016) in 2016 introduced a new operator called Crossover Inverse Mark operator (XIM) for the ATSP in order to improve the solution obtained by GAs. The effectiveness of this Improved Genetic Algorithm (IGA) approach was evaluated using standard benchmark instances from TSPLIB with sizes up to 443 cities. Although the authors claimed that the new operator is able to obtain a better solution, the numerical results showed that the best

results yielded by the IGA approach for all the benchmark problems are far from the optimal solutions except for the br17 instance.

Moreover, several works that review the solution to both symmetric and asymmetric TSP are:

- Gambardella and Dorigo in 1996 introduced the Ant Colony System (ACS) that increases the importance of exploitation of information collected by previous ants with respect to exploration of the search space. The strategies used to achieve this are the usage of a pseudorandom proportional rule that guides the ants to choose the next city to move to and a strong elitist approach to update pheromone trails by allowing only the ant that produced the best solution to update the pheromone trails. The ACS was tested on 6 STSP instances with size ranging from 51 to 1577 cities and 5 ATSP instances with size ranging from 43 to 170 cities. Its application to both symmetric and asymmetric TSP obtained excellent results with over 99% accuracy.
- In 1997, Stutzle and Hoos (Stutzle & Hoos, 1997) suggested a variant of the ant system known as MAX-MIN Ant System (MMAS) by introducing upper and lower bounds to the values of the pheromone trails. These trail bounds alleviate the early stagnation and thus increase the exploration of tours. The computational results on several TSP instances show that the MMAS was the best performing algorithm at that time.
- Odili and Kahar (Odili & Kahar, 2016) in 2016 proposed a new metaheuristic algorithm inspired by the behaviour of African buffalos called African Buffalo Optimization (ABO). The ABO belongs to the swarm intelligence (SI) algorithms which are based on the social behaviour in animals. This algorithm aims to achieve greater exploitation and exploration of the search space and faster speed in reaching the optimal results with relatively fewer parameters. The ABO was implemented to solve 35 STSP and 6 ATSP benchmark problems with accuracy over 98%. Despite the competitive results obtained, only 5 STSP and none of the ATSP instances were solved to optimality.
- In 2016, Osaba et al. (Osaba, Yang, Diaz, Lopez-Garcia, & Carballedo, 2016) described an improved version of the basic bat algorithm (BA) known as IBA to solve both the symmetric and asymmetric TSP. BA was initially suggested by Yang (Yang, 2010) in 2010 and is based on the echolocation behaviour of microbats which

have the skill to find their prey and discriminate different kinds of insects even in complete darkness. The BA was compared to some popular techniques such as GA, evolutionary simulated annealing (Yip & Pao, 1995) and the Island-based Distributed GA (Alba & Troya, 1999) on 22 STSP instances with size ranging from 30 to 1002 cities and 15 ATSP instances with size ranging from 17 to 323 cities. Although the comparative results showed that the IBA algorithm outperformed all other algorithms in most of the instances, the performances of the IBA is quite poor for instances with size more than 150 cities for symmetric TSP and more than 60 cities for asymmetric TSP.

1.5 Overview of the Research

This research investigates the application of a new modified ACO algorithm to solve both symmetric and asymmetric TSP which utilizing a partial optimization technique and 2-opt local search. The basic algorithmic framework of this proposed algorithm is the framework of the ACO. However, in the tour construction mechanism, only parts of the solution tour is constructed using a new proposed state transition rule, aided by intelligent ants. The other remaining part of the solution tour is optimized by a solver. Unlike the basic ACO algorithm, the probabilistic decision rule for this proposed algorithm is biased on pheromone information while the initial pheromone value is calculated as the inverse function of the distance between two nodes. At each iteration, a 2-opt local search is applied to possibly improve the local solution.

The role of the intelligent ant in the tour construction mechanism is to ensure that the best solution is inherited by the next generation (iteration) preventing the best ant of the next colony having a worse value than the best ant of the current solution. This accelerates the convergence rate of the algorithm.

After the completion of the tour construction phase, the value of the pheromone is updated. Only arcs that belong to the colony-best-ant are updated, and the amount of the pheromone deposited by the ant is determined by the constant parameter α and the solution quality of the colony-best-ant. The parameter α is calculated as the square of the problem size. Likewise, the pheromone evaporation rule works only on all arcs belonging to the

colony-worst-ant that are not in the colony-best-ant. The pheromone evaporation parameter ρ is proportional to the inverse of the square of the problem size.

1.6 Outline of the Thesis

The structure of the thesis is as follows:

Chapter 1 presents three variants of TSPs which are symmetric, asymmetric and multiple TSP. Detailed formulations of the TSP are explained focusing on different types of subtour elimination constraints particularly by Dantzig-Fulkerson-Johnson, Miller-Tucker-Zemlin, Gavish and Graves, and Claus. Besides, basic formulation types of TSPs are presented. However, this research only focuses on the first two types of the TSP which are the symmetric and asymmetric TSP. This followed by a basic information about the solution methods: Exact methods that find the optimal solution such as the brute-force method, branch-and-bound, cutting plane, branch-and-cut, cut and solve, column generation, and dynamic programming; Heuristic algorithms that find approximate solutions such as: constructive heuristics and improvement heuristics, and metaheuristics are classified into three groups, and are presented with basic information as follows:

1. Local search based metaheuristics such as simulated annealing (SA) and tabu search (TS).
2. Population-based metaheuristics such as genetic algorithm (GA) and Ant Colony Optimization (ACO).
3. Hybrid metaheuristic.

Chapter 2 provides background information of the Ant Colony Optimization (ACO) and its solution construction mechanism. An overview of different types of ACO algorithms such as Ant System, Elitist Ant System and Ant System are also included.

Chapter 3 describes the main procedures of the proposed algorithm; the tour construction process, pheromone update process and enhancement process. The tour construction process of the proposed algorithm uses a different formulation of state transition rule than the basic ACO; new ways of depositing and evaporating pheromone, and a different approach to global updating of pheromone. This proposed algorithm also uses a special agent

called ‘Intelligent Ants’ to work together with the state transition rule in the tour construction process. The concept of the intelligent ants is further described with an illustrated example.

Chapter 4 demonstrates the implementation and performance of the new proposed state transition rule of ACO to construct the solution tour. Issues relating to the ACO parameters such as initial pheromone value, the maximum number of ants in each colony, the maximum number of colonies are discussed and described through an empirical study. The computational experiments were conducted on three randomly selected TSP standard benchmark problems from the TSPLIB library.

Chapter 5 examines the factors that influence the performance of the proposed algorithm such as bound restriction, variable fixing, edge fixing and representation of the Subtour Elimination Constraints. Eight STSP and six ATSP benchmark problems were used to illustrate the effect of each of these factors.

Chapter 6 investigates the performance of the proposed algorithm on both symmetric and asymmetric TSP. Two kinds of experiments were carried out on 33 TSP benchmarks problems taken from the TSPLIB library. The first kind was used to evaluate the performance of the proposed algorithm using ACO parameters suggested in Chapter 4 against the performance of the proposed algorithm using the set of parameter values recommended in the literature by Dorigo. The second was carried out to compare performance with other studies available in the literature.

Chapter 7 concludes the thesis with a discussion of possible future research directions.

Chapter 2

Ant Colony Optimization

This chapter presents a brief overview to ACO, approaches to building ACO algorithms and an overview of different types of ACO algorithms.

2.1 Ant Colony Optimization Metaheuristics

The ACO algorithms are inspired by the behaviour of ants to find the shortest path to a food source from their nest. These ants find such a path very quickly by using indirect communication via pheromones. This inspiring behaviour is exploited in artificial ants that construct solutions for a given problem by carrying out random walks on a construction graph. The random walk and the resulting solution depend on pheromone values which represent the values on the edges of the construction graph. Concisely, the probability of traversing a certain edge depends on its pheromone value.

The main idea behind an ant algorithm is to use a form of artificial stigmergy to coordinate societies of artificial agents. One of the most prominent examples of the ant algorithm is the ACO. The ACO is inspired by the foraging behaviour of ant colonies and is used in solving the discrete optimization problem.

In the ACO algorithms, ants are agents programmed to find an optimal combination of elements of a given set that maximizes some utility function. Edges are used as solution components when applying the ACO algorithms to the TSP. The pheromone trails τ_{ij} associated to each edge i, j in the TSP refer to the desirability of visiting city j from the current city i. The amount of pheromone trail is proportional to the quality of the ant's path where a shorter path usually results in a greater amount of pheromone. If the ACO algorithm is applied to the symmetric TSP instances, the pheromone trails are also symmetric $\tau_{ij}^t = \tau_{ji}^t$.

At the beginning of the solution construction process, m numbers of ants are placed on a randomly chosen start city. Then, in each construction step, each ant chooses the next unvisited

city probabilistically, biased by the pheromone trail τ_{ij} and locally available heuristic information, which is a function of the edge length. This solution construction process terminates when all the cities have been visited. Once all ants have constructed a tour, the pheromone trails are updated. Edges that are used most frequently by many ants and contained in the shortest tour will usually receive more pheromone and thus are more likely to be chosen in a future iteration of the algorithm.

2.2 Foraging Behaviour of Real Ants

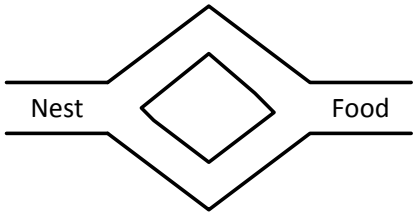
Although most ants are almost blind, to get around, they communicate and gain information about their world by relying on touch from the sensitive antennae, and the smell of chemicals called pheromones. Some ant species in particular, such as Lasius Niger or the Argentine ant *Iridomyrmex humilis* (Goss, Aron, Deneubourg, & Pasteels, 1989), use a special kind of substance called trail pheromones to reinforce the optimum paths between food sources and their nest. To be more specific, these ants lay pheromones on the paths they take, and these pheromone trails act as stimuli because the ants are attracted to follow the paths that have relatively more pheromones.

As a result, an ant that has decided to follow a path due to the pheromone trail on that path reinforces it further by laying its pheromone too. This process can be assumed as a self-reinforcement process since the more ants that follow a specific path, the more likely that it becomes the path that will be followed by other ants in the colony.

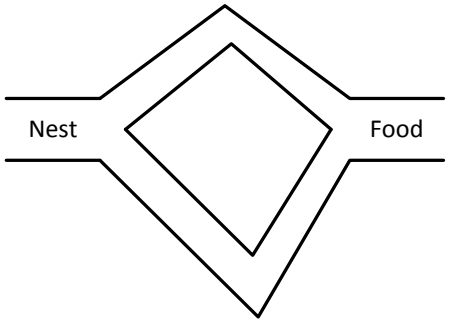
The pheromone trail-laying and foll**o**wing behaviour of some ant species have been investigated in controlled experiments by several researchers. Deneubourg et al. (Deneubourg, Aron, Goss, & Pasteels, 1990) demonstrated the foraging of a colony of ants through the double-bridge experiments. In these experiments, the nest of ants of the Argentine ant species *Iridomyrmex humilis* and the food sources are connected through two different paths. The behaviour is examined by varying the ratio between the lengths of the two paths of the double bridge as shown in Figure 2.1.

In the first experiment, both paths were set to be of equal length as can be seen in Figure 2.1 (a). The result showed that initially, the ants chose the two paths randomly since there was no pheromone on either of the paths yet. After a while, due to random fluctuations, one of the

two paths was followed by a few more ants and so more pheromone was accumulated on that path. Eventually, the whole colony converged to follow that same path.



(a) Paths have equal length



(b) Paths have different length

Figure 2.1: Double Bridge Experiments

In the second experiment, the length of one path was two times as long as the other one as in Figure 2.1(b). Initially, the ants again choose either of the two paths randomly. The ants that had chosen the shortest path arrived at the food source faster and began their return to the nest earlier. Consequently, pheromone accumulated faster on the shortest path, and most of the ants converged to this path.

Besides, Deneubourg et al. (Deneubourg, Aron, Goss, & Pasteels, 1990) were also interested in investigating what would happen if a shorter path was added after the ants had converged to one path. They found that the shorter alternative that was offered after convergence was never discovered by the colony. The majority of the ants continued following the longer branch reinforcing it more. This stagnation is caused by the high pheromone concentration and by the slow evaporation of pheromone, and the real ants always follow the suboptimal path even if there is a shorter one.

2.3 The Design of Artificial Ants

The double bridge experiments in Section 2.2 show that the ant colonies have a built-in optimization capability. These ant colonies make a probabilistic movement to find the shortest path between their nest and the food source based on the intensity of pheromone. In the ACO algorithm, artificial ants or agents are used to find good solutions to difficult combinatorial optimization problems. These artificial ants have the properties of the real ants. Blum in (Blum C. , 2005) explains the significant difference between the characteristics of the artificial ants and the real ants are as follows:

- When foraging for food, real ants will evaluate the intensity of pheromone along their way from the nest to the food source. Contradictorily, artificial ants will evaluate a solution with respect to some quality measure, which is used to determine the intensity of pheromone during their return trip to the nest.
- The real ants might not take the same path on their way to the food sources and on their return trip to their nest. However, each of the artificial ants moves from the nest to the food sources and follows the same path to return.
- The real ants lay pheromone each time they move back and forth to the nest while the artificial ants deposit artificial pheromone only on their way back to the nest.

The earliest ant algorithm was introduced by Dorigo et al. in 1991 and was called the Ant System (AS) (Colorni, Dorigo, & Maniezzo, 1991), (Dorigo, Maniezzo, & Colorni, 1996). Dorigo and Gambardella then proposed the Ant Colony System (ACS) (Dorigo & Gambardella, 1997a) (Dorigo & Gambardella, 1997b) in 1996 while Stützle and Hoos proposed the MAX-MIN Ant System (MMAS) (Stützle & Hoos, 2000). The ACO has drawn much attention, and various extended versions of the ACO paradigm were proposed, such as the Best-Worst Ant System (BWAS) (Cordon O. , Herrera, Viana, & Moreno, 2000) and the Rank-based Ant System (R-AS) (Bullnheimer, Hartl, & Strauss, 1999a).

2.4 Ant System

The Ant System (AS) is the first ACO algorithm proposed in the literature as a means of solving the TSP (Colorni, Dorigo, & Maniezzo, 1991). The AS consists of an initial phase and two

iterative main phases of the ant system which are ants' solution construction and pheromone update. In the initial phase, the pheromone trails are set with an equal amount of pheromone τ_0, such that $\forall(i,j), \tau_{ij} = \tau_0$.

In every iteration, μ ants construct their solution concurrently, each one starting from a randomly chosen city. Each ant k builds a solution city-by-city using a probabilistic decision rule, called random proportional rule. At each construction step, the city selected is added to the partial solution of the ant. In particular, the probability that ant k chooses the next city j, when the last city in the partial tour is i, is defined as follows:

$$p_{ij}^k = \frac{[\tau_{ij}]^\alpha [\eta_{ij}]^\beta}{\sum_{u \in N_i^k} [\tau_{iu}]^\alpha [\eta_{iu}]^\beta} \qquad if\ j \in N_i^k \tag{2.1}$$

where τ_{ij} is the present pheromone trail, η_{ij} is the heuristic information available a priori, α and β are the constant parameters that determine the relative influence of pheromone trail and the heuristic information, respectively, and N_i^k is the neighbourhood of unvisited cities of ant k when its current city is i. The heuristic information is defined as

$$\eta_{ij} = \frac{1}{c_{ij}} \tag{2.2}$$

which is inversely proportional to the distance between city i and j .

After all ants build a feasible solution T^k , the pheromone trails are updated. At the beginning, all the pheromone trails are lowered by a constant factor ρ, due to the pheromone evaporation, such that:

$$\tau_{ij}^{t+1} = (1-\rho)\ \tau_{ij}^t \qquad \forall(i,j) \in E \tag{2.3}$$

given that $0 < \rho \leq 1$ is the pheromone evaporation rate, which helps the ants to eliminate pheromone trails that are not used frequently and have been created from bad decisions previously taken. After evaporation, all ants deposit pheromone on the arcs of their path as follows:

$$\tau_{ij}^{t+1} = \tau_{ij}^t + \sum_{k=1}^{\mu} \Delta\tau_{ij}^k, \quad \forall(i,j) \in E \tag{2.4}$$

where $\Delta\tau_{ij}^{k}$ is the amount of pheromone ant k deposits on the arcs that belong to its tour T^{k}, and is defined as follows:

$$\Delta\tau_{ij}^{k} = \begin{cases} \frac{1}{L^{k}} & if\ (i,j) \in T^{k} \\ 0 & \text{otherwise} \end{cases} \qquad 2.5$$

where L^{k} is the tour cost of the tour T^{k} constructed by ant k. As a result, the amount of pheromone of each ant is proportional to the solution quality. Hence, the better the ants tour, the more pheromone an ant deposits. For more information on AS and its application see (Maniezzo & Colorni, 1999) and (Cordon, Herrera, & Stutzle, 2002).

2.5 Elitist Ant System

This was the first improvement on the original AS introduced in Dorigo et al. (Dorigo, Maniezzo, & Colorni, 1996), (Dorigo, Maniezzo, & Colorni, 1991). The Elitist Ant System (EAS) uses an elitist strategy where the best ant deposits additional pheromone to the edges of its tour. The initial phase and the solution construction are the same as in the AS algorithm. However, after the pheromone evaporation, all the ants deposit pheromone as follows:

$$\tau_{ij}^{t+1} = \tau_{ij}^{t} + \sum_{k=1}^{\mu} \Delta\tau_{ij}^{k} + e\Delta\tau_{ij}^{bs}, \qquad \forall (i,j) \in E \qquad 2.6$$

where $\Delta\tau_{ij}^{k}$ is defined as in equation 2.5, e is the parameter that determines the influence of the elitist strategy and $\Delta\tau_{ij}^{bs}$ is defined as follows:

$$\Delta\tau_{ij}^{bs} = \begin{cases} \frac{1}{L^{bs}} & if\ (i,j) \in T^{bs} \\ 0 & \text{otherwise} \end{cases} \qquad 2.7$$

where L^{bs} is the tour cost of tour T^{bs} which is constructed by the best-so-far ant. Note that the best-so-far ant is a special ant that may not belong to the population in every colony. For more information on EAS refer (Dorigo & Stutzle, 2004).

2.6 The Rank-based Ant System

The Rank-based Ant System (R-AS) was proposed by Bullnheimer et al. (Bullnheimer, Hartl, & Strauss, 1999a). In this R-AS, each ant deposits an amount of pheromone proportional to its rank where the best-so-far ant always deposits a higher amount of pheromone than the other ants as in the EAS. The initial phase and the solution construction are the same as in the AS algorithm. Though, after pheromone evaporation, all the ants are ranked according to their solution quality such that only the $\omega - 1$ best-ranked ants and the best-so-far ant are allowed to deposit pheromone. Formally, the pheromone update in the R-AS is according to:

$$\tau_{ij}^{t+1} = \tau_{ij}^{t} + \sum_{r=1}^{\omega-1} (\omega - r)\Delta\tau_{ij}^{r} + \omega\Delta\tau_{ij}^{bs} \qquad \forall (i,j) \in E \tag{2.8}$$

where $\Delta\tau_{ij}^{bs}$ is defined as in equation 2.7 and $\Delta\tau_{ij}^{r} = 1/L^{r}$ where L^{r} is the tour cost of T^{r} of the r-th best-ranked ant.

For more information on R-AS and its application see (Bullnheimer, Hartl, & Strauss, 1999b) , (Dorigo & Stutzle, 2004) and (Capriles, Fonseca, Barbosa, & Lemonge, 2007).

2.7 The MAX-MIN Ant System

The MAX-MIN Ant System (MMAS) is an improved algorithm of the EAS proposed by Stutzle and Hoos (Stützle & Hoos, 2000). Contrary to the previous AS variations, the MMAS only allows either the best-so-far ant or the colony-best ant to deposit pheromone. However, the initial phase and the solution construction phase are still the same as in the AS algorithm, whereas the pheromone update is defined as follows:

$$\tau_{ij}^{t+1} = \tau_{ij}^{t} + \Delta\tau_{ij}^{best}, \qquad \forall (i,j) \in T^{best} \tag{2.9}$$

where $\Delta\tau_{ij}^{best} = 1/L^{best}$. In case the best-so-far ant is allowed to deposit pheromone

$$\Delta\tau_{ij}^{best} = \frac{1}{L^{bs}} \tag{2.10}$$

while in the case the colony-best ant is allowed to deposit pheromone

$$\Delta\tau_{ij}^{best} = \frac{1}{L^{ib}} \quad 2.11$$

where L^{bs} is the tour cost of the best-so-far ant, and L^{ib} is the tour cost of the best ant of the current colony.

Moreover, the pheromone trails are bounded in the interval $[\tau_{min}, \tau_{max}]$, where τ_{min} and τ_{max} are the lower and upper limits, respectively. Since only the best ant is allowed to deposit pheromone, high intensity of pheromone may be generated on a suboptimal solution. Hence, the mechanism that restricts the range of the pheromone trails avoids stagnation behaviour. Finally, in the case of search stagnation or if no improvement is found for a given number of algorithmic iterations , the pheromone trails are re-initialized to an estimate of the upper pheromone trail limit to increase exploration.

For more information on MMAS and its applications refer to (Dorigo & Stutzle, 2004), (Stutzle & Hoos, 1996), (Afshar, 2006), (Socha, Knowles, & Sampels, 2002), (Zecchin, et al., 2003).

2.8 The Ant Colony System

The Ant Colony System (ACS) was introduced by Dorigo (Dorigo, Maniezzo, & Colorni, 1996); (Dorigo & Gambardella, 1997b) to improve the performance of the AS. The ACS is primarily different from the AS in three aspects:

- State transition rule.
- Global pheromone updating rules.
- Local pheromone updating rule.

The modification to the state transition rule is done to provide the ability to achieve a balance between exploring new arcs and exploiting accumulated knowledge about the problem. An ant k in city i chooses the city j to move to following the rule:

$$j = \begin{cases} \arg\max_{u \in N_i^k} \{[\tau_{iu}]^{\propto}[\eta_{iu}]^{\beta}\} & if\ q < q_0 \\ J & if\ q > q_0 \end{cases} \quad 2.12$$

where q is a random variable uniformly distributed over $[0,1]$ and a predefined parameter q_0 $(0 \leq q_0 \leq 1)$. J is a random variable determined in accordance with equation 2.1. It can be

seen that the ACS transition rule is identical to the AS's when $q > q_0$. This strategy obviously increases the variety of any searching, thus avoiding any premature falling into the local optimal solution.

In the AS, all ants are allowed to deposit pheromone after completing their tours while in the ACS, only the ant that has produced the best solution since the beginning of the trail is allowed to globally update the intensity of pheromone on the edges. The global pheromone updating rule is stated as follows:

$$\tau_{ij}^{t+1} = (1-\rho)\tau_{ij}^{t} + \rho.\Delta\,\tau_{ij}^{t} \qquad \forall(i,j) \in E \tag{2.13}$$

and $\Delta\tau_{ij}^{t}$ is defined as :

$$\Delta\tau_{ij}^{t} = \begin{cases} \frac{1}{L^{+}} & if\ (i,j) \in T^{+} \\ 0 & \text{otherwise} \end{cases} \tag{2.14}$$

where T^{+} is the best tour since the beginning of the trail, L^{+} is the length of T^{+} and ρ is a decay parameter .

Moreover, while building a solution, ants change their pheromone level by applying the local updating rule as in equation 2.15.

$$\tau_{ij}^{t+1} = (1-\rho)\tau_{ij}^{t} + \rho.\tau_0 \qquad \forall(i,j) \in E \tag{2.15}$$

where $0 < \rho \leq 1$ is a decay parameter and $\tau_0 = (n.L_{nn})^{-1}$ is the initial values of the pheromone.

For more information on the ACS and its application see (Dorigo & Gambardella, 1997b) and (Gambardella & Dorigo, 1996).

2.9 Summary

This chapter has reviewed the motivation, frameworks and variants of the ACO. In the next chapter, the new proposed modified ACO approach will be introduced and discussed in detail.

Chapter 3

Proposed Modified ACO Algorithm for Symmetric and Asymmetric TSP

In this chapter, a new formulation of the state transition rule will be implemented to select candidate solutions during the tour construction phase. Besides, a different approach to global updating of pheromone and new ways of depositing and evaporating pheromone are employed in the pheromone update phase. In addition, a special agent named 'Intelligent Ants' is also introduced to work with the state transition rule in the tour construction phase. Finally, an example is included to demonstrate the working process of the proposed algorithm.

3.1 Introduction

As mentioned in Chapter 2, there are three phases to ACO algorithms, namely the construction phase, the pheromone update phase, and the optional daemon phase. In the construction phase, the ants iteratively construct candidate solutions on which they may deposit pheromone. An ant constructs a candidate solution starting with an empty solution and iteratively adds the solution component until the complete candidate solution is generated. After the solution construction is completed, the ant will enter the pheromone update phase. In this phase, the ant gives feedback on the solution that has been constructed by depositing pheromone on that solution's components. Normally, solution components which are used by many ants or are part of better solutions will receive a higher amount of pheromone and, thus, will more likely be used by the ants in the future iterations of the algorithm. Conversely, the pheromone trails are decreased by a factor ρ which is called the evaporation factor.

The ants' solutions are not guaranteed to be optimal with respect to local changes and hence, more explorations are needed to search for global changes. However, the balance between exploration and exploitation has to be considered carefully. Excessive exploitation will reduce the diversity of the solution by focusing only on the neighbourhood and lead the search to local optima quickly. At the same time, extreme exploration will increase the diversity of

solutions but slow down the search's speed. So, an improper balance will lead to ineffective algorithms.

3.2 Algorithm Notations

The following notation is used in the proposed algorithm where index k denotes ant and indexes i and j denotes cities.

$N = \{1, \dots, n\}$: the set of cities

$t = \{1, \dots, m\}$: the set of colonies

$k = \{1, \dots, q\}$: the number of ants used in each colony

N_i^k : set of unvisited adjacent cities for ant k from city i

α : the relative importance of pheromone trail

τ_{ij} : pheromone intensity between cities i and j

τ_{ij}^t : the pheromone information between cities i and j at colony t

ρ : the pheromone evaporation rate

u_i : the position of city i in the solution tour

$T_{k_a}^t$: the tour constructed following the new proposed state transition rule of ant k at colony t

$L_{k_a}^t$: the length of the solution tour constructed following the new proposed state transition rule of ant k at colony t

T_k^t : the solution tour of ant k at colony t

L_k^t : the length of the solution tour of ant k at colony t

T_{best}^t : the colony-best solution tour of colony t

L_{best}^t : the length of the colony-best solution tour of colony t

T_{worst}^t : the colony-worst solution tour of colony t

L_{worst}^t : the length of the colony-worst solution tour of colony t

3.2.1 Initialization

In the basic ACO algorithm, the initial pheromone concentration is uniformly distributed. Thus, the probability of ants choosing other feasible directions at any node (city) is almost the same. Therefore, the ACO will take a lot of time to determine the feasible solution which results in slow convergence speed in the early stage of the algorithm. Although an ant is associated with heuristic information which provides local information of the problem, balancing the exploitation (pheromone value) and exploration (heuristic information) of the search needs many experiments and experience.

For that reasons, in the proposed algorithm, the initial quantities of the pheromone concentration are determined according to equation 3.1 where c_{ij} represents the cost between node i and node j. Equation 3.1 indicates that when the distance between cities i and j is short, the quantity of pheromone between cities i and j is large and hence, the probability to visit city j from city i become higher and vice versa. In short, the edges which have higher cost, obtain lower pheromone making the desirability of that edge to decrease

$$\tau_{ij}^{0} = \frac{1}{c_{ij}}, \quad \forall i, j \in N \tag{3.1}$$

3.2.2 Tour Construction

In the tour construction phase, q ants independently visit each city exactly once. Each ant starts from a randomly selected city, and probabilistically chooses the edge to follow among those that lead to yet unvisited cities. The probability of choosing city j from city i at colony t for ant k is computed by equation 3.2.

$$p_{ij}^{k} = \begin{cases} \dfrac{\tau_{ij}}{\sum_{j \in N_i^k} \tau_{ij}} & if\ j \in N_i^k \\ 0 & if\ j \notin N_i^k \end{cases} \tag{3.2}$$

where N_i^k is the set of unvisited adjacent cities for ant k in city i, and τ_{ij} denotes a quantity of pheromone between cities i and j.

3.2.2.1 Intelligent Ants Strategy

In the ACO algorithm, an ant is a simple computational agent which iteratively constructs a solution for the problem at hand. In the proposed algorithm, a strategy of 'Intelligent Ants' (IA) that combines a partial fixing approach with a local search method is implemented. This strategy has demonstrated effective practical procedures for commonly observed instances of important discrete optimization problems such in the works by (Crowder, Johnson, & Padberg, 1983) and (Crowder & Padberg, 1980). The partial fixing approach fixes a predefined number of arcs and leaves the remaining unfixed nodes to the solver to decide. This strategy helps at reducing the computational time by heuristically fixing part of the solution tour and improving the accuracy of the solutions through the usage of the solver. The 'Intelligent Ants' procedure is summarised as follows:

Step 0: Initialize the value for $T_{k_a}^t$, $L_{k_a}^t$, $T_k^t = \{\}$ and $L_k^t = \infty$.

For $x(i,j) \in T_{k_a}^t$ and $u_i \neq \text{last}\,(T_{k_a}^t)$:

Step 1: Fix the first $X * n$ number of cities form the initial solution tour.

Step 2: Complete the solution tour constructed in Step 1 using the solver and update the T_k^t and L_k^t.

For instance, if the size of partial fixing is set to 50% of the total number of nodes, then $X = 0.5$ and the number of nodes to be fixed is $0.5 * n$. Note that $0 \leq X \leq 1$. For example, for berlin52 instance with 52 cities, the size of partial fixing is 26 nodes while for u159 instance with 159 cities, the size of the partial fixing is 80 nodes.

3.2.2.1.1 An Illustrative Example

Consider a TSP benchmark problem ulysses16 with $n = 16$ and $X = 0.5$;

Step 1: In the tour construction phase, nodes are gradually added to the solution tour according to the state transition rule in equation 3.2. When the number of nodes in the solution tour has reached the maximum number of nodes allowed in the 'Intelligent Ant' procedure, stop. Otherwise, repeat the same process.

Step 2: The solver is used to build a complete closed tour based on the partial route constructed in Step 1.

Figure 3.1 and Figure 3.2 illustrate the partial fixing process in the IA procedure for ulysses16.

15	5	11	9	10	1	8	13

Figure 3.1: Partial solution route constructed in the tour construction phase

15	5	11	9	10	1	8	13	12	16	3	2	4	14	7	6

Figure 3.2: A closed solution tour constructed following the IA procedure

3.2.2.2 Local Search Strategy

In the interest of effectiveness and efficiency, a local search technique called 2-opt is adopted in the proposed algorithm. The local search is applied after the ants have constructed a feasible solution which allows the algorithm to search for a solution that might have a lower cost. The 2-opt move in general consists of removing two random edges and reconnecting the resulting paths into a new tour. The pair that gives the shortest tour among all pairs of edges after the 2-opt exchange will be chosen. This procedure is iterated until no such pair of edges is found. The 2-opt local search procedure is defined as below:

Step 0: Initialize the solution tour T_k^t and its length L_k^t .

For $x(i,j) \in T_k^t$, $u_i = 1, \dots, n-1$, $u_j > u_i$ and $u_j < n$:

Step 1: Swap the position of a pair of cities (u_i, u_j) from the solution tour and calculate its new length L_k^{t*}.

Step 2: If the new length L_k^{t*} is shorter than the previous one L_k^t , update the T_k^t and L_k^t. Otherwise, swap a different pair of cities from the solution tour.

Step 3: Repeat Step 1-2 until no other shorter tour is possible.

3.2.3 Pheromone Update

The pheromone updating rule is meant to simulate the change in the amount of pheromone due to both the addition of new pheromone deposited by ants on the visited edges and for pheromone evaporation (Dorigo & Gambardella, 1997a). By using this rule, ants will search in a wide neighbourhood of the best previous schedule.

In the proposed algorithm, the pheromone is deposited only on the edges belonging to the colony-best solution. Likewise, the pheromone is evaporated only on the edges belonging to the colony-worst solution that are not in the colony-best solution. The underlying idea of the proposed strategy in the context of the proposed algorithm is to place extra emphasis on the best edges found in each colony and make edges of the colony-worst solution become less attractive provided that these edges are not part of the colony-best solution.

In order to prevent the solution from falling into a local optimum, the pheromone evaporation is utilized. Every quantity of pheromone is reduced with the following equation:

$$\tau_{ij}^{t+1} = \rho \cdot \tau_{ij}^{t} \quad ; \quad (i,j) \in T_{worst^*}^{t} \tag{3.3}$$

where

$$T_{worst^*}^{t} = \left\{ (i,j) \middle| (i,j) \in \left\{ T_k^{worst} \cap \overline{T_k^{best}} \right\} \right\} \tag{3.4}$$

In this way, the edges of the colony's longest path $T_{worst^*}^{t}$ become less desirable.

Following the pheromone evaporation, the colony-best ant deposits pheromone as follows:

$$\tau_{ij}^{t+1} = \tau_{ij}^{t} + \Delta\tau_{ij}^{t_{best}} \quad (i,j) \in T_{best}^{t} \tag{3.5}$$

where $\Delta\tau_{ij}^{t_{best}}$ is a quantity of pheromone between cities i and j deposited by the colony-best ant and computed based on the following formula:

$$\Delta\tau_{ij}^{t_{best}} = \begin{cases} \dfrac{\alpha}{L_{best}^{t}} & \text{if } (i,j) \text{ belongs to } T_{best}^{t} \\ 0 & \text{otherwise} \end{cases} \tag{3.6}$$

Given that L_{best}^{t} is the length of the colony-best solution and α is the persistence of the pheromone trail.

In such a way, the edges of the shortest path of the colony become more attractive and are updated based on the values of the L_{best}^{t}.The lower the value of the L_{best}^{t}, the greater the pheromone deposited on the visited edges.

3.3 Design of the Proposed Algorithm

The proposed algorithm starts by adopting the Nearest-neighbour heuristic with the 2-opt local search to construct its first colony. The solution with the minimum tour length is considered as the colony-best solution T_{best}^{t} and is set as the first ant of the next colony.

Then, at colony $t+1$ provided that $t \geq 1$, the proposed algorithm begins by randomly placing ants in the nodes of the graph in which every ant moves to a new node and the parameters controlling the algorithm are updated. Assuming that the TSP is represented as a closed connected graph, each edge is labelled by trail intensity τ_{ij}^{t} at colony t. An ant decides the next node with a probability that is based on the distance to that node and the amount of trail intensity on the connecting edge. A function p_{ij}^{k} is considered to favour the selection of an edge that has a high intensity of pheromone trail when N_i^k are the unvisited neighbours of node i by ant k and $j \in N_i^k$.

However, the numbers of nodes selected to be included in the solution tour are dependent on the predefined value in the IA strategy. When the number of nodes selected is equal to the predefined value, the IA uses the solver to construct a complete tour with minimum tour length. If the tour constructed is greater than the current best solution, the tour is terminated. Otherwise, a 2-opt local search is applied to further enhance the tour. This tour construction phase is repeated until all the ants have completed their tours.

Once all the ants have constructed a tour, the pheromone trails are updated. The pheromone updating rule enforces two things; pheromone evaporation which stops pheromone trails from unlimited accumulation, and pheromone deposit which makes the favourite edges have stronger pheromone trails. In each colony, only the ant that generates the colony-best solution T_{best}^{t} is allowed to globally update the pheromone. Likewise, the pheromone evaporation is only applied on all arcs belonging to the colony-worst solution T_{worst}^{t} that were not in the colony-best solution T_{best}^{t}, and the amount of its evaporation is dependent on the pheromone evaporation rate ρ. After the evaporation process, the quantity of pheromone

deposited on each arc of the colony-best solution is inversely proportional to the cost of the colony-best solution.

Before the algorithm starts the next iteration, the colony-best solution is used as a new set of cities, N as well as being the first ant in the following colony. Therefore, the colony-best solution is also the global-best solution which prevents a worse quality solution in the next colony. The algorithm runs until the maximum number of colonies allowed is reached.

3.3.1 Algorithm

The algorithm proposed is described as follows:

Step 0: Initialize the initial value for $\tau_{ij}^{t} = \frac{1}{c_{ij}}$, $T_{best}^{1} = \{\}$, $T_{worst}^{t} = \{\}$, $L_{best}^{1} = 0$ and $L_{worse}^{t} = \infty$.

For $t = \{1\}$, $k = \{1, \dots, q\}$:

Step 1: Randomly pick an initial city.

Step 2: Choose the next city to move to from the list of unvisited cities using the Nearest-neighbour approach.

Step 3: Repeat Step 2 until all unvisited cities have been visited and compute the tour length.

Step 4: Apply the 2-opt local search and update the T_k^t and L_k^t.

Step 5: Repeat Step 1-4 until all ants have constructed their solutions.

Step 6: Calculate and update the T_{best}^{t} and L_{best}^{t}.

For $t = \{2, \dots, m\}$ and $k = \{1\}$:

Let $T_1^t = T_{best}^{t-1}$ and $L_1^t = L_{best}^{t-1}$.

For $t = \{2, \dots, m\}$, $k = \{2, \dots, q\}$:

Step 1: Randomly pick an initial city.

Step 2: Choose the next city to move to from the list of unvisited cities using the new proposed state transition rule.

Step 3: Repeat Step 2 until all unvisited cities have been visited.

Step 4: Fix the first $X * n$ number of the cities in Step 3.

Step 5: Complete the solution tour constructed in Step 4 using the solver and check for feasibility.

Step 6: If the solution tour found in Step 5 is feasible, apply the 2-opt local search and return T_k^t and L_k^t. Otherwise, let $T_k^t = \{\}$ and $L_k^t = \infty$.

Step 7: Repeat Step 1-6 until all ants have constructed their tours.

Step 8: Calculate and update the T_{best}^t, T_{worst}^t, L_{best}^t and L_{worst}^t.

Step 9: Reinforce the pheromone value on the arcs belong to the T_{best}^t and evaporate on the arcs belong to the T_{worst}^t but not in the T_{best}^t.

3.3.2 An Illustrative Example

A symmetric instance with 16-cities is used to demonstrate the procedures when the proposed algorithm is applied. In this example, the number of cities n is 16 and the value of X in the IA procedure is 0.5; hence the number of nodes to be fixed is 8 (excluding the first node). If the number of colonies and ants is, respectively, 15% and 20% of the problem size, then the number of colonies is 2 and the number of ants in each colony is 3. The algorithm works as follows:

Colony 1:

In this colony, $0.2n$ number of ants build their solutions using the Nearest-neighbour heuristic with 2-opt local search as shown in Table 3.1. After all the ants have constructed their tours, the tour with the minimum length will be chosen as the colony-best solution T_{best}^1.

Ant	Route
1	T_1^1 : 16 12 13 14 7 6 15 5 11 9 10 8 4 2 3 1 L_1^1 : 7068
2	T_2^1 : 16 12 13 14 7 6 15 5 11 9 10 8 4 2 3 1 L_2^1 : 7068
3	T_3^1 : 9 11 5 15 14 13 6 7 12 16 1 8 4 2 3 10 L_3^1 : 7033

Table 3.1: List of solution tours of ulysses16 in the first colony

From the solutions in Table 3.1, the colony-best solution is

T_{best}^1 : 9 11 5 15 14 13 6 7 12 16 1 8 4 2 3 10

L_{best}^t : 7033

Colony 2:

The first ant in this colony inherited the previous colony-best solution as its solution tour. Thus, $T_1^2 \equiv T_{best}^1$.

The other ants construct their route following the new proposed state transition rule, IA and the 2-opt procedure. As displayed in Table 3.2, a 'list of visited cities' is created using the state transition rule while the 'IA' fixes the first half of the 'list of visited cities' and sends this partial route to the solver. The 'solution tour' is the solution tour generated by the solver. The '2-opt' is the resultant tour of the 2-opt moves. These procedures are repeated for each ant in the colony.

After all the ants have constructed their routes, the route with the shortest length is selected as the colony-best solution and the route with the longest length is selected as the colony-worst solution. Therefore,

$T_{best}^2 = T_3^2$:

(8,1)(1,16)(16,12)(12,13)(13,14)(14,7)(7,6)
(6,15)(15,5)(5,11)(11,9)(9,10)(10,3)(3,2)(2,4)

L_{best}^2 : 6913

and

$T^2_{worst} = T^2_2$:

(7,6)(6,15)(15,5)(5,11)(11,9)(9,10)(10,4)(4,2)

(2,3)(3,16)(16,1)(1,8)(8,14)(14,13)(13,12)

L^2_{worst}: 7157

Before moving to the next colony, arcs belonging to the colony-best ant will receive more pheromone while arcs belonging to the colony-worst solution that are not in the colony-best solution will receive less pheromone.

Arcs that belong to the colony-worst solution :

T^2_{worst}:

(7,6)(6,15)(15,5)(5,11)(11,9)(9,10)(10,4)(4,2)

(2,3)(3,16)(16,1)(1,8)(8,14)(14,13)(13,12)

Arcs that belong to the colony-best solution :

T^2_{best} :

(8,1) (1,16) (16,12) (12,13) (13,14)(14,7)(7,6)

(6,15)(15,5)(5,11)(11,9)(9,10)(10,3)(3,2)(2,4)

Hence, the arcs belongings to the colony-worst solution but not in the colony-best solution are

T^2_{worst}:

~~(7,6)~~ ~~(6,15)~~ ~~(15,5)~~ ~~(5,11)~~ ~~(11,9)~~ ~~(9,10)~~ **(10,4)** ~~(4,2)~~

~~(2,3)~~ **(3,16)** ~~(16,1)~~ ~~(1,8)~~ **(8,14)** ~~(14,13)~~ ~~(13,12)~~ **(12,7)**

T^2_{best} :

~~(8,1)~~ ~~(1,16)~~ (16,12) ~~(12,13)~~ ~~(13,14)~~(14,7) ~~(7,6)~~ ~~(6,15)~~

~~(15,5)~~ ~~(5,11)~~ ~~(11,9)~~ ~~(9,10)~~ (10,3) ~~(3,2)~~ ~~(2,4)~~ (4,8)

$T_k^{worst} \cap \overline{T_k^{best}}$: (10,4) (3,16) (8,14) (12,7).

Ant	Route
1	T_1^2 : 9 11 5 15 14 13 6 7 12 16 1 8 4 2 3 10 L_1^2 : 7033
2	List of visited cities : 7 6 15 5 11 9 10 2 4 8 1 16 12 13 14 3 IA : 7 6 15 5 11 9 10 2 4 Solution tour: 7 6 15 5 11 9 10 2 4 3 16 1 8 14 13 12 (7665) 2opt : 7 6 15 5 11 9 10 4 2 3 16 1 8 14 13 12 (7157) T_2^2 : 7 6 15 5 11 9 10 4 2 3 16 1 8 14 13 12 L_2^2 : 7157
3	List of visited cities : 8 1 16 12 13 14 7 6 15 5 11 9 10 3 2 4 IA : 8 1 16 12 13 14 7 6 15 Solution tour: 8 1 16 12 13 14 7 6 15 5 11 9 10 3 2 4 (6913) 2opt : 8 1 16 12 13 14 7 6 15 5 11 9 10 3 2 4 (6913) T_3^2 : 8 1 16 12 13 14 7 6 15 5 11 9 10 3 2 4 L_3^2 : 6913

Table 3.2: List of solution tours of ulysses16 in the second colony

The pheromones update phases will deposit and evaporate the following arcs accordingly:

Deposit : (8,1)(1,16)(16,12)(12,13)(13,14)(14,7)(7,6)
(6,15)(15,5)(5,11)(11,9)(9,10)(10,3)(3,2)(2,4)

Evaporation : (10,4) (3,16) (8,14) (12,7)

The procedure is terminated when the number of colonies reached 15% of the problem size. Therefore, the solution tour produced by the proposed algorithm is

8 1 16 12 13 14 7 6 15 5 11 9 10 3 2 4

with the length of 6913 unit.

3.4 Conclusion

This research proposes a new method based on the ant colony optimization for solving STSP and ATSP problems. This algorithm implemented a new state transition rule formulation along with an 'Intelligent Ants' strategy to construct a solution tour with minimum length. The 2-opt local search is then employed to enhance this tour before the pheromone is updated. The pheromone is evaporated on arcs belonging to the colony-worst solution provided that it does not belong to the colony-best solution while the pheromone is deposited only on arcs belonging to the colony-best solution. An illustrative example of a 16-cities problem is also included in this chapter to demonstrate the procedures of the proposed algorithm.

The choice of ACO parameters setting applied to the proposed algorithm is described in Chapter 4 while Chapter 5 shows experimental results in determining the best approach or value that helps to boost the performance of the proposed algorithm.

Chapter 4

The ACO and its Parameters

The main parameters of the ACO algorithm include pheromone concentration information α, heuristic information β, pheromone evaporation rate ρ and the number of ants in the colony m. Setting up the values of these parameters is crucial for good performances of an algorithm. In practice, parameter values are usually selected by experimental comparisons as in (Dorigo & Stutzle, 2004) or through adaptive parameter setting as in the works of Watanabe, Pilat and Gambardella.

4.1 Introduction

The study of the impact of various parameters on the behaviour of the ACO algorithms has been an important subject since the first articles by (Dorigo, Maniezzo, & Colorni, 1996). The values of these parameters determine whether the algorithm will find an optimal or near-optimal solution, and whether it will find such a solution efficiently. However, finding the appropriate settings of an algorithm's parameters is considered to be a non-trivial task and a substantial amount of work has been devoted to it.

The process of finding the appropriate setting of these parameters are commonly known as parameter setting and can be further categorized into parameter tuning and parameter control (Eiben, Hinterding, & Michalewicz, 1999). Parameter tuning can be expressed as a process of finding the correct combination of an algorithm's parameters for each individual problem in order to find the optimal solution (Bhríde, McGinnity, & McDaid, 2005). Meanwhile, Eiben et. al. (Eiben, Hinterding, & Michalewicz, 1999) defined parameter tuning as an approach for finding good values of the parameters before deploying the algorithm and then running the algorithm using these values, which remain fixed during the run. Alternatively, parameter control starts a run with initial parameter values that are changed during the run. For instance, parameter setting methods are classified depending on whether they attempt to set parameters before the run (tuning) or during the run (control).

4.2 Experimental Settings

In the following experiments, three STSP benchmark problems were considered from the TSPLIB where the number of cities varied from 52 to 100. A numeric value in the problem name indicates the number of cities in that instance. As an example, berlin52 has 52 cities.

The proposed algorithm (MACO) was implemented in AMPL (AMPL, 2013) using CPLEX 12.5.1.0 as the MIP solver and the computational experiments were conducted on a PC with an Intel (R) Core (TM) i5-3470 processor with 3.20 GHz and 8.00 GB of RAM.

4.3 Parameter Tuning for the ACO Algorithm

The behaviour of the ACO algorithm depends strongly on the values given to its parameters. In most ACO applications, parameter values are kept constant throughout each run of the algorithm. However, varying the parameters at computation time may enhance the performance of the algorithm. In the ACO literature, several strategies have been proposed and tested for modifying parameters while solving a problem instance.

Dorigo and Gambardella (Dorigo & Gambardella, 1997a) have proposed the optimum ant colony size in ACS. Their experimental observation has shown that the ACS works well when the number of ants is 10. In addition, they conclude that the optimum number of ants is influenced by the problem size.

Gambardella and Dorigo (Gambardella & Dorigo, 2000) expressed the parameter q_0 in equation 2.12 as a function of the problem size for a sequential ordering problem (SOP). The value of q_0 is given by $q_0 = 1 - s/n$ which makes q_0 dependent on the problem size n and s is the expected number of nodes selected by the probabilistic transition rule.

Pilat and White (Pilat & White, 2002) suggested two hybrid methods which incorporate GA into ACS. The first method uses a GA to evolve a population of genetically modified ants to improve the performance of the ACO algorithm. However, this algorithm did not find significant results in determining optimum solutions when compared to the ACS. The second method uses a GA to evolve the optimal parameter values used in the ACS. The algorithm results suggested that the performance of the ACS can be improved by using these values. Again, it concludes that the performance of the algorithm is influenced by the parameter values.

Watanabe and Matsui (Watanabe & Matsui, 2003) developed a mechanism to dynamically tune the size of the candidate set in the ACS. This candidate set was used to restrict the search space only to promising regions. With this mechanism, it is not necessary to set the size of the candidate set in advance. The computational results with several graph colouring instances indicate that the proposed control mechanism can potentially improve the efficiency of the ACS, especially for large optimization problems.

Qin et al. (Qin, et al., 2006) used self-adaptive ACO to solve a phylogenetic tree construction problem. The adaptive term in this algorithm refers to dynamically tuning the value of parameters α and β. The tuning method is based on the strength of the pheromone on the edges. At the initial stage of the algorithm, the pheromone value on each edge is relatively small. To speed up the convergence, the ants should select the path according to the heuristic information. Thus, the value of parameter α should be relatively large at this stage. After some iteration, the pheromone values on the edges are increased, thus, their influence will become more and more important. Therefore, the value of β will be relatively large. Experimental results show that the proposed method has better performance than the GA.

Hao et al. (Hao, Cai, & Huang, 2006) introduced an adaptive parameter strategy based on PSO for the ACO. The PSO works by moving particle swarms which contain ACO parameters in the search space when a new best solution is encountered. The test results on 10 benchmark TSP problems show that the PSO-ACS performs better than the ACS. In addition, Hao with different groups of researchers also examined dynamic parameter tuning for the weight importance of heuristic information β (Huang, Yang, Hao, & Cai, 2006) and trail persistence ρ (Hao, Huang, Qin, & Cai, 2007), and proved that both algorithms are more effective than the traditional ACO.

Favuzza et. al. (Favuzza, Graditi, & Sanseverino, 2006) used an adaptive instead of fixed, parameter q_0 as in equation 2.12 to push exploration or exploitation to escape local minimum for a dynamic optimization problem. The parameter q_0 varies adaptively based on the number of unimproved iterations. If the number of unimproved iterations reaches a certain value, then the value of the parameter q_0 will be decreased allowing the algorithm to focus its attention on the diversification process. Once the algorithm leaves the local convergence, the q_0 value will be increased, allowing the intensification process to happen. The proposed

algorithm has proven to be robust in finding the optimal reinforcement strategy for a distribution system problem.

Randall (Randall, 2004) proposed a near parameter free ACO. The author integrated the parameter search process with the running of ACO and thus removes the need of tuning the parameters by hand. The proposed method shows comparable results to the standard implementation of ACO.

Amir et al (Amir, Badr, & Farag, 2007) developed a Fuzzy Logic Controller (FLC) module embedded in the ACS algorithm. The FLC is used to tune the parameters β and q_0 according to robust performance measures of the algorithm. The rule-base of the fuzzy controller represents the fuzzy rules that govern the performance of the ACS algorithm in response to the changes in the parameters' values. The fuzzy rules were deduced using a genetic algorithm that produces its output with the help of a data set. The test results show that the adaptive ACS converged faster and outperformed the standard ACS.

Castillo et al. (Castillo, Neyoy, Soria, Melin, & Valdez, 2015) presented a new fuzzy approach to prevent the total convergence through the dynamic variation of parameter α in the ACO by maintaining a certain reference level of the average lambda branching factor. This average lambda branching factor is used to provides an indication of the size of the search space effectively explored and measured the distribution of the values of the pheromone trails. When the value of the average lambda branching factor reference level changed, the value of the parameter α is increased to maintain the diversity of the search. The proposed strategy shows an improvement when compared to the AS, R-AS and EAS.

4.4 Parameter Tuning for the Proposed Algorithm

The process of finding the appropriate parameter values for metaheuristic search algorithm can be a time consuming and tedious task. An alternative approach to setting good initial parameter settings is desired. Therefore, in order to reduce the amount of time spent in tuning the parameters, the parameters of the proposed algorithm are expressed as a function of the problem size. In such a way, the proposed algorithm can be used on various problem instances in which the number of ants m, α and ρ are peculiar to each problem.

4.5 The Number of Ants in a Colony

In the ACO algorithm, the number of ants in each colony determines the diversification in the search space. As an ant represents a solution and determines the diversification of the search space, a small number of ants tend not to find a good solution due to a low exploration of the search space. Contrarily, a large number of ants can sometimes be too much and does not reflect an improvement of the current best solution. Also, it requires more computational processing time.

For certain computational conditions such as maximum computation time, the number of ants plays a critical parameter for determining the trade-off between the maximum numbers of colonies and broadness of the search at each of the colony.

In order to estimate the optimal number of ants and investigate their impact on the proposed algorithm solutions, five conditions were proposed:

- the first one with ants equal to 5% of the problem size
- the second with ants equal to 10% of the problem size
- the third with ants equal to 15% of the problem size
- the fourth with ants equal to 20% of the problem size
- the fifth with ants equal to 25% of the problem size

$\alpha = 1 \quad \rho = 0.5$ colonies $= 5$					
Benchmark problems	Number of ants (m)				
	$0.5n$	$0.1n$	$0.15n$	$0.2n$	$0.25n$
berlin52	10384	9519	9452	**8500**	8664
eil76	**682**	719	742	689	752
krob100	32822	35293	**27229**	28860	30502

Table 4.1: Best solutions found for a different number of ants

As displayed in Table 4.1, the numerical results indicate that the number of ants needed to find the best solution is:

- berlin52 - 20% of the problem size
- eil76 - 5% of the problem size

- krob100 - 15% of the problem

Hence, the number of ants needed to find the best solution for all the three instances is at most 20% of the problem size.

4.6 Relative Influence of Pheromone Trail α and Relative Influence of Heuristic Information β

These two parameters are used to weight the influence of the pheromone trail and heuristic information in the ants' solution construction phase. In the case of $\alpha = 0$, nodes with better heuristic preference have a higher probability of being selected, thus making the algorithm close to a classical probabilistic greedy algorithm. Likewise, if $\beta = 0$, only the pheromone trails are considered to guide the constructive process, which may cause a quick stagnation. This stagnation normally occurs if the pheromone trails associated with some transitions are significantly higher than the remainder, making the ants always build the same solutions, usually a local optima. Therefore, there is a need to establish a proper balance between the importance of heuristic and pheromone trail information.

However, in the proposed algorithm, the heuristic information β is not considered in the solution construction process. This would allow the proposed algorithm to simulate a real-world situation where such kind of information may not be available or too expensive to compute.

In order to estimate the value of the relative influence of pheromone trail parameter α on the proposed algorithm solution, four conditions were proposed which are $0.5n^2$, n^2, $0.5n^3$ and n^3 where n represents the problem size.

$m = 0.2n \quad \rho = 0.5 \quad \text{colonies} = 5$					
Benchmark problems	n	α			
		$0.5n^2$	n^2	$0.5n^3$	n^3
berlin52	52	**7775**	**7775**	7920	7920
eil76	76	**581**	602	602	602
krob100	100	23142	**2295**	23656	23656

Table 4.2: Best solutions found for different value of the relative importance of pheromone trail parameter α

Table 4.2 presents the best result of the simulations for each condition. These results denote that the most promising value for the pheromone trail parameter α is either $0.5n^2$ or n^2.

4.7 Evaporation Rate

The pheromone evaporation rate ρ which represents the degree of pheromone evaporation, reflects the degree of mutual influence among ants. The value of ρ prevents the infinite accumulation of pheromone effectively and helps to eliminate the trails of solutions that may bias the ants to search in non-promising areas of the search space. If the evaporation rate value is too small, the global search ability of the proposed algorithm will be reduced. Conversely, it will improve the global search ability but with slower convergence speed.

In order to estimate the best value of the pheromone evaporation rate on the proposed algorithm solution, four possible values were proposed for the pheromone evaporation rate:

- $1/0.5n^2$
- $1/n^2$
- $1/0.5n^3$
- n^3

Besides, two different values of the pheromone trail parameter α are considered for each of the above conditions which are $0.5n^2$ and n^2.

	$m = 0.2n$ colonies $= 5$							
Benchmark problem	$\alpha = 0.5n^2$				$\alpha = n^2$			
	ρ							
	$\frac{1}{0.5n^2}$	$\frac{1}{n^2}$	$0.5n^3$	n^3	$\frac{1}{0.5n^2}$	$\frac{1}{n^2}$	$0.5n^3$	n^3
berlin52	11854	**7860**	**7860**	**7860**	7920	7920	7920	7920
eil76	793	**595**	**595**	**595**	602	602	602	602
krob100	42295	**23137**	**23137**	**23137**	23137	23137	23137	23137

Table 4.3 : Different evaporation rate value when $\alpha = n^2$ and $\alpha = 0.5n^2$

Table 4.3 compares the performance of the proposed algorithm with a different combination of parameter values for the test instances. These results show that for all the test instances, good solutions were obtained when the value of the pheromone trail parameter α is $0.5n^2$ and the pheromone evaporation rate ρ is either $\frac{1}{n^2}$ or $0.5n^3$ or n^3.

Benchmark problem	$\alpha = 0.5n^2$			
	$\rho = \frac{1}{0.5n^2}$	$\rho = \frac{1}{n^2}$	$\rho = 0.5n^3$	$\rho = n^3$
berlin52	11854, [44.518]	**7860,[223.963]**	7860,[231.437]	7860,[228.407]
eil76	793,[347.141]	**595,[422.226]**	595,[449.623]	595,[457.698]
krob100	42295,[5482.96]	**23137,[6405.08]**	23137,[6681.08]	23137,[6559.01]

Table 4.4 : Computational time for different evaporation rate value

Further, Table 4.4 showed that among these three possible combinations, $\alpha = 0.5n^2$ and $\rho = \frac{1}{n^2}$ has a better computational time than others. Note that for each combination in Table 4.4, the value in brackets represents the computational time while the other value represents the best solution found for that instances. Therefore, it can be concluded that the best parameters combination for the proposed algorithm are $\alpha = 0.5n^2$ and $\rho = \frac{1}{n^2}$.

4.8 Comparative Analysis of the Proposed Algorithm Using Different Set of Parameter Values

The parameter settings for the experimental runs are shown in Table 4.5. The parameter's value for the proposed algorithm using Dorigo's (MACOd) is from (Dorigo, Maniezzo, & Colorni, 1996) while the value of the parameters for the proposed algorithm (MACO) are summarised from Section 4.5, Section 4.6 and Section 4.7.

Algorithm	Parameters				
	m	α	β	ρ	Colonies
MACOd	n	1	-	0.5	10
MACO	$0.2n$	$0.5n^2$	-	$\frac{1}{n^2}$	10

Table 4.5: ACO parameters setting for the experimental runs

Benchmark problem		MACOd	MACO
berlin52	Optimal solution	8257	**7775**
	Number of colonies	10	8
	CPU time (seconds)	3057.31	**624.261**
eil76	Optimal solution	**559**	567
	Number of colonies	10	10
	CPU time (seconds)	7482.93	1766.91
krob100	Optimal solution	24155	**22842**
	Number of colonies	5	9
	CPU time (seconds)	35812.3	**11860.4**

Table 4.6 : Comparative results for MACO and MACOd on berlin52,eil76 and krob100 instances

Based on the results presented in Table 4.6, it can be seen that:

- For berlin52 : the MACO has better solutions quality, less number of iterations and better CPU time when compared to the MACOd.
- For eil76 : although the solution quality of the M-ACO is slightly worse from the MACOd, the CPU time recorded for the MACO is 76% less than the CPU time of the MACOd.
- For krob100 : the MACO is more effective and efficient than the MACOd .

4.9 Conclusion

The ACO parameters of the MACO are chosen depending on the problem size. This will significantly reduce the time spent for parameter tuning. In addition, the MACO has proved that these parameters value are able to achieve good solutions for the test instances with less computational time.

Based on the computational experiments on these three datasets, it can be concluded that the most possible ideal combinations of parameters for the MACO are:

- The number of ants is at most 20% of the problem size ; $m = 0.2n$
- The relative influence of the pheromone trail is 50% of the square of the problem size ; $\alpha = 0.5n^2$
- The pheromone evaporation rate is computed as an inverse square of the problem size; $\rho = 1/n^2$

The next chapter will review some of the factors that play an important role in the performance of the proposed algorithm. Thus, unless stated otherwise, the parameters in the subsequent investigation are set to these values.

Chapter 5

Factors Influencing the Performance of the Proposed Algorithm

This chapter discusses the implementations and factors that influence the performance of the proposed algorithm. The investigation will be based on computation experiments, which will be made on two set of benchmark problems; selected STSP and ATSP instances from the TSPLIB standard library.

5.1 Introduction

In general, the performance of an algorithm can be evaluated mainly in two aspects which are effectiveness and efficiency. Effectiveness may refer to the quality of the solution concerning the objective function value while efficiency usually relates to the computational cost required to run the algorithm. An algorithm is said to be efficient if its computational cost is at or below some reasonable amount of time on an available computer. The two most common ways to measure computational cost are speed and memory usage. The speed or time complexity defines the amount of time needed for an algorithm to execute. Likewise, the memory usage describes the amount of memory required for an algorithm to execute.

However, in practice, there are various factors which can affect the efficiency and computational time of an algorithm such as the language used, type of computing hardware and optimisation in the compiler. To minimise the effects of such factors, few strategies have been implemented on the proposed algorithm including variable fixing, bound tightening, a heuristic approach used to construct the initial solution and a number of variables fixed in the tour construction phase.

5.2 Test Instances

The experiments were carried out on 8 STSP and 6 ATSP benchmark problems selected from the TSPLIB standard library. The selected benchmark problems range in size from 17 cities up to 323 cities. The best-known solutions for these problems were taken from the (TSPLIB,

2014b). A city in a benchmark problem is represented as a coordinate; therefore, the TSP cost matrix is calculated by the Euclidean distance as in equation 5.1 and then rounded off to the nearest integer.

$$c_{ij} = int\left(\sqrt{\left(x_i - x_j\right)^2 + \left(y_i - y_j\right)^2}\right) \quad 5.1$$

The following measures are used in order to evaluate the performance of the proposed algorithm in all the subsequent tables:

- Best-known solution - best solution given by the TSPLIB.
- Best – best solution found by the algorithm.
- Relative error (RE) - indicates how close the solution is to the best-known solution and calculated by:

$$\text{RE} = \frac{\text{best solution} - \text{best known solution}}{\text{best known solution}} \times 100 \quad 5.2$$

- CPU time – time, in seconds, when the best solution is found.

5.3 Termination Criterion

The proposed algorithm terminates when a specific number of colonies is reached. In this research, the number of colonies is set to be 15% of the problem size. In addition, the following conditions are applied when running the experiments:

- Time limit : The limit on the CPU time spent solving before terminating a search is 150 seconds.
- Solution limit : The limit on the number of feasible solutions found before terminating a search is set to 25.

5.4 Factors Influencing the Performance of the Search Strategy

Factors that were considered to improve the overall performance of the proposed algorithm are the heuristic approach used to produce the initial solution, the bound strengthening, the number

Benchmark problems	Best-known solution	MACO				MACO with NN2-opt			
		Best	RE	CPU time	Number of colonies	Best	RE	CPU time	Number of colonies
berlin52	7542	**7775**	3.09	**96.99**	8	7775	3.09	205.7	7
eil76	538	567	5.39	467.1	10	**538**	0.00	**195.51**	6
pr76	108159	109043	0.82	1104.3	7	**109043**	0.82	**589.15**	8
kroa100	21282	21981	3.28	5776.72	13	**21282**	0.00	**2983.14**	7
krob100	22141	22211	0.32	5369.05	15	**22199**	0.26	**2736.98**	8
pr136	96772	96772	0.00	17226.5	11	**96772**	0.00	**16145**	11
u159	42080	**42080**	0.00	**2011**	2	42080	0.00	10793.7	8
pr226	80369	80369	0.00	101545	18	**80369**	0.00	**27147.2**	6

Table 5.1: Effect of different heuristics applied in the first colony on STSP benchmark problems

Benchmark problems	Best-known solution	MACO				MACO with NN2-opt			
		Best	RE	CPU time	Number of colonies	Best	RE	CPU time	Number of colonies
br17	39	66	69.23	0.18	1	**42**	7.69	**0**	1
ftv33	1286	1369	6.45	3.931	5	**1340**	4.20	**1.576**	3
ftv47	1776	**1825**	2.76	63.212	7	1907	7.38	37.643	7
ft70	38673	39017	0.89	267.17	7	**39015**	0.88	530.037	9
ftv170	2755	**2755**	0.00	15484.3	14	2764	0.33	4418.42	13
rbg323	1326	1920	44.80	15645.1	16	**1680**	26.70	**0**	1

Table 5.2: Effect of different heuristics applied in the first colony on ATSP benchmark problems

of variables fixed in the tour construction phase and the variable-fixing rules.

5.4.1 Impact of Different Solution Approach Used in the First Colony

In the proposed algorithm, the best solution found in the first colony is set as the upper bound for the objective function. This value will be updated immediately if a better solution is found. Ever since the initial solution will become the upper bound for the objective function, a good initial solution would be an important factor for achieving good results.

To investigate the effect of a different approach used in the first colony to produce the initial solution, a simple Nearest-neighbour with 2-opt (NN2-opt) heuristic is used as a comparison. Table 5.1 and Table 5.2 show the comparative results conducted on 8 STSP and 6 ATSP benchmark problems. The best solution for each instance is detailed in bold.

As can be seen in Table 5.1, the MACO with NN2-opt heuristic produces better quality solutions than the MACO in 37.5% of the instances (3 out of 8) and finds the same best solutions with the MACO in 62.5% of the eight instances including berlin52, pr76, pr136, u159 and pr226. Further, in most of the instances, the MACO with NN2-opt heuristic demonstrates lower CPU time and relative error than the MACO. Also, when both heuristics found the same best solution, the CPU time and number of colonies for the MACO with NN2-opt are better than the MACO heuristic.

Likewise, Table 5.2 shows that the MACO with NN2-opt heuristic obtains better quality solutions in 66.7% of the ATSP instances (4 out of 6) while the MACO finds better quality solutions in 33.3% of the six instances. Also, the average relative error and CPU time for the MACO with NN2-opt is less than the MACO for most of the benchmark problems.

In general, the numerical results in Table 5.1 and Table 5.2 illustrated that the MACO with NN2-opt heuristic produced better quality solutions and CPU time than the MACO for both symmetric and asymmetric TSP benchmark problems. Hence, as expected, a different heuristic approach used to generate the initial solution can affect the overall performance of the proposed algorithm. In particular, the MACO with NN2-opt heuristic adopted in the first colony has produced a better initial solution and provides good upper bounds which have contributed to positive results on the overall performance of the proposed algorithm.

5.4.2 Impact of Bound Strengthening

Another strategy executed for improving the effectiveness and efficiency of the proposed algorithm is to strengthen the upper bound of the objective function. Upper bounds play a significant role in improving the convergence rate of an algorithm by allowing the fathoming of nodes whose lower bound is greater than the smallest upper bound and therefore reducing the final size of the B&B tree. Moreover, tightening the upper bounds can significantly reduce the solution space due to the combinatorial nature of the problem. An upper bound on the solution of a given node can be obtained in several ways. In the proposed algorithm, the best-so-far solution is used to update the value of the upper bound. To further understand the impact of the upper bound on the performance of the MACO, two different values were considered for the upper bounds of the objective function:

- UBI : The best-so-far solution
- UBII : 1.1 of the best-so-far solution

Table 5.3 illustrates the results of running the proposed algorithm on 8 symmetric TSP benchmark problems with different upper bound values imposed on the objective function. The experimental results showed that UBI and UBII have tremendously reduced the CPU time for most of the benchmark problems. As shown in Table 5.3, the UBI obtains better quality solutions in 25% of the instances (2 out of 8) and worse quality solutions in 37.5% of the instances (3 out of 8) while the UBII and 'No Bound' find better quality solutions in 37.5% of the instances (3 out of 8) and worse solutions in 12.5% of the instance (1 out of 8), respectively.

Nevertheless, all the three conditions found the same best solution to 2 instances which are pr136 and u159. For pr136 instance, the 'No Bound' obtained the best solution with the minimum CPU time followed by the UBI and UBII with respectively, 26% and 75% more CPU time than the first condition. Contrarily, for u159 instance, the UBII found the best solution with the minimum CPU time while the UBI and 'No Bound' recorded 56% and 70% more CPU time than the UBII, respectively. In addition, the UBII found the optimal solutions to u159 and pr226 instances with great CPU time.

On the other hand, Table 5.4 shows that for the 6 instances, the UBI finds better quality solution in 50% of the instances (3 out of 6) and in 16.7% of the instances (1 out of 6) for the UBII and 'No Bound', respectively. For br17 instance, all the 3 cases found the same best solution but the minimum CPU time is reported for the UBI.

Benchmark problems	Best-known solution	No Bound				UB I				UB II			
		Best	RE	CPU time	No. of colonies	Best	RE	CPU time	No. of colonies	Best	RE	CPU time	No. of colonies
berlin52	7542	**7775**	3.09	96.994	8	7775	3.09	170.236	8	7792	3.31	187.513	7
eil76	538	567	5.39	467.102	10	**538**	0.00	50.048	10	538	0.00	187.538	8
pr76	108159	109043	0.82	1104.3	7	**108159**	0.00	1333.25	6	109043	0.82	131.47	3
kroa100	21282	21981	3.28	5776.72	13	22205	4.34	1136.94	7	**21345**	0.30	3023.39	12
krob100	22141	**22211**	0.32	5369.05	15	22396	1.15	7405.34	15	22239	0.44	6023.63	11
pr136	96772	**96772**	0.00	17226.5	11	96772	0.00	21664.2	15	96772	0.00	30081.3	20
u159	42080	42080	0.00	2011	2	42080	0.00	883.718	2	**42080**	0.00	595.941	2
pr226	80369	80369	0.00	101545	18	80769	0.50	91583.7	16	**80369**	0.00	30019	6

Table 5.3: STSP: Impact of different upper bound values on the efficiency of the proposed algorithm

Benchmark problems	Best - known solution	No Bound				UB I				UB II			
		Best	RE	CPU time	No. of colonies	Best	RE	CPU time	No. of colonies	Best	RE	CPU time	No. of colonies
br17	39	66	69.23	0.18	1	**66**	69.23	0.057	1	66	69.23	0.091	1
ftv33	1286	1369	6.45	3.931	5	**1324**	2.95	2.788	5	1388	7.93	2.572	4
ftv47	1776	1825	2.76	63.212	7	**1784**	0.45	31.03	7	1790	0.79	38.942	7
ft70	38673	39017	0.89	267.17	7	**38808**	0.35	39.667	9	38967	0.76	135.945	10
ftv170	2755	2755	0.00	15484.3	14	2825	2.54	513.242	6	**2755**	0.00	1122.98	5
rbg323	1326	**1920**	44.80	15645.1	16	2980	124.74	567.492	4	2399	80.92	247.24	2

Table 5.4: ATSP: Impact of different upper bound values on the efficiency of the proposed algorithm

Thus, it can be concluded that having a sharp upper bound on the objective function reduces the CPU time while not affecting the quality of the solutions. According to the numerical results in Table 5.3 and Table 5.4, the ideal upper bound for the STSP is the UBII (1.1 times of the best-so-far solution) while the best upper bound for the ATSP is UBI (the best-so-far solution).

5.4.3 Number of Variables Fixed in the Tour Construction Phase

As mentioned in Chapter 3, the number of variables fixed in the tour construction phase is 50% of the problem size (Case I). To investigate the impact of this value on the overall performance of the proposed algorithm, two more conditions were considered:

- Case 0 : 25% of the problem size
- Case II : 75% of the problem size

Table 5.5 and Table 5.6 , respectively, display the experimental results for Case 0, Case I and Case II on symmetric and asymmetric TSP instances.

As presented in Table 5.5, the benchmark problems were solved to optimality in 87.5% of the instances (7 out of 8) for Case 0 , in 37.5% in relation to Case I (3 out of 8) and in 12.5% in relation to Case II (1 out of 8). The average relative error recorded for Case 0 is 0.6%, 1.61% for Case I and 27.37% for Case II. However, in spite of the solution quality, Case II displayed the lowest average CPU time followed by Case I and Case 0. Also, Table 5.5 shows that the best solutions for the two largest benchmark problems considered which are u159 and pr226 instances were obtained for Case 1.

On the other hand, the experimental results in Table 5.6 illustrate that the best solutions for the ATSP were found when the number of variables fixed in the tour construction phase is 25% of the problem size. For all the instances, Case 0 found the optimal solutions in 50% of instances (3 out of 6) with an average relative error of 5.25%. The average relative error reported for Case I and Case II are 20.7% and 36.5% respectively.

For instance, less number of variables fixed in the tour construction phase would lead to a better solution quality for both symmetric and asymmetric TSP. However, if the main consideration is to improve the effectiveness of the proposed algorithm, Case I would be a better option for a good trade-off between the solution quality and CPU time.

Benchmark problems	Best known solution	Number of variables fixed											
		25%				50%				75%			
		Best	RE	CPU time	Number of colonies	Best	RE	CPU time	Number of colonies	Best	RE	CPU time	Number of colonies
berlin52	7542	**7542**	0	64.273	3	7775	3.09	96.994	8	10413	38.07	177.078	8
eil76	538	**538**	0	162.542	3	567	5.39	467.102	10	732	36.06	62.913	11
pr76	108159	**108159**	0	566.38	2	109043	0.82	1104.3	7	134101	23.99	158.673	4
kroa100	21282	**21282**	0	3949.17	4	21981	3.28	5776.72	13	31707	48.99	3882.45	14
krob100	22141	**22141**	0	3194.64	3	22211	0.32	5369.05	15	31185	40.85	5303.86	15
pr136	96772	**96772**	0	14578.5	5	96772	0.00	17226.5	11	120817	24.85	19911.5	19
u159	42080	42080	0	6188.1	2	**42080**	0.00	2011	2	42080	0.00	2174.87	3
pr226	80369	84230	4.80	139762	21	**80369**	0.00	101545	18	85338	6.18	74314.1	28

Table 5.5: STSP: Variants percentage of variables fixed

Benchmark problems	Best known solution	Number of variables fixed											
		25%				50%				75%			
		Best	RE	CPU time	Number of colonies	Best	RE	CPU time	Number of colonies	Best	RE	CPU time	Number of colonies
br17	39	**40**	2.56	2.677	3	66	69.23	0.18	1	57	46.15	0.318	3
ftv33	1286	**1286**	0.00	17.295	4	1369	6.45	3.931	5	1822	41.68	1.343	4
ftv47	1776	**1776**	0.00	18.358	3	1825	2.76	63.212	7	2505	41.05	24.705	6
ft70	38673	**38707**	0.09	79.127	3	39017	0.89	267.17	7	41600	7.57	97.765	9
ftv170	2755	**2755**	0.00	3691.55	2	2755	0.00	15484.3	14	3044	10.49	13345.5	25
rbg323	1326	**1710**	28.96	181801	45	1920	44.80	15645.1	16	2284	72.25	4567.95	29

Table 5.6: ATSP: Variants percentage of variables fixed

5.4.4 Variable-Fixing Rules

According to a survey by Atamturk and Savelsbergh (Atamturk & Savelsbergh, 2005), variable-fixing procedures are used in many of linear-relaxation-based solvers. In this research, the variable-fixing rules are considered as an essential component of the proposed algorithm due to its significant influence on the algorithm computational time. Indeed, a proper implementation of this procedure is critical to speed up the resolution of the successive decision problems.

In general, the 'fixing' operation means that a variable gets permanently assigned to a constant value. However, it was not clear that the solver used in this research could detect all the redundancies introduced by the variable-fixing rules in the tour construction phase and thus has caused the high computational times. As a consequence, forcing the variable-fixing explicitly using the *fix*, *drop*, *unfix* and *restore* commands may eliminate those redundant constraints. By doing this, only the immediate relevant variables and constraints are sent to the solver. Hence, it helps to enhance the computational performance of the symmetric and asymmetric TSP.

The variable fixing-rules use for this proposed algorithm are summarised as follows:

For any unfixed variable x_{ij} :

(a) if i is the initial city and $x_{ij} = 1$, then:

i) u_i can be assigned a rank of 1;

ii) x_{ik} can be fixed to 0 for all $k \in V, k \neq i$ and $k \neq j$.

(b) if $u_i \neq 0$ and $u_j \neq (n-1)$ then ignore the out-degree and in-degree constraints as in equation 1.8 and 1.9, respectively. Thus;

i) let $u_j = (u_i + 1)$;

ii) let x_{ij} equal to 1 and x_{ik} to 0 for all $k \in V, k \neq i$ and $k \neq j$.

5.4.4.1 An Illustrative Example

The general procedure of the variable-fixing rules implemented in the proposed algorithm is shown in Figure 5.1- Figure 5.5. The straight line arrow represents $\mathrm{x[k, k1]} = 1$ while dashed arrow represent $\mathrm{x[k, k1]} = 0$. Consider a 6-city instance with $V = \{1,2,3,4,5,6\}$:

Step 1 : In Figure 5.1, if the first node selected is $k = 2$ and the shortest distance from node 2 is to node 3; fix the position of the first node by letting $u[2] = 1$ and $x[2,3] = 1$. Following the rules stated in (a) ii) above, let $x[k, k1] = 0$ for all $k1 \neq 3$ and $k \neq k1$.

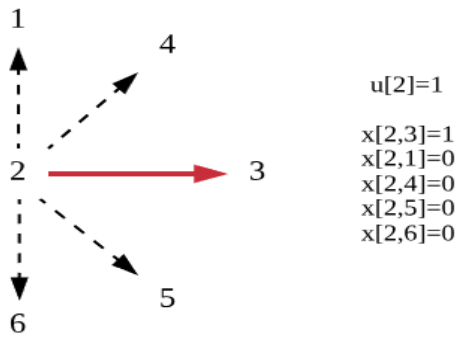


Figure 5.1: Fixing the position and out-degree arc of the first node

Step 2(a) : Repeat Step 1 for $k = 3$ and $\text{next}(k) = 1$. See Figure 5.2

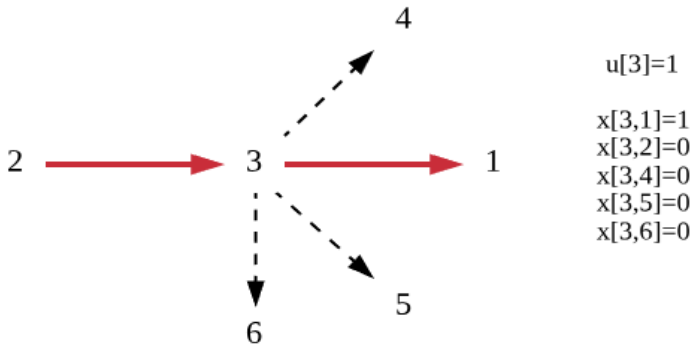


Figure 5.2: Fixing the position and out-degree arcs of the second node

Step 2(b) : When the position of the node considered is greater than 1, the in-degree constraints implies that $x[k1, k] = 0$ for all $k1 \neq 2$ and $k \neq k1$. See Figure 5.3.

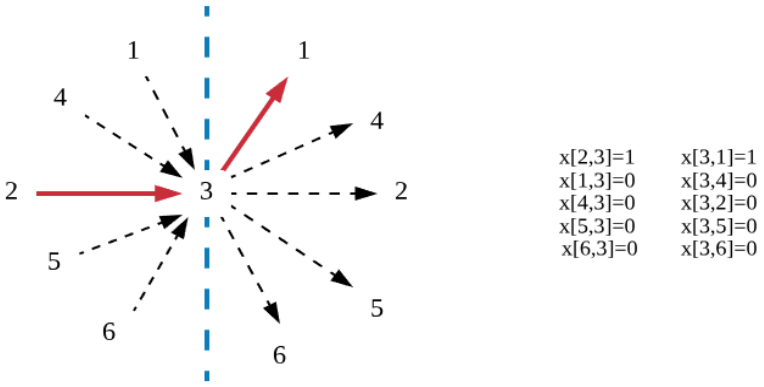


Figure 5.3: Fixing the position and in-degree arcs of the second node

Step 3 **:** Repeat Step 1-2 for all nodes excluding the last node. See Figure 5.4 and Figure 5.5.

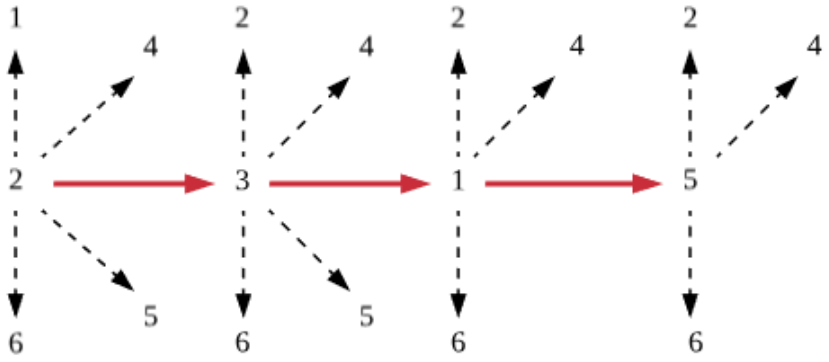


Figure 5.4: The nodes position and out-degree arcs of the fixed nodes

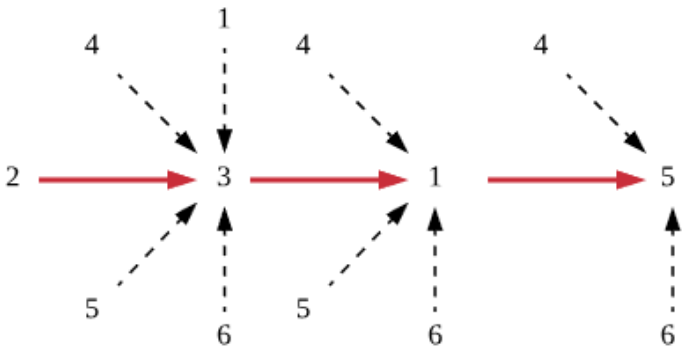


Figure 5.5: The in-degree arcs of the fixed nodes

Benchmark problems	Best-known solution	Without variable-fixing rules				With variable-fixing rules			
		Best	RE	CPU time	Number of colonies	Best	RE	CPU time	Number of colonies
berlin52	7542	7775	3.09	455.04	8	**7775**	3.09	96.994	8
eil76	538	**555**	3.16	1737.19	11	567	5.39	467.102	10
pr76	108159	109791	1.51	7674.90	11	**109043**	0.82	1104.3	7
kroa100	21282	**21345**	0.30	25704.60	14	21981	3.28	5776.72	13
krob100	22141	22580	1.98	24951.70	15	**22211**	0.32	5369.05	15
pr136	96772	96772	0	52497.3	14	**96772**	0.00	17226.5	11
u159	42080	42080	0	7630.5	2	**42080**	0.00	2011	2
pr226	80369	90635	12.77	227856	34	**80369**	0.00	101545	18

Table 5.7: STSP: Impact of variable-fixing procedure on the performance of the proposed algorithm

Benchmark problems	Best-known solution	Without variable-fixing rules				With variable-fixing rules			
		Best	RE	CPU time	Number of colonies	Best	RE	CPU time	Number of colonies
br17	39	66	69.23	1.806	1	**66**	69.23	0.18	1
ftv33	1286	1369	6.45	50.7	5	**1369**	6.45	3.931	5
ftv47	1776	2076	16.89	223.024	7	**1825**	2.76	63.212	7
ft70	38673	39017	0.89	4423.34	7	**39017**	0.89	267.17	7
ftv170	2755	2882	4.61	43949.2	24	**2755**	0.00	15484.3	14
rbg323	1326	1969	48.49	189114	42	**1920**	44.80	15645.1	16

Table 5.8: ATSP: Impact of variable-fixing procedure on the performance of the proposed algorithm

5.4.4.2 Computational Results and Numerical Analysis

Table 5.7 shows that the proposed algorithm with the variable-fixing rules obtains better quality solutions in 37.5% of the benchmark problems (3 out of 8) while the proposed algorithm without the variable-fixing rules found better quality solutions in 25% of the benchmark problems (2 out of 8). Although both approaches found the same best solutions for berlin52, pr136 and u159 instances, the CPU times reported for the proposed algorithm with the variable-fixing rules is less than half of the CPU time reported for the other algorithm. Besides, the average CPU time for the proposed algorithm with the variable-fixing rules has improved by 61.6% when compared to the average CPU time for the proposed algorithm without the variable-fixing rules.

Likewise, as presented in Table 5.8, the results obtain by the proposed algorithm with the variable-fixing rules are more efficient and effective than the proposed algorithm without the variable-fixing rules. The average relative error for the proposed algorithm with the variable-fixing rules and without variable-fixing rules are 20.7% and 24.43% respectively. In conclusion, in spite of the solution quality, the variable-fixing rule implemented in the proposed algorithm has successfully reduced the average CPU time by 86.7%.

5.5 Conclusion

The experimental results have shown that each factor has a different impact on the overall performance of the proposed algorithm. In particular, a good initial solution produced in the first colony leads to a better quality solution while strengthening the bounds help to reduce the CPU time and contributes to fast convergence. Besides, fixing a small number of nodes in the tour construction phase could produce better quality solutions but with higher CPU time. In most cases, the less number of nodes fixed could improve the effectiveness of the proposed algorithm while a higher number of nodes fixed could improve the efficiency of the proposed algorithm. Also, applying the variable-fixing rules in the tour construction phase could significantly improve the efficiency of the proposed algorithm.

Therefore, the empirical results suggest the following strategies to enhance the performance of the proposed algorithm for solving the TSPs:

- The NN2-opt heuristic will be used as a tour construction approach in the first colony of the proposed algorithm for both symmetric and asymmetric TSPs.
- The ideal upper bound for the proposed algorithm are :

  Symmetric TSP : upper bound = $1.1(\text{best-so-far solution})$

  Asymmetric TSP : upper bound = best-so-far solution
- A number of variables fixed in the solution construction phase are:

  Symmetric TSP : 50% of the problem size

  Asymmetric TSP : 25% of the problem size.

Although the solver could detect all the redundancies, the experimental results demonstrated that it is best to remove all the possible redundancies in the algorithm which also helps to speed up the CPU time.

In the next chapter, all experiments of the proposed algorithm will be based on the above settings, except when indicated differently.

Chapter 6

Experimental Results for the Proposed Algorithm

This chapter provides a comprehensive comparison of the proposed algorithm with other algorithms available in the literature. Each algorithm is compared according to the types of TSP benchmark problems solved such as symmetric or asymmetric.

6.1 Test Instances and Termination Criterion

A total of 33 TSP benchmark problems consisting of 15 ATSP and 18 STSP instances are used to evaluate both the effectiveness and efficiency of the proposed algorithm. For the asymmetric TSP, the size of the instance is shown by the numerical suffix in the dataset name except for the problem class 'ftv' for which the size of instances is equal to *number of cities + 1*. For example, for ftv33, the problem size is 34 while for ftv47, the problem size is 48.

The terminating condition is the number of colonies (iterations). The maximum number of colonies for each instance is listed in Table 6.1.

6.2 Computational Results and Numerical Analysis for the Proposed Algorithm Applied to the Symmetric TSP

Two kinds of experiments were carried out to evaluate the performance of the proposed algorithm (MACO). The first experiment examines MACO's performance on symmetric TSP instances using a set of ACO parameter values as suggested in Chapter 4 against a set of ACO parameter values recommended by (Dorigo, Maniezzo, & Colorni, 1996). For simplification, the latter algorithm is called MACOd. The second set of experiments compares the performance of the MACO with other studies in the literature applied to symmetric TSP instances. For each benchmark problem, the results of the MACO algorithm are reported as best, relative error (RE), number of colonies and CPU time.

In all the following tables, column 'best-known solution' denotes the best tour length as reported in the TSPLIB standard library, column 'best' denotes the best solution found by each algorithm and column 'RE' reveals the percentage deviation of the best solution (best) in

comparison to the best-known solution. The '-' sign indicates the result was not available in the respective study or source. Most of all, it should be noted that the comparisons are only based on the quality of the best solutions on comparable results. Therefore, a better algorithm is considered to be those whose values of best and relative error (RE) are smaller than those of the other algorithms.

No	Instance	Max no. of colonies
1	eil51	8
2	berlin52	8
3	st70	11
4	eil76	11
5	pr76	11
6	kroa100	15
7	krob100	15
8	eil101	15
9	lin105	16
10	pr124	19
11	pr136	20
12	ch150	23
13	u159	24
14	d198	30
15	kroa200	30
16	pr226	34
17	lin318	48

No	Instance	Max no. of colonies
18	pcb442	66
19	br17	3
20	ftv33	5
21	ftv35	5
22	ftv38	6
23	p43	6
24	ftv44	7
25	ftv47	7
26	ry48p	7
27	ft53	8
28	ftv55	8
29	ftv64	10
30	ftv70	11
31	ft70	11
32	kro124p	15
33	ftv170	26

Table 6.1: Maximum number of colonies for the symmetric and asymmetric TSP instances

Benchmark Problems	Best-known solution	Best	RE	No. of Colonies	CPU time
berlin52	7542	7916	4.96	6	168.656
st70	675	683	1.19	5	94.809
eil76	538	538	0	5	35.001
pr76	108159	108159	0	7	698.82
kroa100	21282	21282	0	4	344.34
krob100	22141	22199	0.26	12	7543.09
u159	42080	42080	0	8	7703.1
pr226	80369	80369	0	7	33704.5

Table 6.2: The performance of the MACO on 8 STSP instances

In the first experiments, the results obtained by the MACO algorithm are given in Table 6.2 and the results obtained by the MACOd are given in Table 6.3. Results of these experiments are comparatively provided in Table 6.5 where the best results are emphasised in bold.

Table 6.2 shows that the MACO finds the best-known solution to all benchmark problems except for berlin52, st70 and krob100 while the MACOd finds the optimal solution to 1 instance as depicted in Table 6.3. However, for berlin52, st70 and krob100, if the number of nodes fixed in the tour construction phase is reduced from 50% to 25% of the problem size, those instances were solved to optimality as shown in Table 6.4.

Benchmark Problems	Best-known solution	Best	RE	No. of Colonies	CPU time
berlin52	7542	7542	0	3	20.29
st70	675	696	3.11	3	684.33
eil76	538	546	1.49	3	57.24
pr76	108159	114656	6.01	3	2796.42
kroa100	21282	22152	4.09	2	273.2
krob100	22141	22971	3.75	3	2117.99
u159	42080	44597	5.98	5	5567.22
pr226	80369	81063	0.86	2	7745.59

Table 6.3: The performance of the MACOd on 8 STSP instances

Benchmark Problems	Best-known solution	Best	RE	No. of Colonies	CPU time
berlin52	7542	7542	0	2	174.48
st70	675	675	0	3	977.90
krob100	22141	22141	0	10	15672.6

Table 6.4: The performance of the MACO when the variables fixed is 25% of the problem size

Further, as can be seen in Table 6.5, for the 8 STSP benchmark problems, the MACO finds the optimal solutions to 5 instances while the MACOd finds the optimal solution to 1 instance. Although the MACOd found the optimal solution for berlin52, the algorithm yields worse solutions for the other 7 instances. Besides, the computational results displayed in Table 6.5 also show that the MACOd easily gets trapped in local optima thus reported less number of colonies and CPU time. Likewise, the relative errors of the MACOd are much higher than the MACO, indicating that the MACO has a better search capability than the MACOd.

Benchmark Problems	Best-known solution	Algorithm	Best	RE	No. of colonies	CPU time
berlin52	7542	MACOd	**7542**	0	3	20.29
		MACO	7916	4.96	6	168.656
st70	675	MACOd	696	3.11	3	684.33
		MACO	**683**	1.19	5	94.809
eil76	538	MACOd	546	1.49	3	57.24
		MACO	**538**	0	5	35.001
pr76	108159	MACOd	114656	6.01	3	2796.42
		MACO	**108159**	0	7	698.82
kroa100	21282	MACOd	22152	4.09	2	273.2
		MACO	**21282**	0	4	344.34
krob100	22141	MACOd	22971	3.75	3	2117.99
		MACO	**22199**	0.26	12	7543.09
u159	42080	MACOd	44597	5.98	5	5567.22
		MACO	**42080**	0	8	7703.1
pr226	80369	MACOd	81063	0.86	2	7745.59
		MACO	**80369**	0	7	33704.5

Table 6.5: Comparative experimental results

In the second experiments, the comparisons are made with 9 algorithms presented in the literature which are the RABNET-TSP (A.S.Masutti & Castro, 2009), GSA-ACS-PSO (Chen & Chien, 2011), GA-PSO-ACO (Deng, et al., 2012), SEE (Tuba & Jovanovic, 2013), ACO-ABC (Gunduz, Kiran, & Ozceylan, 2015), PSO-ACO-3Opt (Mahi, Baykan, & Kodaz, 2015), SSA (Wang, Lin, Zhong, & Zhang, 2016), REACSGA (Yousefikhoshbakht, Malekzadeh, & Sedighpour, 2016) and AEAS (Mohsen, 2016). The experimental results for these algorithms are shown in Table 6.6 while the comparative analysis is summarised in Table 6.7 and Figure 6.1, respectively.

As can be seen in Table 6.7, the proposed algorithm has performed better than the ACO-ABC in 75% of the instances (6 out of 8), in 66.67% in relation to the GA-PSO-ACO (8 out of 12), in 60% of the instances in relation to RABNET-TSP (6 out of 10) and in 44.44% in relation to PSO-ACO-3OPT (4 out of 9). Furthermore, the proposed algorithm yielded equal solutions to the SEE in 27.27% of the instances (3 out of 11). Besides, although the proposed algorithm obtained fewer numbers of best solutions than the GSA-ACS-PSO, SSA, REACSGA and AEAS, the number of the optimal solutions found for all cases are more than 55%.

Benchmark Problems	Best-known solution	MACO		RABNET-TSP		SEE		GA-PSO-ACO		ACO-ABC	
		Best	RE	Best	RE	Best	RE	Best	RE	Best	RE
eil51	426	432	1.41	427	0.23	427	0.23	**426**	**0**	431.74	1.35
berlin52	7542	7916	4.96	**7542**	**0**	**7542**	**0**	7544.37	0.03	7544.37	0.03
st70	675	683	1.19	-	-	**675**	**0**	679.6	0.68	687.24	1.81
eil76	538	**538**	**0**	541	0.56	**538**	**0**	545.39	1.37	551.07	2.43
pr76	108159	**108159**	**0**	-	-	108358	0.18	109206	0.97	113798.56	5.21
kroa100	21282	**21282**	**0**	21333	0.24	**21282**	**0**	-	-	22122.75	3.95
krob100	22141	22199	0.26	22343	0.91	-	-	-	-	-	-
eil101	629	**629**	**0**	638	1.43	-	-	633.07	0.65	672.71	6.95
lin105	14379	**14379**	**0**	**14379**	**0**	**14379**	**0**	14397	0.13	-	-
pr124	59030	**59030**	**0**	-	-	**59030**	**0**	59051	0.04	-	-
pr136	96772	**96772**	**0**	-	-	96781	0.01	-	-	-	-
ch150	6528	**6528**	**0**	6602	1.13	-	-	-	-	6641.69	1.74
u159	42080	**42080**	**0**	-	-	**42080**	**0**	42395	0.75	-	-
kroa200	29368	**29368**	**0**	29600	0.79	29490	0.42	29731	1.24	-	-
lin318	42029	43296	3.01	42834	1.92	-	-	42633	1.44	-	-
pcb442	50778	**50778**	**0**	-	-	-	-	51414	1.25	-	-

Table 6.6: A comparison of the proposed MACO algorithm with RABNET-TSP, GSA-ACS-PSO, GA-PSO-ACO, SEE, ACO-ABC, PSO-ACO-3OPT, SSA, REACSGA and AEAS according to the best solutions and relative errors.

Benchmark Problems	Best- known solution	GSA-ACS-PSO		PSO-ACO-3OPT		SSA		REACSGA		AEAS	
		Best	RE	Best	RE	Best	RE	Best	RE	Best	RE
eil51	426	427	0.23	**426**	**0**	**426**	**0**	**426**	**0**	**426**	**0**
berlin52	7542	**7542**	**0**	**7542**	**0**	**7542**	**0**	**7542**	**0**	**7542**	**0**
st70	675	-	-	676	0.15	**675**	**0**	-	-	-	-
eil76	538	**538**	**0**	**538**	**0**	**538**	**0**	**538**	**0**	**538**	**0**
pr76	108159	-	-	-	-	**108159**	**0**	-	-	-	-
kroa100	21282	**21282**	**0**	21301	0.09	**21282**	**0**	**21282**	**0**	**21282**	**0**
krob100	22141	**22141**	**0**	-	-	**22141**	**0**	**22141**	**0**	**22141**	**0**
eil101	629	630	0.16	631	0.32	**629**	**0**	**629**	**0**	**629**	**0**
lin105	14379	**14379**	**0**	**14379**	**0**	**14379**	**0**	**14379**	**0**	**14379**	**0**
pr124	59030	-	-	-	-	**59030**	**0**	-	-	-	-
pr136	96772	-	-	-	-	-	-	-	-	-	-
ch150	6528	**6528**	**0**	6538	0.15	**6528**	**0**	-	-	**6528**	**0**
u159	42080	-	-	-	-	**42080**	**0**	-	-	-	-
kroa200	29368	29383	0.05	29468	0.34	**29368**	**0**	**29368**	**0**	**29368**	**0**
lin318	42029	42489	1.09	-	-	42081	0.12	42543	1.22	**42029**	**0**
pcb442	50778	-	-	-	-	50820	0.08	-	-	-	-

Table 6.6: A comparison of the proposed MACO algorithm with RABNET-TSP, GSA-ACS-PSO, GA-PSO-ACO, SEE, ACO-ABC, PSO-ACO-3OPT, SSA, REACSGA and AEAS according to the best solutions and relative errors.

Approach	No. of instances	Optimal Solution		Best Solution		Worse solution		Equal solution	
		No.	%	No.	%		No.	%	No.
MACO	10	6	**60.00**	6	**60**	3	**30.00**	1	10
RABNET-TSP		2	20.00	3	30	7	70.00		
MACO	11	8	**72.73**	3	**27.27**	3	**27.27**	5	45.45
SEE		7	63.64	3	**27.27**	3	**27.27**		
MACO	12	8	**66.67**	8	**66.67**	4	**33.33**	0	0
GA-PSO-ACO		1	8.33	4	33.33	8	66.67		
MACO	8	5	**62.5**	6	**75**	2	**25**	0	0
ACO-ABC		0	0	2	25	6	75		
MACO	10	6	**60**	2	20	4	40	4	40
GSA-ACS-PSO		6	**60**	4	**40**	2	**20**		
MACO	9	6	**66.67**	4	**44.44**	3	**33.33**	2	22.22
PSO-ACO-3OPT		4	44.44	3	33.33	4	44.44		
MACO	15	10	66.67	1	6.67	5	33.33	9	60
SSA		13	**86.67**	5	**33.33**	1	**6.67**		
MACO	9	5	55.56	0	0	4	44.44	5	55.56
REACSGA		8	**88.89**	4	**44.44**	0	**0**		
MACO	10	6	60	0	0	4	40	6	60
AEAS		10	**100**	4	**40**	0	**0**		

Table 6.7: The overall performance comparison of the MACO with RABNET-TSP, SEE, SSA ACO-ABC, GA-PSO-ACO, GSA-ACS-PSO, PSO-ACO-3OPT, REACSGA and AEAS on the STSP instances.

According to Figure 6.1, the proposed MACO algorithm is better than the RABNET-TSP, GA-PSO-ACO, ACO-ABC and PSO-ACO-3OPT.

In summary, the numerical results illustrated that the proposed MACO algorithm is competitive when compared to the other existing algorithms with accuracy more than 95% in all of the benchmark problems considered. On top of that, the computational results also demonstrated that the proposed algorithm provides good performance on large-scale instances in reasonable computation time.

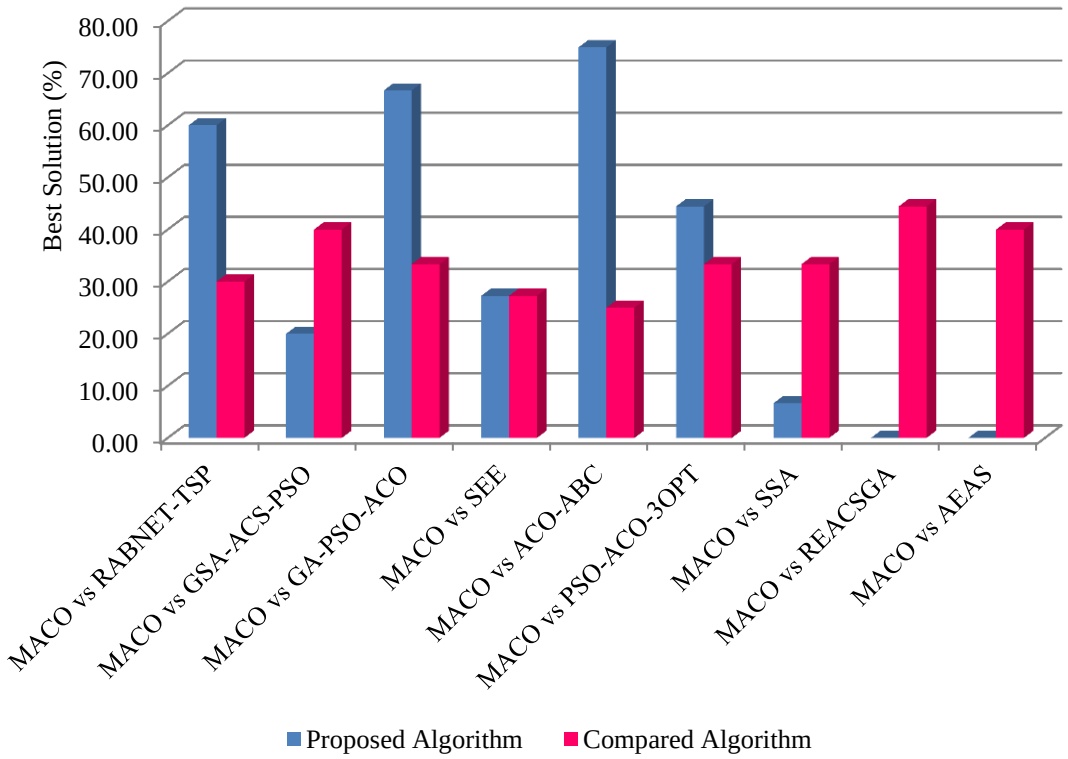


Figure 6.1: Percentage of the best solutions found by each algorithm applied to the STSP instances.

6.3 Computational Results and Numerical Analysis of the Proposed Algorithm Applied to Asymmetric TSP

For this category, the comparison is made on the results obtained by the MACO and MACOd on 7 ATSP benchmark problems taken from the TSPLIB standard library. The results obtained by the MACO are displayed in Table 6.8 and the results obtained by MACOd are displayed in Table 6.9. Table 6.10 compares these results with the best results shown in bold.

As shown in Table 6.8, all the benchmark problems were solved to optimality within a reasonable CPU time. In contrast, only 14.3% of the benchmark problems were solved to optimality for the MACOd as shown in Table 6.9.

From the experimental results displayed in Table 6.10, for the 7 ATSP benchmark problems, the MACO once again outperformed the MACOd and found the optimal solutions to

all of the instances while the MACOd finds the optimal solution to only 1 instance. Although in most cases, the CPU times of the MACOd are lower than the MACO, the average relative error for the MACOd are higher than the MACO.

Benchmark Problems	Best-known solution	Best	RE	No. of colonies	CPU time
ftv33	1286	1286	0	2	2.05
ftv44	1613	1613	0	4	8.09
ry48p	14422	14422	0	4	173.41
ftv55	1608	1608	0	5	28.86
ftv70	1950	1950	0	6	549.51
kro124p	36230	36230	0	3	1713.42
ftv170	2755	2755	0	3	1852.44

Table 6.8: The performance of the MACO on ATSP instances

Benchmark Problems	Best-known solution	Best	RE	No. of colonies	CPU time
ftv33	1286	1286	0	2	3.50
ftv44	1613	1647	2.11	2	7.55
ry48p	14422	14617	1.35	6	62.20
ftv55	1608	1724	7.21	2	196.40
ftv70	1950	2105	7.95	2	60.15
kro124p	36230	38694	6.80	13	4039.7
ftv170	2755	2915	5.81	3	390.71

Table 6.9: The performance of the MACOd on ATSP instances

Further comparisons are made with 3 algorithms presented in the literature which are Guided Variable Neighbourhood Search (GVNS) by (Burke, Cowling, & Keuthen, 2001), Randomized Arbitrary Insertion (RAI) by (Brest & Zerovnik, 2005) and Improved Genetic Algorithm (IGA) by (Abdoun, Tajani, Abouchabaka, & Khatir, 2016) .

According to the experimental results in Table 6.11, Table 6.12 and Figure 6.2, the proposed MACO algorithm has outperformed the GVNS and IGA algorithms with respect to best solutions. As can be seen in Table 6.12, the proposed algorithm is more efficient when compared to the GVNS in 64.29% of the instances (9 out of 14), and in 93.33% in relation to the IGA (14 out of 15). Additionally, the computational results of the proposed algorithm and the RAI show that these algorithms have a close competition with RAI produced 3 better

solutions than the proposed algorithm while the proposed algorithm yielded 2 better solutions than the RAI. Though, for all cases, the numbers of the optimal solutions found by the proposed algorithm are between 64% and 67%.

Benchmark Problems	Best-known solution	Algorithm	Best	RE	No. of colonies	CPU time
ftv33	1286	MACOd	1286	0	2	3.5
		MACO	**1286**	0	2	2.05
ftv44	1613	MACOd	1647	2.11	2	7.55
		MACO	**1613**	0	5	8.09
ry48p	14422	MACOd	14617	1.35	6	62.2
		MACO	**14422**	0	5	173.41
ftv55	1608	MACOd	1724	7.21	2	196.4
		MACO	**1608**	0	3	28.86
ftv70	1950	MACOd	2105	7.95	2	60.15
		MACO	**1950**	0	6	549.51
kro124p	36230	MACOd	38694	6.80	13	4039.7
		MACO	**36230**	0	5	1713.42
ftv170	2755	MACOd	2915	5.81	3	390.71
		MACO	**2755**	0	4	1852.44

Table 6.10: Comparative experimental results on the ATSP benchmark problems

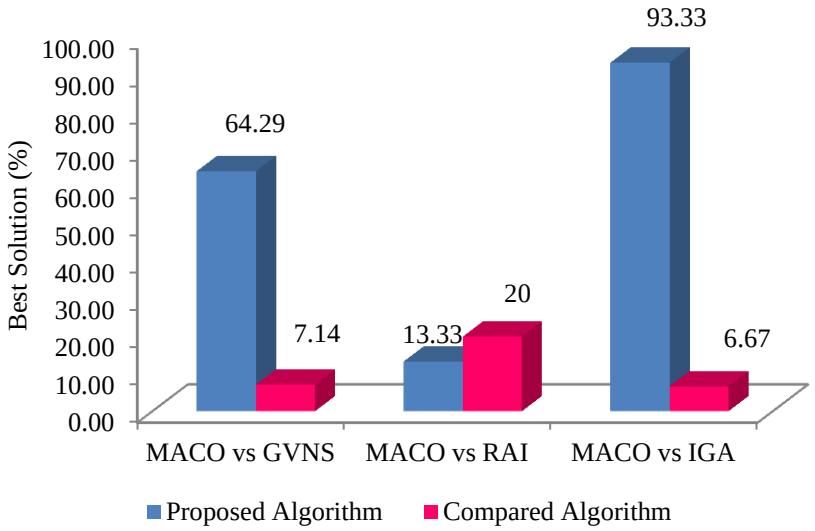


Figure 6.2: Percentage of the best solutions found by each algorithm applied to the ATSP instances.

Benchmark problems	Best-known solution	MACO		GVNS		RAI		IGA	
		Best	RE	Best	RE	Best	RE	Best	RE
br17	39	40	2.56	**39**	0	**39**	0	**39**	0
ftv33	1286	**1286**	0	1290	0.29	**1286**	0	2021	57.15
ftv35	1473	1475	0.14	1474	0.09	**1473**	0	2559	73.73
ftv38	1530	1532	0.13	1536	0.37	**1530**	0	2786	82.09
p43	5620	5628	0.14	5621	0.01	**5620**	0	5743	2.19
ftv44	1613	**1613**	0	1623	0.60	**1613**	0	3320	105.83
ftv47	1776	**1776**	0	1778	0.10	**1776**	0	3305	86.09
ry48p	14422	**14422**	0	14462	0.28	**14422**	0	22268	54.40
ft53	6905	**6905**	0	6913	0.12	**6905**	0	16032	132.18
ftv55	1608	**1608**	0	1609	0.08	**1608**	0	3742	132.71
ftv64	1839	**1839**	0	1847	0.44	**1839**	0	4849	163.68
ftv70	1950	**1950**	0	1966	0.82	**1950**	0	5604	187.38
ft70	38673	**38707**	0.09	38723	0.13	38850	0.47	56671	46.54
kro124p	36230	**36230**	0	-	-	36241	0.03	101284	179.56
ftv170	2755	**2755**	0	2805	1.83	**2755**	0	16982	516.41

Table 6.11: A comparison of the proposed MACO algorithm with GVNS, RAI and IGA according to the best solutions and relative errors.

Approach	No. of instances	Optimal Solution		Best Solution		Worse solution		Equal solution	
		No.	%	No.	%	No.	%	No.	%
MACO	14	9	**64.29**	9	**64.29**	1	**7.14**	0	0
GVNS		1	7.14	1	7.14	9	64.29		
MACO	15	10	66.67	2	13.33	4	26.67	9	60
RAI		13	**86.67**	3	**20.00**	2	**13.33**		
MACO	15	10	**66.67**	14	**93.33**	1	**6.67**	0	0
IGA		1	6.67	1	6.67	14	93.33		

Table 6.12: The overall performance comparison of the proposed MACO algorithm with MACO, RAI, IGA and GVNS on the ATSP instances.

6.4 Comparison with Other Algorithms Tested with Both Symmetric and Asymmetric TSP Benchmark Problems

In this section, the proposed algorithm is compared with other studies available in the literature applied to both symmetric and asymmetric TSP. The proposed algorithm was verified on a set of 30 benchmark problems with sizes ranging from 51 to 318 cities. Table 6.13 makes a comparison of the experimental results of the proposed algorithm with the ACS (Gambardella & Dorigo, 1996), MMAS (Stutzle & Hoos, 1997), African Buffalo Optimization (ABO) by (Odili & Kahar, 2016) and Improved BAT Algorithm (IBA) by (Osaba, Yang, Diaz, Lopez-Garcia, & Carballedo, 2016) with respect to the best solutions. Besides, the comparative analyses of these results are displayed in Table 6.14 and Figure 6.3.

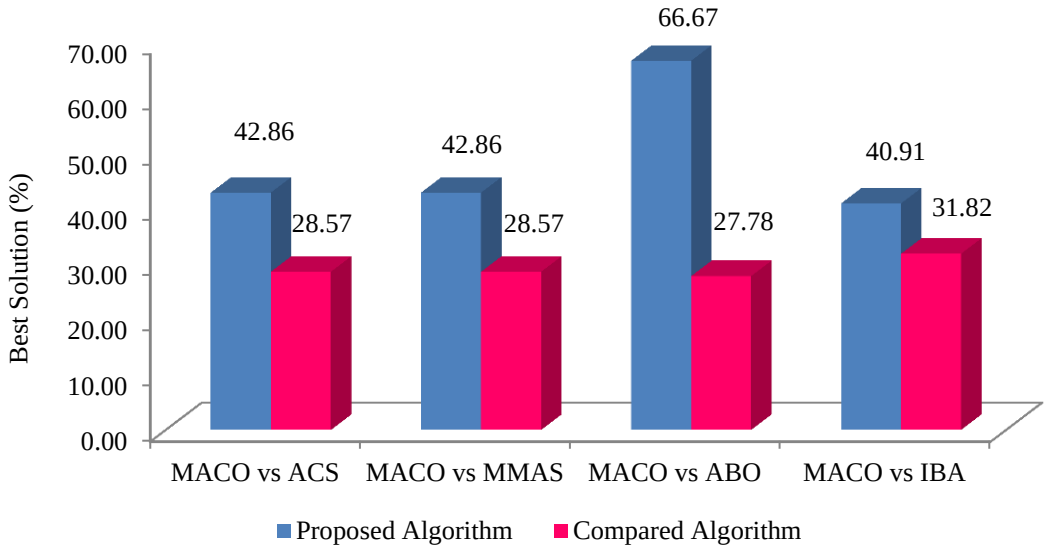


Figure 6.3: Percentage of the best solutions found by each algorithm applied to both STSP and ATSP

As can be seen in Figure 6.3, the proposed algorithm has outperformed the ACS, MMAS, ABO and IBA in terms of optimal solutions and best solution. In particular, according to Table 6.14, the proposed algorithm has yielded better results than the ACS and MMAS in 42.86% of the instances (3 out of 7), respectively, in 66.67% in relation to the ABO (12 out of 18), and in 40.91% in relation to the IBA (9 out of 22). On average, the numbers of optimal solutions obtained by the proposed algorithm are over 57% in each case.

Benchmark Problems	Best-known solution	MACO		ACS		MMAS		ABO		IBA	
		Best	RE	Best	RE	Best	RE	Best	RE	Best	RE
eil51	426	432	1.41	**426**	0	**426**	0	**426**	0	**426**	0
berlin52	7542	7916	4.96	-	-	-	-	**7542**	0	**7542**	0
st70	675	683	1.19	-	-	-	-	676	0.15	**675**	0
eil76	538	**538**	0	-	-	-	-	**538**	0	539	0.19
pr76	108159	**108159**	0	-	-	-	-	108167	0.01	-	-
kroa100	21282	**21282**	0	**21282**	0	**21282**	0	21311	0.14	**21282**	0
krob100	22141	22199	0.26	-	-	-	-	**22160**	0.09	-	-
eil101	629	**629**	0	-	-	-	-	640	1.75	634	0.79
lin105	14379	**14379**	0	-	-	-	-	14419	0.28	-	-
pr124	59030	**59030**	0	-	-	-	-	-	-	**59030**	0
pr136	96772	**96772**	0	-	-	-	-	-	-	97547	0.8
ch150	6528	**6528**	0	-	-	-	-	6532	0.06	-	-
d198	15780	15927	0.93	**15888**	0.68	15963	1.16	-	-	-	-
kroa200	29368	**29368**	0	-	-	-	-	29370	0.01	-	-
lin318	42029	43296	3.01	-	-	-	-	**42101**	0.17	-	-
br17	39	40	2.56	-	-	-	-	-	-	**39**	0
ftv33	1286	**1286**	0	-	-	-	-	-	-	**1286**	0
ftv35	1473	1475	0.14	-	-	-	-	-	-	**1473**	0
ftv38	1530	1532	0.13	-	-	-	-	-	-	**1530**	0
p43	5620	5628	0.14	-	-	-	-	5645	0.44	**5620**	0
ftv44	1613	**1613**	0	-	-	-	-	-	-	**1613**	0
ftv47	1776	**1776**	0	-	-	-	-	-	-	1796	1.13
ry48p	14422	**14422**	0	**14422**	0	**14422**	0	**14440**	0.12	**14422**	0
ft53	6905	**6905**	0	-	-	-	-	-	-	7001	1.39
ftv55	1608	**1608**	0	-	-	-	-	-	-	**1608**	0
ftv64	1839	**1839**	0	-	-	-	-	-	-	1879	2.18
ftv70	1950	**1950**	0	-	-	-	-	1955	0.26	2111	8.26
ft70	38673	38707	0.09	38781	0.28	**38690**	0.04	38753	0.21	39901	3.18
kro124p	36230	**36230**	0	36241	0.03	36416	0.51	36275	0.12	37538	3.61
ftv170	2755	**2755**	0	2774	0.69	2787	1.16	2795	1.45	-	-

Table 6.13: A comparison of the proposed MACO algorithm with ACS, MMAS, ABO and IBA according to the best solutions and relative errors.

Approach	No. of instances	Optimal Solution		Best Solution		Worse solution		Equal solution	
		No.	%	No.	%	No.	%	No.	%
MACO	7	4	**57.14**	3	**42.86**	2	**28.57**	2	28.57
ACS		3	42.86	2	28.57	3	42.86		
MACO	7	4	**57.14**	3	**42.86**	2	**28.57**	2	28.57
MMAS		3	42.86	2	28.57	3	42.86		
MACO	18	11	**61.11**	12	**66.67**	5	**27.78**	1	5.56
ABO		3	16.67	5	27.78	12	66.67		
MACO	22	14	**63.64**	9	**40.91**	7	**31.82**	6	27.27
IBA		13	59.09	7	31.82	9	40.91		

Table 6.14: The overall performance comparison of the proposed MACO algorithm with ACS, MMAS, ABO and IBA on the STSP and ATSP instances.

6.5 Numerical Analysis

According to the computational results, the numerical analysis indicates the following:

- The proposed algorithm provides good solution quality for both symmetric and asymmetric TSP benchmark problems with up to 442 cities. As illustrated in Figure 6.4, the percentage of the optimal solutions achieved by the proposed algorithm on 15 ATSP and 18 STSP problem instances are 67%, respectively. In particular, as shown in Table 6.6 and Table 6.13, the average relative error for both the STSP and ATSP are below 1% and the accuracy of the proposed algorithm is over 97% for all of the instances except for berlin52 where it obtained 95%.
- In comparison to alternative algorithms considered in this research, the computational results show that the performance of the proposed algorithm is equal or better than most of these algorithms in terms of best solutions. As summarised in Figure 6.5, despite the types of TSP instances used to validate the effectiveness of these algorithms, the proposed algorithm shows better performance than the other 10 algorithms including RABNET-TSP, GA-PSO-ACO, ACO-ABC, SEE, PSO-ACO-3OPT, GVNS, IGA, ACS, ABO and IBA. Further, the proposed algorithm demonstrates an equal

performance with the MMS but a worse performance when compared to the GSA-ACS-PSO, SSA, REACSGA, AEAS and RAI. However, for instances with size larger than 100 cities, the proposed algorithm shows competitive results when compared to these algorithms. In particular, the proposed algorithm found better solutions than the SSA for lin318 and pcb442 as shown in Table 6.7. Moreover, as displayed in Table 6.11, the proposed algorithm shows an equal or better performance than the RAI for kro124p and ftv170, and have a close competition with REACSGA and AEAS for instances with cities between 101 and 318.

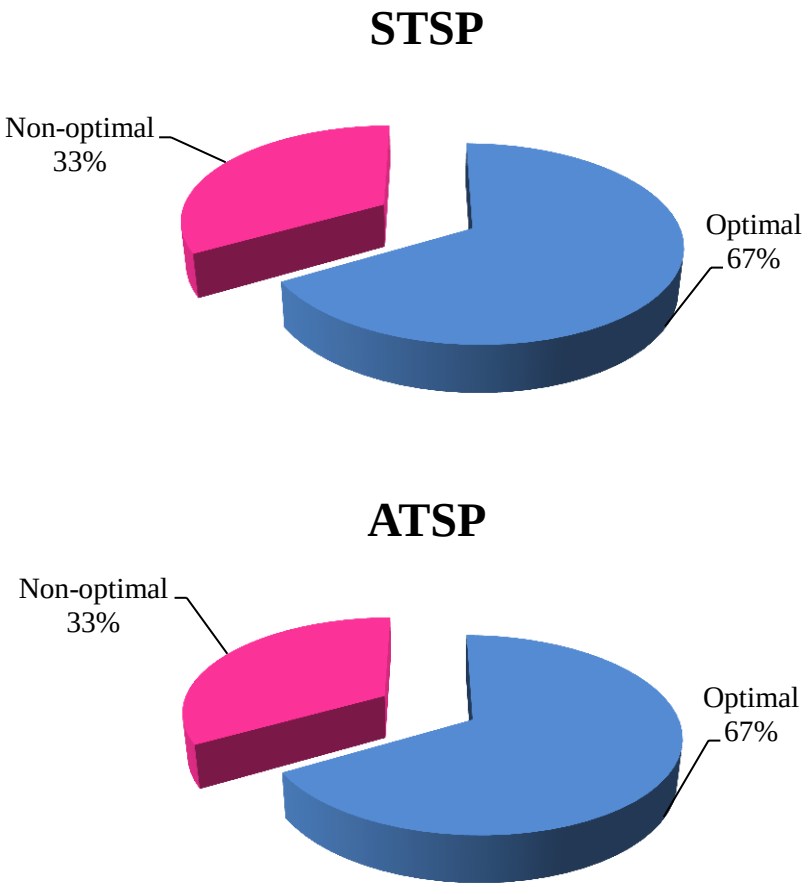


Figure 6.4: The overall performance of the proposed algorithm on the TSP benchmark problems

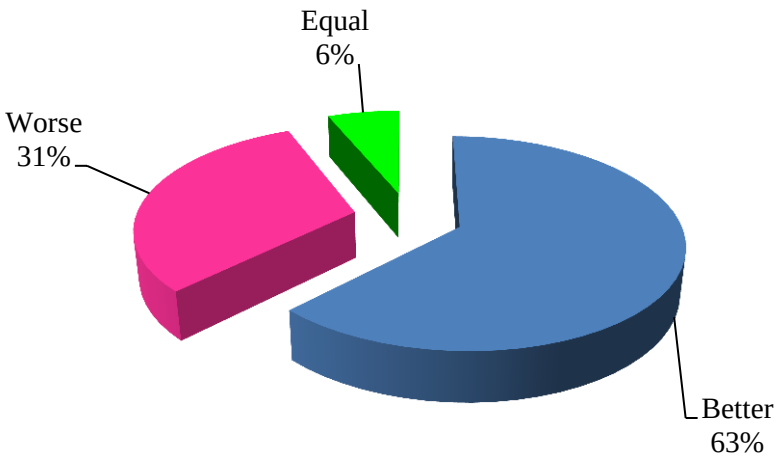


Figure 6.5: The overall performance of the proposed algorithm against other algorithms

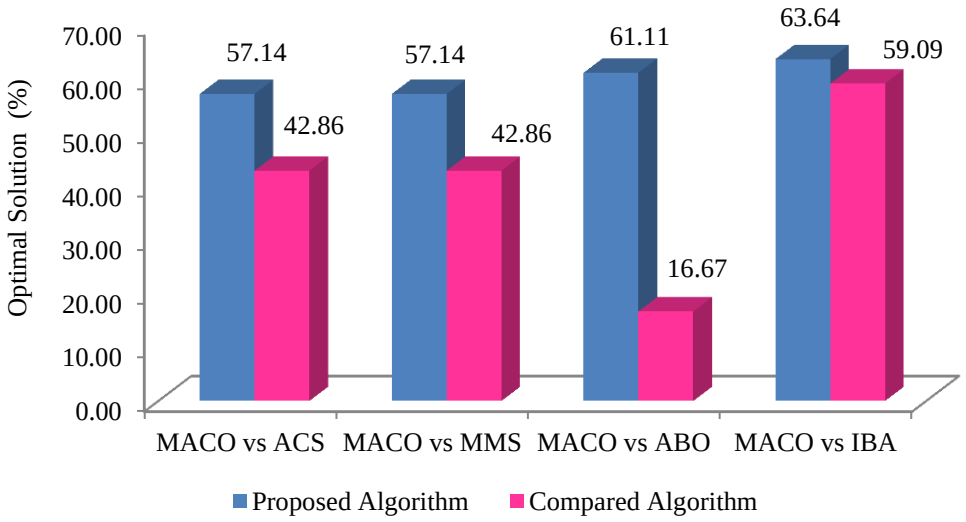


Figure 6.6: Percentage of the optimal solutions found by each algorithm applied to both ATSP and STSP

- The proposed algorithm exhibited excellent performance as compared to other algorithms applied to both symmetric and asymmetric TSP. As presented in Figure 6.6,

the proposed algorithm obtained a higher number of optimal solutions than the other algorithms in all cases.

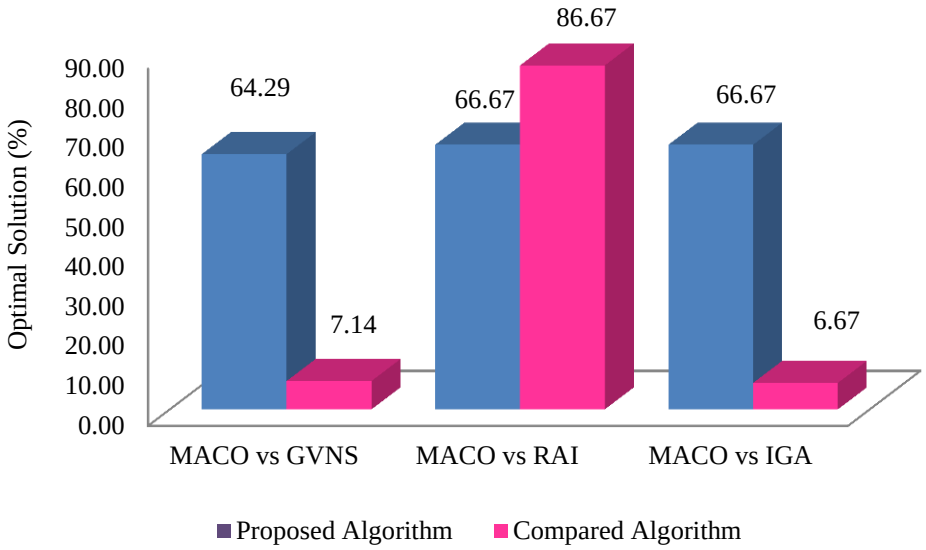


Figure 6.7: Percentage of the optimal solutions found by each algorithm applied to the ATSP instances

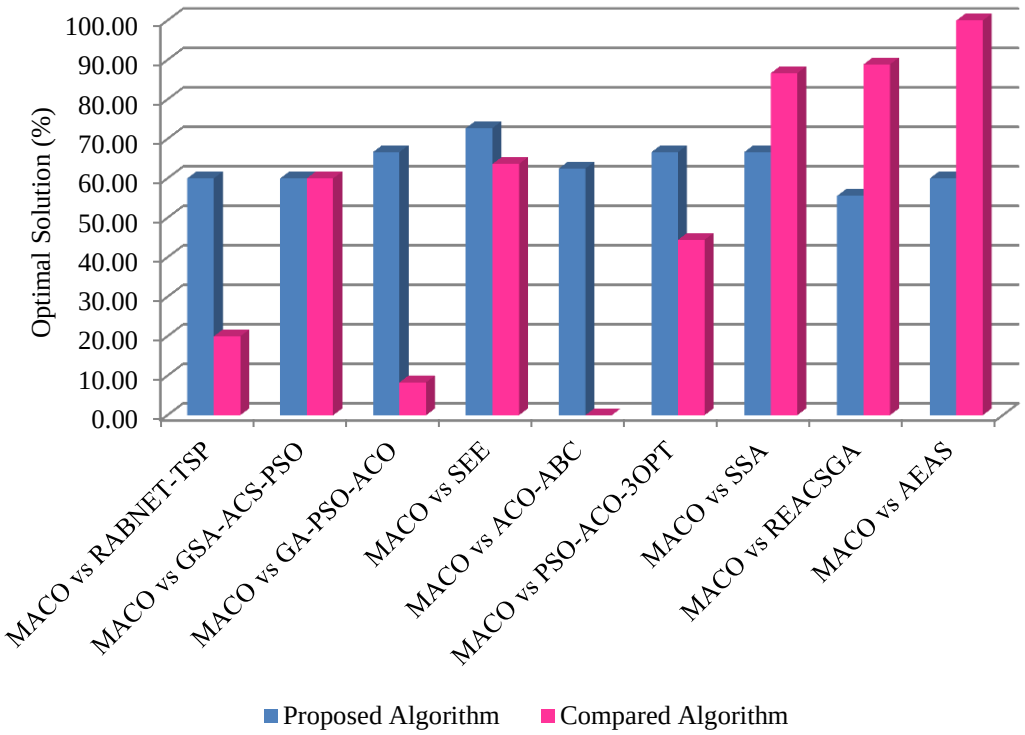


Figure 6.8: Percentage of the optimal solutions found by each algorithm applied to the STSP instances

- In term of optimal solutions, the proposed algorithm demonstrated a good performance with regards to other algorithms applied to the symmetric or asymmetric TSP. In particular, as shown in Figure 6.7, for each case, the proposed algorithm has obtained the optimal results in more than 64% of the ATSP instances. Likewise, the proposed algorithm has achieved the optimal solutions in over 55% of the STSP benchmark problems as displayed in Figure 6.8.

6.6 Conclusion

This chapter has demonstrated that the proposed MACO algorithm is able to produce good quality solutions for both symmetric and asymmetric TSPs within a reasonable CPU time. In particular, the performance of the proposed algorithm is highly competitive with regards to other studies in the literature designed for solving symmetric or asymmetric TSP. The relative errors reported on all the benchmark problems are below 5% for symmetric TSPs and 3% for

asymmetric TSPs. Similarly, the results achieved by the proposed MACO algorithm are very good in comparison to ACS, MMAS, ABO and IBA with more than 95% accuracy.

Additionally, the performance of the proposed MACO algorithm is better than the MACOd for both symmetric and asymmetric TSP instances. This indicates an appropriate choice of the ACO parameters settings which enables the proposed algorithm to be used for new problems without the need for parameter tuning thus helping to save time and cost.

The following chapter will conclude all the findings, contributions and future direction of this research.

Chapter 7

Conclusions

7.1 Overview

This research addressed two types of travelling salesman problems (TSP), the symmetric TSP and the asymmetric TSP. These problems can be used to model several practical problems, especially in scheduling and vehicle routing, thus the interest in researching how to solve them. Though exact methods can be used to solve these problems, it is not an alternative when the problem size increases, due to the exponentially increasing CPU times. Therefore, an ACO based algorithm was developed to solve both problems in an attempt to provide a good alternative to exact methods. In this chapter, a summary of scientific contributions and future research directions are presented. The main contribution of this thesis is the modified ACO algorithm with adaptive parameter settings for solving the symmetric and asymmetric TSP.

The first two chapters of this thesis gave an extensive review on the basic formulations of the TSP and solutions approaches applied to solve the TSP. This provided a complete picture of both TSP and ACO theoretical properties. In the following chapter, the new proposed approach was presented with an illustrative example. Chapter 4 suggested an adaptive setting for the ACO parameters used in the said approach while in Chapter 5 some factors that influence the performance of the proposed algorithm such as bound restriction and variable fixing were reviewed.

In Chapter 6, the proposed algorithm was compared with several algorithms in the literature. For the symmetric TSP, the comparisons were made with the RABNET-TSP (A.S.Masutti & Castro, 2009), GSA-ACS-PSO (Chen & Chien, 2011), GA-PSO-ACO (Deng, et al., 2012), SEE (Tuba & Jovanovic, 2013), ACO-ABC (Gunduz, Kiran, & Ozceylan, 2015), PSO-ACO-3Opt (Mahi, Baykan, & Kodaz, 2015), SSA (Wang, Lin, Zhong, & Zhang, 2016), REACSGA (Yousefikhoshbakht, Malekzadeh, & Sedighpour, 2016) and AEAS (Mohsen, 2016). Meanwhile, for the asymmetric TSP, the comparisons were made with the GVNS (Burke, Cowling, & Keuthen, 2001), RAI (Brest & Zerovnik, 2005) and IGA (Abdoun, Tajani, Abouchabaka, & Khatir, 2016). The comparisons for both symmetric and asymmetric TSP were made with the ACS (Gambardella & Dorigo, 1996), MMAS (Stutzle & Hoos, 1997), ABO

(Odili & Kahar, 2016) and IBA (Osaba, Yang, Diaz, Lopez-Garcia, & Carballedo, 2016). The computational comparisons have shown that the proposed approach has produced competitive or better results than the other algorithms. In addition, this chapter also has proved that the choice of parameter setting to be used in the proposed approach has contributed to a good performance of the algorithm.

7.2 Contribution

The goal of this research is to offer an alternative approach to solve both the symmetric and asymmetric TSP problems with adaptive parameter settings in a reasonable CPU time. The technical and conceptual contributions contained in this thesis are divided into three parts and briefly discussed below:

- The first part proposes a new approach to solve the symmetric and asymmetric TSP. A detailed review of the proposed algorithm which includes tour construction process, pheromone update process and enhancement process is described. In the tour construction process, a new formulation of the state transition rule has been proposed. This strategy aims at reducing the computational time by heuristically fixing part of the solution tour and improving the accuracy of the solutions through the usage of the solver.

  Further, in the pheromone update process, a different approach has been adopted in which the pheromone is deposited only on the edges belonging to the colony-best solution and evaporated only on the edges belonging to the colony-worst solution but not in the colony-best solution. Moreover, to prevent a worse quality solution in the next colony, the colony-best solution has been used as the first ant in the following colony.

- The second part suggests an adaptive parameter settings strategy of ACO parameters used in the proposed approach. Those parameters are the number of colonies, the number of ants in each colony, the relative influence of the pheromone trail α and pheromone evaporation rate ρ. These parameters have been defined as a function of the problem size.

- The last part presents the experimental results and performance of the proposed algorithm. The core contributions of this chapter lie in the empirical validation of the effectiveness of the proposed algorithm and the ability of the adaptive parameter setting strategy in producing good solutions.

  In order to provide a comprehensive validation of the algorithm, the proposed algorithm has been tested over a reasonable set of instances from the TSPLIB. A computational comparison between the results obtained using the set of parameter settings suggested in the literature and the ones proposed in this thesis has been performed, showing advantages of the proposed parameter setting.

  Further, these experimental results have been compared to other algorithms in the literature and the solutions obtained were competitive with the ones obtained with another algorithm designed for symmetric TSP or asymmetric TSP or both. Nevertheless, the numbers of the optimal or best solutions found by the proposed algorithm were always better than the other algorithms designed for both symmetric and asymmetric TSP.

  Therefore, the proposed algorithm has been a good alternative method to solve the TSP problems, having demonstrated a good performance over the existing heuristics.

7.3 The Idea for Future Work

The ‘Intelligent Ant’ strategy introduced in this thesis may be improved with a different value of X depending on the phase or situation of the algorithm. For instance, the number of nodes fixed in the tour construction process could be changed if the gap between the colony-best solution and the colony-worst solution is less than a certain value to diversify the search space. For larger problems, which may include thousands of nodes, the number of ants in the colony could be reduced by setting the number of nodes to be fixed on a per ant basis. In a case of stagnation, the search area could be set in differently for each ant or colony.

References

Alejandro Rodríguez; Rubén Ruiz. (2010). *The effect of asymmetry on traveling salesman problems*.

AMPL. (2013). *AMPL Optimization Inc.* United States.

TSPLIB. (2014a). Retrieved July 7, 2014, from http://elib.zib.de/pub/mp-testdata/tsp/tsplib/tsplib.html

TSPLIB. (2014b). Retrieved July 7, 2014, from http://elib.zib.de/pub/mp-testdata/tsp/tsplib/stsp-sol.html

A.S.Masutti, T., & Castro, L. N. (2009). A self-organizing neural network using ideas from the immune system to solve the traveling salesman problem. *Information Sciences, 179*(10), 1454-1468.

Aarts, E. H., & Lenstra, J. K. (2003). Introduction. In E. Aarts, & J. K. Lenstra (Eds.), *Local search in combinatorial optimization* (pp. 1-18). Princeton University Press, Princeton and Oxford.

Aarts, E. H., Korst, J. H., & Laarhoven, P. J. (1997). Simulated annealing. In E. Aarts, & J. K. Lenstra (Eds.), *Local Search in Combinatorial Optimization* (pp. 91-120). John Wiley & Sons Ltd.

Abdoun, O., Tajani, C., Abouchabaka, J., & Khatir, H. E. (2016). Improved Genetic Algorithm to Solve Asymmetric Traveling Salesman Problem. *International Journal of Open Problems in Computer Science & Mathematics, 9*(4), 42-55.

Affenzeller, M., Winkler, S., Wagner, S., & Beham, A. (2009). *Genetic Algorithms and Genetic Programming : modern concepts and practical applications.* CRC Press.

Afshar, M. (2006). Afshar, M. H. (2006). Application of a max–min ant system to joint layout and size optimization of pipe networks. *Engineering optimization, 38*(3), 299-317.

Alba, E., & Troya, J. M. (1999). A survey of parallel distributed genetic algorithms. *Complexity, 4*(4), 31-52.

Ali, M. K., & Kamoun, F. (1993). Neural Networks for Shortest Path Computation and Routing in Computer Networks. *IEEE Transactions on Neural Networks, 4*(6), 941-952.

Amir, C., Badr, A., & Farag, I. (2007). A Fuzzy Logic Controller for Ant Algorithms. *Computing and Information Systems, 11*(2), 26.

Applegate, D. L., Bixby, R. E., Chvatal, V., & Cook, W. (1998). On the solution of traveling salesman problems. *Documenta Mathematica Journal der Deutschen Mathematiker-Vereinigung, International Congress of Mathematicians*, 645-656.

Atamturk, A., & Savelsbergh, M. W. (2005). Integer-Programming Software Systems. *Annals of Operations Research, 140*, 67–124.

Baez, S., Angel-Bello, F., & Alvarez, A. (2016). Time-dependent formulation for minimizing total completion time in a parallel machine scheduling problem with dependent setup times. *IFAC-PapersOnLine, 49*(12), 857-862.

Balas, E., & Toth, P. (1983). *Branch and Bound Methods for the Traveling Salesman Problem.*

Barral, D., Perrin, J.-P., Dombre, E., & Liegeois, A. (1999). An Evolutionary Simulated Annealing Algorithm for Optimizing Robotic Task Point Ordering. *Proceeding of the 1999 IEEE lntemational Symposium on Assembly and Task Planning*, (pp. 157-162). Porto,Portugal.

Basu, J. K., Bhattacharyya, D., & Kim, T.-h. (2010). Use of Artificial Neural Network in Pattern Recognition. *International Journal of Software Engineering and Its Applications*, 23-33.

Basu, S. (2012). Tabu Search Implementation on Traveling Salesman Problem and Its Variations: A Literature Survey. *American Journal of Operations Research*, 163-173.

Bektas, T. (2006). The multiple traveling salesman problem : an overview of formulations and solution procedures. *Omega, 34*(3), 209-219.

Bhríde, F. M., McGinnity, T., & McDaid, L. (2005). Landscape classification and problem specific reasoning for genetic algorithms. *Kybernetes, 34*(9/10), 1469-1495.

Bigras, L.-P., Gamache, M., & Savard, G. (2008). The time-dependent traveling salesman problem and single machine scheduling problems with sequence dependent setup times. *Discrete Optimization, 5*(4), 685-699.

Birattari, M., Paquete, L., Stutzle, T., & Varrentrapp, K. (2001). *Classification of Metaheuristics and Design of Experiments for the Analysis of Components.* Tech. Rep AIDA-01-05.

Blum, C. (2005). Ant colony optimization: Introduction and recent trends. *Physics of Life Reviews, 2*, 353-373.

Blum, C., & Roli, A. (2008). Hybrid Metaheuristics : An Introduction. In C. Blum, A. Roli, & M. Sampels (Eds.), *Hybrid Metaheuristics : An Emerging Approach to Optimization* (pp. 1-30). Springer.

Blum, C., Puchinger, J., Raidl, G., & Roli, A. (2010). A Brief Survey on Hybrid Metaheuristics. *Proceedings of BIOMA*, (pp. 3-18).

Blum, C., Roli, A., & Sampels, M. (Eds.). (2008). *Hybrid Metaheuristics. An Emerging Approach to Optimization.* Springer.

Bonomi, E., & Lutton, J.-L. (1984). The N-city Travelling Salesman Problem : Statistical Mechanics and the Metropolis Algorithm. *SIAM Review, 26*(4), 551-568.

BoussaiD, I., Lepagnot, J., & Siarry, P. (2013). A survey on optimization metaheuristics. *Information Sciences : an International Journal, 237,* 82-117.

Brest, J., & Zerovnik, J. (2005). A Heuristic for the Asymmetric Traveling Salesman Problem. *MIC2005. The 6th Metaheuristics International Conference*, (pp. 145-150). Vienna,Austria.

Bullnheimer, B., Hartl, R. F., & Strauss, C. (1999a, January). A New Rank Based Version of the Ant System - A Computational Study. *Central European Journal of Operations Research, 7*(1), 25-38.

Bullnheimer, B., Hartl, R. F., & Strauss, C. (1999b). An improved ant System algorithm for thevehicle Routing Problem. *Annals of operations research, 89*, 319-328.

Burke, E. K., Cowling, P. I., & Keuthen, R. (2001). Eff ective Local and Guided Variable Neighbourhood Search Methods for the Asymmetric Travelling Salesman Problem. In E. J. Boers (Ed.), *Applications of Evolutionary Computing. EvoWorkshops 2001. Lecture Notes in Computer Science, vol 2037* (pp. 203-212). Springer.

Capriles, P. V., Fonseca, L. G., Barbosa, H. J., & Lemonge, A. C. (2007). Rank-based ant colony algorithms for truss weight minimization with discrete variables. *Communications in numerical methods in engineering, 23*(6), 553-575.

Castillo, O., Neyoy, H., Soria, J., Melin, P., & Valdez, F. (2015). A new approach for dynamic fuzzy logic parameter tuning in Ant Colony Optimization and its application in fuzzy control of a mobile robot. *Applied Soft Computing, 28*, 150-159.

Chen, S.-M., & Chien, C.-Y. (2011). Solving the traveling salesman problem based on the genetic simulated annealing ant colony system with particle swarm optimization techniques. *Expert Systems with Applications, 38*(12), 14439-14450.

Cheng, C.-H., Lee, W.-K., & Wong, K.-F. (2002). A Genetic Algorithm-Based Clustering Approach for Database Partitioning. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews), 32*(3), 215-230.

Claus, A. (1984, March). A New Formulation for the Traveling Salesman Problem. *SIAM Journal on Algebraic Discrete Methods, 5*(1), 21-25.

Climer, S., & Zhang, W. (2006). Cut and Solve:An iterative search strategy for combinatorial optimization problems. *Artificial Intelligence, 170*, 714-738.

Colorni, A., Dorigo, M., & Maniezzo, V. (1991). Distributed Optimization by ant colonies. *European Conference on Artifical Life* (pp. 134-142). Paris: Elsevier Publishing.

Coppin, B. (2004). *Artificial Intelligence Illuminated.* Jones and Bartlett Publishers.

Cordeau, J.-F., & Laporte, G. (2005). Tabu Search Heuristics for the Vehicle Routing Problem. In P. R. Sharda, P. D. Voβ, C. Rego, & B. Alidaee (Eds.), *Metaheuristic Optimization via Memory and Evolution.* Kluwer Academic Publisher.

Cordeau, J.-F., Laporte, G., Savelsbergh, M. W., & Vigo, D. (2007). Vehicle Routing. In C. Barnhart, & G. Laporte (Eds.), *Handbooks in Operations Research and Management Science* (pp. 367-). North-Holland.

Cordon, O., Herrera, F., & Stutzle, T. (2002). A Review on the Ant Colony Optimization Metaheuristic : Basis,Models and New Trends. *Mathware and Soft Computing, 9*(2-3), 141-175.

Cordon, O., Herrera, F., Viana, I. F., & Moreno, L. (2000). A New ACO Model Integrating Evolutionary Computation Concepts: The Best-Worst Ant System. *Proceedings of ANTS'2000.* Brussels, Belgium.

Cotta-Porras, C. (1998). A study of hybridisation techniques and their application to the design of evolutionary algorithms. *AI Communications, 11*(3,4), 223-224.

Crowder, H., & Padberg, M. W. (1980). Solving Large-Scale Symmetric Travelling Salesman Problems to Optimality. *Management Science, 26*(5), 495-509.

Crowder, H., Johnson, E. L., & Padberg, M. (1983). Solving Large-Scale Zero-One Linear Programming Problems. *Operations Research, 31*(5), 803-834.

Dantzig, G. B. (1963). *Linear Programming and Extensions.* Princeton University Press.

Darani, N. M., Dolatnejad, A., & Yousefikhoshbakht, M. (2015). A reactive bone route algorithm for solving the traveling salesman problem. *Journal of Industrial Engineering and Management Studies, 2*(2), 13-25.

Davis, L. (1991). *Handbook of Genetic Algorithm.* New York: Van Nostrand Reinhold.

Deneubourg, J., Aron, S., Goss, S., & Pasteels, J. M. (1990). The Self-Organizing Exploratory Pattern of the. *Journal of lnsect Behavior, 3*(2), 159-168.

Deng, W., Chen, R., He, B., Liu, Y., Yin, L., & Guo, J. (2012). A novel two-stage hybrid swarm intelligence optimization algorithm and application. *Soft Computing, 16*(10), 1707--1722.

Dorigo, M., & Caro, G. D. (1999). Ant Colony Optimization : A New Meta-Heuristic. *Proceedings of the 1999 Congress on Evolutionary Computation-CEC99. 2*, pp. 1470-1477. IEEE.

Dorigo, M., & Gambardella, L. M. (1997a). Ant Colony System:A Cooperative Learning Approach to the Traveling Salesman Problem. *IEEE Transactions on Evolutionary Computation, 1*(1), 53–66.

Dorigo, M., & Gambardella, L. M. (1997b). Ant colonies for the travelling salesman problem. *Biosystems, 43*(2), 73-81.

Dorigo, M., & Stutzle, T. (2004). *Ant Colony Optimization.* MIT Press.

Dorigo, M., & Stutzle, T. (2010). Ant Colony Optimization : Overview and Recent Advances. In M. Gendreau, & J.-Y. Potvin (Eds.), *Handbook of Metaheuristics. International Series in Operation Research and Management Science* (Vol. 146). Springer.

Dorigo, M., Birattari, M., & Stutzle, T. (2006). Ant Colony Optimization Artificial Ants as a Computational Intelligence Technique. *IEEE Computational Intelligence Magazine, 1*(4), 28-39.

Dorigo, M., Maniezzo, V., & Colorni, A. (1991). *Positive Feedback as a Search Strategy.* Politecnico Di Milano, DipartimentoI Di Elettronica.

Dorigo, M., Maniezzo, V., & Colorni, A. (1996, February). Ant System: Optimization by a colony of cooperating agents. *IEEE Transaction on Systems, Man, and Cybernetics - Part B, 26*(1), 29-41.

Dowsland, K. A. (2014). Classical Techniques. In E. K. Burke, & G. Kendall (Eds.), *Search Methodologies: Introductory Tutorials in Optimization and Decision Support Techniques* (Second Edition ed., pp. 19-65). Springer.

Dowsland, K. A. (2014). Classical Techniques. In E. Burke, & G. Kendall (Eds.), *Search Methadologies: Introductory Tutorials in Optimization and Decision Support Techniques* (pp. 19-65). Springer.

Dumitrescu, I., & Stutzle, T. (2003). Combinations of Local Search and Exact Algorithms. In G. R. al. (Ed.), *Applications of Evolutionary Computation* (pp. 211-223).

Eiben, A. E., Hinterding, R., & Michalewicz, Z. (1999). Parameter Control in Evolutionary Algorithms. *IEEE Transactions on Evolutionary Computation, 3*(2), 124-141.

Favuzza, S., Graditi, G., & Sanseverino, E. R. (2006). Adaptive and Dynamic Ant Colony Search Algorithm for Optimal Distribution Systems Reinforcement Strategy. *Applied Intelligence, 24*, 31-42.

Fred Glover. (1986, January). Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research, 13*(5), 533-549.

Fujimura, K., Fujiwaki, S.-i., Kwaw, O.-C., & Tokataka, H. (2001). Optimization of electronic chip-mounting machine using SOM-TSP method with 5 dimensional data. *Proc. ICII, 4*, pp. 26-31.

Furini, F., Persiani, C. A., & Toth, P. (2016). The Time Dependent Traveling Salesman Planning Problem in Controlled Airspace. *Transportation Research Part B, 90*, 38-55.

G.B.Dantzig, Fulkerson, D., & Johnson, S. (1954). Solution of a large-scale traveling salesman problem. *Journal of the Operations Research Society of America, 2*(4), 393-410.

Gambardella, L. M., & Dorigo, M. (1996). Solving symmetric and asymmetric TSPs by ant colonies. *Proceedings of IEEE International Conference on Evolutionary Computation* (pp. 622-627). IEEE.

Gambardella, L. M., & Dorigo, M. (2000). An Ant Colony System Hybridized with a New Local Search for the Sequential Ordering Problem. *INFORMS Journal on Computing, 12*(3), 237-255.

Gavish, B., & Graves, S. C. (1978). *The Travelling Salesman Problem and Related Problems.* Operation Research Centre. Cambridge: Massachusetts Institute of Technology.

Gendreau, M. (2003). An Introduction to Tabu Search. In F. Glover, & G. A. Kochenberger (Eds.), *Handbook of Metaheuristics. International Series in Operational Research & Management Science* (Vol. 57, pp. 37-54). Springer.

Gendreau, M., & Potvin, J.-Y. (2005). Metaheuristics in Combinatorial Optimization. *Annals of Operation Research, 140*, 189-213.

Gendreau, M., & Potvin, J.-Y. (2010). Tabu Search. In M. Gendreau, & J.-Y. Potvin (Eds.), *Handbook of Metaheuristic* (pp. 41-59). Springer.

Gendreau, M., Soriano, P., & Salvail, L. (1993). Solving the Maximum Clique Problem Using A Tabu Search Approach. *Annals of Operation Research, 43*(1-4), 385-403.

Glover, F., & Laguna, M. (1997). *Tabu Search.* Kluwer Academic Publisher.

Golden, B. L., & Skiscim, C. C. (1986). Using Simulated Annealing to SOlve Routing and Location Problems. *Naval Research Logistics Quarterly, 33*, 261-279.

Golden, B., Bodin, L., Doyle, T., & Jr, W. S. (1980). Approximate Traveling Salesman Algorithms. *Operations Research, 28*(3,Part 2), 694-711.

Gomory, R. E. (1958). Outline of an algorithm for integer solutions to linear programs. *Bulletin of the American Mathematical Society, 64*, 275-278.

Goss, S., Aron, S., Deneubourg, J., & Pasteels, J. (1989). Self-organized Shortcuts in the Argentine Ant. *Naturwissenschaften, 76*, 579- 581.

Gu, X. F., Liu, L., Li, J.-P., & Yuan-Yuan Huang, J. L. (2008). Data Classification based on Artificial Neural Networks. *2008 International Conference on Apperceiving Computing and Intelligence Analysis.* IEEE.

Gunduz, M., Kiran, M. S., & Ozceylan, E. (2015). A hierarchic approach based on swarm intelligence to solve the traveling salesman problem. *Turkish Journal of Electrical Engineering & Computer Sciences*, 103-117.

Hansen, P., & Mladenovic, N. (2001). Variable neighborhood search : Principles and applications. *European Journal of Operational Research, 130*, 449-467.

Hansen, P., Mladenovic, N., Brimberg, J., & Moreno, J. (2010). Variable Neighbourhood Search. In M. Gendreau, & J.-Y. Potvin (Eds.), *Handbook of Metaheuristics* (pp. 61-86).

Hao, Z., Huang, H., Qin, Y., & Cai, R. (2007). An ACO Algorithm with Adaptive Volatility Rate of Pheromone Trail. In Y. Shi, G. D. Albada, J. Dongarra, & P. M. Sloot (Ed.), *Computational Science-ICCS 2007*, (pp. 1167-1170).

Hao, Z.-F., Cai, R.-C., & Huang, H. (2006). An Adaptive Parameter Control Strategy for ACO. *Proceedings of the Fifth International Conference on Machine Learning and Cybernetics*, (pp. 203-206).

Hassoun, M. H. (1995). *Fundamentals of Artificial Neural Networks.* MIT Press.

Haykin, S. (1999). *Neural Networks A Comprehensive Foundation.* USA: Prentice-Hall International, Inc.

Helsgaun, K. (2000). An Effective Implementation of the Lin-Kernighan Traveling Salesman Heuristic. *European Journal of Operational Research, 126*, 106-130.

Hertz, A., Taillard, E., & Werra, D. d. (1997). Tabu search. In E. Aarts, & J. K. Lenstra (Eds.), *Local Search in Combinatorial Optimization* (pp. 121-136). John Wiley & Sons Ltd.

Hillier, F. S., & Lieberman, G. J. (2010). *Introduction to Operations Research.* McGraw-Hill.

Hoffman, K. L., & Padberg, M. (1991). Improving LP-Representations of Zero-One Linear Programs for Branch-and-Cut. *ORSA Journal on Computing, 3*(2), 121-134.

Holland, J. H. (1992). *Adaptation in Natural and Artificial Systems:An Introductory Analysis with Application to Biology,Control, and Artificial Intelligence.* First MIT Press.

Huang, H., Yang, X., Hao, Z., & Cai, R. (2006). A Novel ACO Algorithm with Adaptive Parameter. In D.-S. Huang, K. Li, & G. W. Irwin (Ed.), *International Conference on Intelligent Computing, ICIC 2006*, (pp. 12-21).

Karaboga, D. (2005). *An Idea Based on Honey Bee Swarm For Numerical Optimization.* Erciyes university, Engineering Faculty, Computer Engineering Department.

Kennedy, J., & Eberhart, R. (1995). Particle swarm optimization. *Proceedings of ICNN'95 - International Conference on Neural Networks.* IEEE.

Kirkpatrick, S., Gelatt, C. D., & Vecchi, M. P. (1983). Optimization by Simulated Annealing. *Science, 220*, 671-680.

Knox, J. (1994). Tabu Search Performance on the Symmetric Traveling Salesman Problem. *Computers & Operations Research, 21*(8), 867-876.

Kolman, B., & Beck, R. E. (1995). *Elementary Linear Programming with Applications.* Academic Press.

Land, A., & Doig, A. (1960). An Automatic Method of Solving Discrete Programming Problems. *Econometrica, 28*(3), 497--520.

Laporte, G., Gendreau, M., Potvin, J.-Y., & Semet, F. (2000). Classical and modern heuristics for the vehicle routing problem. *International Transactions in Operational Research, 7*(4-5), 285-300.

Lawler, E. L., Lenstra, J. K., Kan, A. H., & Shmoys, D. B. (1985). *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization.* John Wiley & Sons.

López-Ibáñez, M., Stützle, T., & Dorigo, M. (2016). Ant Colony Optimization : A Component-Wise Overview. In R. Martí, P. Panos, & M. G. Resende (Eds.), *Handbook of Heuristics* (pp. 1-37). Springer.

Mahi, M., Baykan, O. K., & Kodaz, H. (2015). A new hybrid method based on Particle Swarm Optimization, Ant Colony Optimization and 3-Opt algorithms for Traveling Salesman Problem. *Applied Soft Computing, 30*, 484-490.

Malim, M. R., & Halim, F. A. (2012). Immunology and Artificial Immune Systems. *International Journal on Artificial Intelligence Tools, 21*(6). doi: 10.1142/S0218213012500315

Maniezzo, V., & Colorni, A. (1999). The Ant System Applied to the Quadratic Assignmnet Problem. *IEEE Transactions on Knowledge and Data Engineering, 11*(5), 769-778.

Mehrotra, K., Mohan, C. K., & Ranka, S. (2000). *Elements of Artificial Neural Networks.* Massachusetts Institute of Technology.

Michalewicz, Z., & Fogel, D. B. (2004). *How to Solve It :Modern Heuristics.* Springer.

Miller, C., Tucker, A., & Zemlin, R. (1960, October). Integer Programming Formulation of Traveling Salesman Problems. *Journal of the ACM, 7*(4), 326-329.

Miller, S. J. (2017). *Mathematics of Optimization: How to do things faster.* American Mathematical Society.

Mitchell, J. E. (2011). Branch and Cut. In J. J. Cochran, J. Louis Anthony Cox, P. Keskinocak, J. P. Kharoufeh, & J. C. Smith (Eds.), *Wiley Encyclopedia of Operations Research and Management Science.* John Wiley & Sons.

Mitchell, M. (1998). *An Introduction to Genetic Algorithms.* First MIT Press.

Mladenovic, N., & Hansen, P. (1997). Variable Neighborhood Search. *Computers & Operations Research, 24*(11), 1097-1100.

Mohsen, A. M. (2016). Annealing Ant Colony Optimization with Mutation Operator for Solving TSP. *Computational Intelligence and Neuroscience.*

Naddef, D., & Rinaldi, G. (2002). Branch and Cut Algortihms for the Capacited VRP. In P. Toth, & D. Vigo (Eds.), *The Vehicle Routing Problem* (pp. 53-84). Society of Industrial and Applied Mathematics.

Nahar, S., Sahni, S., & Shragowitz, E. (1986). Simulated Annealing and COmbinatorial Optimization. *23rd ACM/IEEE Design Automation Conference,* (pp. 293-299).

Nikolaev, A. G., & Jacobson, S. H. (2010). Simulated Annealing. In M. Gendreau, & J. Potvin (Eds.), *Handbook of Metaheuristics* (pp. 1-40). Springer .

Odili, J. B., & Kahar, M. N. (2016). Solving the Traveling Salesman's Problem using the African Buffalo Optimization. *Computational Intelligence and Neuroscience*.

Odili, J. B., Kahar, M. N., & Anwar, S. (2015). African Buffalo Optimization: A Swarm-Intelligence Technique. *Procedia Computer Science, 76*, 443-448.

Oncan, T., Altinel, I. K., & Laporte, G. (2009). A comparative analysis of several asymmetric traveling salesman problem formulations. *Computers & Operations Research, 36*, 637-654.

Osaba, E., Yang, X.-S., Diaz, F., Lopez-Garcia, P., & Carballedo, R. (2016). An improved discrete bat algorithm for symmetric and asymmetric Traveling Salesman Problems. *Engineering Applications of Artificial Intelligence, 48*, 59-71.

Osman, I. H., & Kelly, J. P. (1996). Meta-heuristics:An Overview. In I. H. Osman, & J. P. Kelly (Eds.), *Meta-heuristics:Theory & Application* (pp. 1-22). Kluwer Academic Publisher.

Padberg, M., & Sung, T.-Y. (1991). An analytical comparison of different formulations of the travelling salesman problem. *Mathematical Programming, 52*, 315-357.

Pasti, R., & Castro, L. N. (2006). A Neuro-Immune Network for Solving the Traveling Salesman Problem . *International Joint Conference on Neural Networks,* (pp. 3760-3766).

Pilat, M. L., & White, T. (2002). Using Genetic Algorithm to Optimize ACS-TSP. In D. M., D. C. G., & S. M. (Eds.), *Ant Algorithms. ANTS 2002. Lecture Notes in Computer Science* (Vol. 2463, pp. 282-287). Springer,Berlin, Heidelberg.

Priddy, K. L., & E.Keller, P. (2005). *Artificial Neural Networks: An Introduction.* The Society of Photo-Optical Instrumentation Engineers.

Puchinger, J., & Raidl, G. R. (2005). Combining Metaheuristics and Exact Algorithms in Combinatorial Optimization : A Survey and Classification. In J. Mira, & J. R. Alvarez (Eds.), *Artificial Intelligence and Knowledge Engineering Applications: A Bioinspired Approach. IWINAC 2005* (Lecture Notes in Computer Science ed., Vol. 3562, pp. 41-53). Springer,Berlin,Heidelberg.

Punnen, A. P. (2007). The Traveling Salesman Problem:Applications,Formulations and Variations. In G. Gutin, & A. P. Punnen (Eds.), *The Traveling Salesman Problem and Its Variations* (pp. 1-28). USA: Springer.

Qin, L., Luo, J., Chen, Z., Guo, J., Chen, L., & Pan, Y. (2006). Phelogenetic Tree Construction using Self adaptive Ant Colony Algorithm. *Proceedings of the First International Multi-Symposiums on Computer and Computational Sciences (IMSCCS'06)* (pp. 179-187). IEEE.

Rader, D. J. (2010). *Deterministic Operation Research : Models and Methods in Linear Optimization.* John Wiley & Sons.

Raidl, G. R. (2006). A Unified View on Hybrid Metaheuristics. In A. F. al. (Ed.), *Hybrid Metaheuristics. HM 2006. Lecture Notes in Computer Science* (Vol. 4030, pp. 1-12). Springer.

Raidl, G. R., & Puchinger, J. (2008). Combining (Integer) Linear Programming Techniques and Metaheuristics for Combinatorial Optimization. In C. Blum, A. Roli, & M. Samples (Eds.), *Hybrid Metaheuristics An Emerging Approach to Optimization* (pp. 31-62). Springer.

Raidl, G. R., & Puchinger, J. (2008). Combining (Integer) Linear Programming Techniques and Metaheuristics for Combinatorial Optimization. In C. Blum, M. O. Aguilera, A. Roli, & M. Sampels (Eds.), *Hybrid Metaheuristics An Emerging Approach to Optimization* (pp. 31-62). Springer.

Raidl, G. R., Puchinger, J., & Blum, C. (2010). Metaheuristic Hybrids. In M. Gendreau, & J.-Y. Potvin (Eds.), *Handbook of Metaheuristics* (pp. 469-496). Springer.

Randall, M. (2004). Near Parameter Free Ant Colony Optimisation. In M. Dorigo, M. Birattari, C. Blum, L. M. Gambardella, F. Mondada, & T. Stutzle (Ed.), *4th Internation Workshop, ANTS 2004*, (pp. 374-381).

Reeves, C. (2003). Genetic Algorithms. In F. Glover, & G. A. Kochenberger (Eds.), *Handbook of Metaheuristics. International Series in Operations Research & Management Science* (Vol. 57, pp. 55-82). Springer.

Reeves, C. R. (1997). Genetic Algoithms for the Operations Researcher. *INFORMS Journal on Computing, 9*(3), 231-250.

Rego, C., & Glover, F. (2007). Local search and metaheuristics. In G. Gutin, & A. P. Punnen (Eds.), *The Traveling Salesman Problem and Its Variations* (pp. 309-368). Springer.

Reinelt, G. (1994). *The Traveling Salesman: Computational Solutions for TSP Applications.* Springer.

Riadl, G. R., & Puchinger, J. (2008). Combining (Integer) Linear Programming Techniques and Metaheuristics for Combinatorial Optimization. In C. Blum, M. O. Aguilera, A. Roli, & M. Sampels (Eds.), *Hybrid Metaheuristics An Emerging Approach to Optimization* (pp. 31-62). Springer.

Rochat, Y., & Taillard, E. D. (1995). Probabilistic Diversification and Intensification in Local Search for Vehicle Routing. *Journal of Heuristics, 1*(1), 147-167.

Rossier, Y., Troyon, M., & Liebling, T. M. (1986). Probabilistic exchange algorithm and Euclidean traveling: Salesman problems. *Operations Research Spektrum, 8*, 151-164.

Sastry, K., Goldberg, D. E., & Kendall, G. (2014). Genetic Algorithms. In E. K. Burke, & G. Kendall (Eds.), *Search Methodologies. Introductory Tutorials in Optimization and Decision Support Techniques* (pp. 93-117). Springer.

Silberholz, J., & Golden, B. (2010). Comparison of Metaheuristics. In M. Gendreau, & J.-Y. Potvin (Eds.), *Handbook of Metaheuristics* (pp. 625-640). Springer.

Skorin-Kapov, J. (1990). Tabu Search Applied to the Quadratic Assignment Problem. *ORSA Journal on Computing, 2*(1), 33-45.

Socha, K., Knowles, J., & Sampels, M. (2002). A MAX-MIN Ant System for the University Course Timetabling Problem. In M. Dorigo, G. D. Caro, & M. Samples (Eds.), *Ant Algorithms: Third International Workshop, ANTS 2002, Brussels, Belgium, September 12-14, 2002. Proceedings* (Vol. 2463). Springer Science & Business Media.

Sorensen, K., & Glover, F. (2016). Metaheuristics. In S. I. Gass, & M. C. Fu (Eds.), *Encylopedia of Operations Research and Management Science* (pp. 960-970). New York: Springer.

Stutzle, T., & Hoos, H. (1997). MAX-MIN Ant System and Local Search for the Traveling Salesman Problem. *Proceeding of IEEE Conference on Evolutionart Computation*, (pp. 309-314). Indianapolis.

Stutzle, T., & Hoos, H. (1997). MAX-MIN Ant System and Local Search for the Traveling Salesman Problem. *Proceeding of IEEE Conference on Evolutionart Computation*, (pp. 309-314). Indianapolis.

Stutzle, T., & Hoos, H. H. (1996). *Improving the Ant System:A detailed report on the MAX-MIN Ant System.* FG Intellektik, FB Informatik,TU Darmstadt,Germany,Tech. Rep. AIDA-96-12.

Stützle, T., & Hoos, H. H. (2000). MAX –MIN Ant System. *Future Generation Computer Systems, 16*, 889-914.

Taillard, E. (1990). Some Efficient Heuristic Methods for the Flow Shop Sequencing Problem. *European Journal of Opertaional Research, 47*, 65-74.

Taiwo, O. S., Mayowa, O. O., & Ruka, K. B. (2013). Application of Genetic Algorithm to Solve Traveling Salesman Problem. *International Journal of Advanced Research (IJOAR), 1*(4), 27-46.

Tarantilis, C., & Kiranoudis, C. (2002). BoneRoute: An Adaptive Memory-Based Method for Effective Fleet Management. *Annals of Operations Research, 115*, 227-241.

Ting, T. O., Yang, X.-S., Cheng, S., & Huang, K. (2015). Hybrid Metaheuristic Algorithms: Past, Present, and Future. In X.-S. Yang (Ed.), *Recent Advances in Swarm Intelligence and Evolutionary Computation* (pp. 71-83). Springer.

Tuba, M., & Jovanovic, R. (2013). Improved ACO Algorithm with Pheromone Correction Strategy for the Traveling Salesman Problem. *International Journal of Computers Communications & Control, 8*(3), 477-485.

Vanderbei, R. J. (2001). *Linear Programming Foundations and Extensions Second Edition.* Springer Science+Business Media,LLC.

Wang, C., Lin, M., Zhong, Y., & Zhang, H. (2016). Swarm simulated annelaing algorithm with knowledge-based sampling for travelling salesman problem. *Int. J. Intelligent Systems Technologies and Applications, 15*(1), 74-94.

Watanabe, I., & Matsui, S. (2003). Improving the Performance of ACO Algorithms by Adaptive Control of Candidate Set. *Evolutionary Computation, 2003. CEC'03. The 2003 Congress. 2*, pp. 1355--1362. IEEE.

Williams, H. P. (2006). The Formulation and Solution of Discrete Optimisation Models. In G. Appa, L. Pitsoulis, & H. P. Williams (Eds.), *Handbook on Modelling for Discrete Optimization* (pp. 3-38). Springer.

Williams, H. P. (2006). The Formulation and Solution of Discrete Optimisation Models. In *Handbook on Modelling for Discrete Optimization* (G. M. Appa, L. Pitsoulis, & H. Williams, Trans., pp. 3-38). Springer.

Wong, L.-P., Low, M. Y., & Chong, C. S. (2008). A Bee Colony Optimization Algorithm for Traveling Salesman Problem. *Modeling & Simulation,2008. AICIMS 08. Second Asia International Conference*, (pp. 818-823).

Yang, X.-S. (2010). A New Metaheuristic Bat-Inspired Algorithm. In J. R. González, D. A. Pelta, C. Cruz, G. Terrazas, & N. Krasnogor (Eds.), *Nature Inspired Cooperative Strategies for Optimization (NICSO 2010). Studies in Computational Intelligence* (Vol. 284, pp. 65-74). Springer, Berlin, Heidelberg.

Yang, X.-S. (2010). A New Metaheuristic Bat-Inspired Algorithm. In J. R. González, D. A. Pelta, C. Cruz, G. Terrazas, & N. Krasnogor (Eds.), *Nature Inspired Cooperative Strategies for Optimization (NICSO 2010).* Springer, Berlin, Heidelberg.

Yip, P. P., & Pao, Y.-H. (1995). Combinatorial Optimization with Use of Guided Evolutionary Simulated Annealing. *IEEE Transactions on Neural Networks, 6*(2), 290-295.

Yousefikhoshbakht, M., Malekzadeh, N., & Sedighpour, M. (2016). Solving the Traveling Salesman Problem Based on The Genetic Reactive Bone Route Algorithm whit Ant Colony System. *International Journal of Production Management and Engineering, 4*(2), 65-73.

Zecchin, A. C., Maier, H. R., Simpson, A. R., Roberts, A. J., Berrisford, M. J., & Leonard, M. (2003). Max–Min Ant System applied to water distribution system

optimisation. *International Congress on Modelling and Simulation (MODSIM03), Modelling and Simulation*, (pp. 795-800).

Zhong, W.-l., Zhang, J., & Chen, W.-n. (2007). A Novel Discrete Particle Swarm Optimization to Solve Traveling Salesman Problem. *Evolutionary Computation*, 3283-3287.