Comparison of techniques for solving vehicle routing problems

Hlompho Napo Qhomane
(Student No: 468489)

Supervised by: Dr. Terry-Leigh Oliphant

School of Computer Science and Applied Mathematics,
University of the Witwatersrand, Johannesburg, South Africa.

This dissertation is submitted in fulfillment of the requirements for the degree of *Master of Science*.

07 August 2018

UNIVERSITY OF THE
WITWATERSRAND,
JOHANNESBURG

Declaration

I, Hlompho Napo Qhomane, declare that this Dissertation is my own original work, I also confirm that:

- This work was done wholly while in candidature for a Master degree at the University of the Witwatersrand.
- Where any part of this Dissertation has previously been submitted for a degree or any other qualification at this University or any other institution, I clearly stated.
- Where I have consulted the published work of others and quoted their work, I have also clearly stated.
- I have acknowledged all main sources of help.

Signed: _____

Date: _____

Abstract

Abstract. The vehicle routing problem is a common combinatorial optimization, which is modelled to determine the best set routes to deploy a fleet of vehicles to customers, in order to deliver or collect goods efficiently. The vehicle routing problem has rich applications in design and management of distribution systems. Many combinatorial optimization algorithms which have been developed, were inspired through the study of vehicle routing problems. Despite the literature on vehicle routing problems, the existing techniques fail to perform well when n (the number of variables defining the problem) is very large, i.e., when $n > 50$. In this dissertation, we survey exact and inexact methods to solve large problems. Our attention is on the capacitated vehicle routing problem. For exact methods, we investigate only the Cutting Planes method which has recently been used in conjunction with other combinatorial optimization problem algorithms (like the Branch and Bound method) to solve large problems. In this investigation, we study the polyhedral structure of the capacitated vehicle routing problem. We compare two metaheuristics, viz., the Genetic Algorithm and the Ant Colony Optimization. In the genetic algorithm, we study the effect of four different crossover operators. Numerical results are presented and conclusion are drawn, based on our findings.

Acknowledgements

I am grateful to God, Almighty Father, Creator of all things. Thank you so much for granting me this opportunity to finish my degree. 'Na esale o mphihlela mohau oona, ke maketse feela.

I would like to express my sincere gratitude to my supervisor, Dr. Terry-Leigh Oliphant. She has put a tremendous amount of time and effort to supervise this dissertation. She was patient with me and kept me motivated. I have learned some crucial research skills from her insightful suggestions and knowledge.

To my parents, Ntate Eric Moeletsi Qhomane and Mme Ruth Mamohlomi Qhomane, I cannot fully express my gratitude. Thank you for the love, support, encouragement and patience that you have shown over the years. To my siblings, Mothusi, Khauhelo and Kananelo, thank you for the support. A big thank you to the rest of my family. A special thank you to Mme Morongoe Florence Makote, who, although no longer with us, played a big role in my life and got to see me start my Masters degree. Thank you for the love, advice and sympathetic ear.

A special thank you to the Free State Department of Education for their financial support towards my research.

Lastly, my special gratitude goes to the School of Computer Science and Applied Mathematics and their supporting staff members. To my colleagues and friends you are not forgotten, let us keep on encouraging one another other. I am grateful to all those I have had the pleasure to work with during this dissertation.

Contents

Declaration of Authorship i

Abstract ii

Acknowledgements iii

Contents iv

List of Figures vi

List of Tables vii

1 Introduction 1
- 1.1 Background . . . 1
- 1.2 Application . . . 3
- 1.3 Exact Methods and Metaheuristics . . . 4
- 1.4 Motivation . . . 6
- 1.5 Organization of the Dissertation . . . 6

2 Exact Methods: Cutting planes 8
- 2.1 Overview . . . 8
  - 2.1.1 Basic definitions . . . 10
- 2.2 Pure Integer Cutting Planes . . . 11
  - 2.2.1 Problem formulation and algorithm . . . 11
  - 2.2.2 Gomory's Cutting Plane Algorithm . . . 13
- 2.3 Cutting Planes Method Applied to the TSP . . . 17
  - 2.3.1 Mathematical formulation . . . 18
  - 2.3.2 The Complexity of the TSP . . . 20
  - 2.3.3 The TSP Polytope . . . 22
  - 2.3.4 TSP Polyhedron . . . 23
  - 2.3.5 Valid Inequalities defining facets to the STSP . . . 24
  - 2.3.6 The STSP Cutting Planes . . . 32
- 2.4 Cutting Planes Method Applied to CVRP . . . 33
  - 2.4.1 The CVRP Polytope . . . 35
  - 2.4.2 Inequalities from Capacity Constraints . . . 36

2.4.3 Inequalities from Routing Constraints . . . 36
2.4.4 Summary . . . 38

3 Metaheuristics: Genetic Algorithm 39
3.1 The main features of the Genetic Algorithm (GA) . . . 39
3.1.1 Biological Optimization . . . 40
3.1.2 The Genetic Algorithm . . . 41
3.2 Genetic Algorithm for CVRP's . . . 43
3.2.1 Problem Description and Mathematical Model . . . 43
3.2.2 Chromosome Representation . . . 44
3.2.3 Fitness Function . . . 46
3.2.4 Selection Operator . . . 46
3.2.5 Crossover Operator . . . 48
3.2.6 Mutation . . . 52

4 Metaheuristics: Ant Colony Optimization 53
4.1 The main features of the Ant Colony Algorithm . . . 53
4.1.1 Real Ants - The double bridge experiment . . . 54
4.1.2 Artificial Ants . . . 56
4.2 ACO Mathematical Formulation and Algorithm . . . 58
4.2.1 Travelling Salesman Problem using ACO . . . 59
4.2.2 Mathematical formulation . . . 59
4.2.3 The ACO Metaheuristic . . . 61
4.2.4 ACO Implementation . . . 65
4.2.5 ACO for Capacitated vehicle routing problem . . . 69
4.2.6 Proposed Solution Model . . . 71

5 Numerical Results and Analysis 74
5.1 Comparison and Analysis of Results . . . 74
5.1.1 Results obtained using the Genetic Algorithm . . . 74
5.1.2 Results obtained using the Ant colony Optimization . . . 78
5.1.3 Analysis . . . 80

6 Conclusion 84
6.1 Future work . . . 85

Bibliography 86

List of Figures

1.1 Solution methodologies of TSP and VRP, by Ganesh et al. [33] 6

2.1 Polytope representation of Example 1. 17
2.2 The LHS image (a), $S = 3$ and the graph is connected. The RHS image (b), $S = 3$ and it is not connected and has as infeasible solution. 19
2.3 The LHS image (a), has a feasible solution, that is not a tour. The RHS image (b), is Figure (a) after applying the subtour elimination constraint . 20
2.4 A Comb with handle H and teeth T_i. 25
2.5 Handle and teeth of a clique -tree 26
2.6 The existing classes of handle-tooth-cutset inequalities [55]. 27
2.7 A K-path configuration 30
2.8 The bicycle inequality 30
2.9 The wheelbarrow inequality 30

3.1 (a) Figure on the left represents the feasible solution as a TSP, capacity restrictions not considered. (b) The figure on the right represents the desired feasible solution for the CVRP, after capacity restriction are considered 45

4.1 Experimental setup for the double bridge experiment. (a) The left image has equal branch lengths. (b) The right image has different branch lengths. ([28], 2004, p.3) 55
4.2 (a) Image on the left has a feasible solution, all customers are visited. (b) The image on the right has an feasible solution, all cars are used up, but some customers are not visited 72

5.1 Percentage deviation from the optimum for different genetic operators in the GA, as per Table 5.1 76
5.2 Percentage deviation from the optimum for different genetic operators in the GA, as per Table 5.2 76
5.3 Percentage deviation from the optimum for the different genetic operator implementations in Table 5.3. 78
5.4 Percentage deviation from the optimum for ACO, as per Table 5.4 79
5.5 Percentage deviation from the optimum for ACO, as per Table 5.6 80
5.6 CPU time required for convergence of the GA and ACO implementations 81
5.7 Percentage deviations from the optimum for the GA and ACO implementations. 82
5.8 Percentage deviations from the optimum for the GA and ACO implementations. 82

List of Tables

2.1 **Optimal Simplex Tableau** 13
2.2 *Initial simplex tableau* 15
2.3 *Optimal simplex tableau* 15
2.4 *New simplex tableau* 16
2.5 *Optimal simplex tableau* 16
2.6 *Final optimal simplex Tableau* 17

3.1 Representation of an individual 45

5.1 Solutions obtained for the GA without mutation 75
5.2 Solutions obtained for the GA without mutation 75
5.3 Solutions obtained without mutation 77
5.4 Computational results of the CVRP instance 79
5.5 % dev. comparison 79
5.6 Computational results of the CVRP instance using ACO 80

Chapter 1

Introduction

The purpose of this chapter is to provide a brief discussion about this dissertation. We will introduce the reader to the vehicle routing problem (VRP) by discussing its background, the different variations of the problem, and its significance. In section 1.3, we discuss the difference between exact methods and metaheuristics, and diagrammatically illustrate the existing algorithms within each method. We present the motivation for this work, in Section 1.4, and conclude with the layout for the rest of this dissertation, in Section 1.5.

1.1 Background

Optimization problems are usually classified under two categories; continuous optimization and discrete optimization. In discrete optimization, there exists a finite set S of feasible solutions which is usually a set of subsets of a finite ground set E. The objective is to optimize a real valued objective function f on S. The set S may contain all subsets of E ($2^{|E|}$), in which case enumeration across all sets is impractical in the search of an optimal subset. The challenge is, therefore, to find better means of searching the feasible space to find the optimal solution. We refer to such problems as combinatorial optimization problems (COPs). Ibaraki [47] provides the following necessary, but incomplete description of the general COP:

$$\text{Minimize} \quad f(x), \tag{1.1}$$

$$\text{subject to} \quad x \in S, \tag{1.2}$$

where $S \subset Z^n, n \in Z$, and $f : X \to Z$, where Z is the set of integers, and $Z^n = Z \times Z \times \cdots \times Z$, (is the set of n-vectors with integer components). In this

dissertation, however, the problem we are concerned with is classified as a subset of COPs. VRPs are integer programming problems (IPs), which belong to the subset of COPs. Generally, they are described using a complete graph for undirected (symmetric) problems or by directed (asymmetric) graphs. In the latter case, the graphs may not be complete.

A single universally accepted definition for VRPs does not exist because of the diversity of constraints encountered in real life. Ignoring these constraints Dantzig and Ramser [20], defined the classical VRP as follows:

Let $G = (V, A)$ be a graph where $V = \{v_0, v_1, \cdots, v_n\}$ is the vertex set, and $A = \{(v_i, v_j) : v_i, v_j \in V, i \neq j\}$ is the arc set. The node v_0 represents the depot, and the remaining nodes are customers. A is associated with a cost matrix c_{ij} or a travel time matrix t_{ij}. If A is symmetric, then the VRP is an undirected graph represented by $G = (V, E)$, where $E = \{(v_i, v_j) : v_i, v_j \in V, i < j\}$ is an edge set. Each customer i has a non-negative demand q_i and a service time t_i. A fleet of vehicles with unrestricted capacity is also based at the depot. The number of vehicles are either known in advance or can be treated as decision variables. The VRP consists of designing a set of at most m delivery or collection routes such that:

- Each route starts and ends at the depot.
- Each customer is visited exactly once, by exactly one vehicle.
- The total duration of each route does not exceed a present limit D, and the total routing cost is minimized.

In reality, many more constraints exist in addition to those we have mentioned in the formulation of the VRPs. Some include constraints which describe fatigue rules and driver breaks, vehicle reuse, the ability to change vehicle characteristics, the use of limited resources, and the variable loading or unloading times. Toth et al.[74], also highlights other factors in real life which may affect the routing problem.

The VRP can be viewed as the combination of two well known problems; the traveling salesman problem (TSP) and the bin packing problem (BPP). The difficulty in solving TSPs and BPPs increases with the problem size and these problems are $NP-complete$[1]. Nevertheless, methods like the decomposition based optimization techniques, involving the *relaxation* of one or more of the underlying structures of the problem, have proved to work, in searching for a solution. By the relaxation of a problem, we mean expansion

[1] NP stands for nondeterministic polynomial time. This is a computational problem which is as hard as any reasonable problem.

of the feasible space. This is done by dropping certain constraints in the problem. It is worthwhile noting that even after relaxation, VRPs can still be $NP - complete$.

1.2 Application

VRPs have rich application across various fields in the real world. For instance, the capacitated vehicle routing problem (CVRP), which is our main focus in this dissertation is seen as the core to logistics planning. When Dantzig et al.[20] first formulated the VRP, it was through the inspiration of studying a real world problem on how to find the optimum routing of gasoline delivery trucks between a depot and a large number of service stations. Some applications to the VRP include the dynamic fleet management, vendor managed distribution system, bank cheque collection, couriers, rescue and repair service companies, dial and ride systems, emergency services, taxi cab services, and newspaper distribution. The practical significance of the VRP is highlighted by its application to many real life problems.

1.2.1 Types of VRPs

VRPs are usually classified by the type of constraints they have in addition to the classical formulation mentioned above. Below we list some of these differences.

(i) *Capacitated Vehicle Routing Problem* (CVRP)
The CVRP has a fixed fleet of delivery vehicles of the same capacity deployed for a single commodity from a common depot, and the aim is to minimize the usage of vehicles and the total distance traveled.

(ii) *Vehicle Routing Problem with Time Window* (VRPTW).
The VRPTW involves a time restriction by which customers are served in this time window. Every customer i is given a time window interval (e_i, u_i); the vehicle can not arrive earlier than time e_i and no later than time u_i. The assumption is made that if a vehicle arrives too early at a customer location, then it will wait until the customer time window opens.

(iii) *Vehicle Routing Problem with Backhauls.* (VRPB)
In this extension of the VRP, the set of customers is partitioned into two sets. The first set is the set of customers that require a given quantity of products, and the second is the set of customers from which a given quantity of products must be picked up. Courier delivery companies are a simple example of this.

(iv) *Distance-Constraint Vehicle Routing Problem* (DCVRP)
In the DCVRP, each route has a maximum length (or time) constraint instead of capacity

constraint. This could be thought of as the restriction of the maximum distance that can be traveled by each car.

(v)*Multi-Depot Vehicle Routing Problem* (MDVRP)
The MDVRP has multiple depots as the name suggest. By multiple depots, businesses can increase efficiency by allocating goods to the different depots according to the demand of customers near a specific depot. In doing this, vehicles need not travel long distances.

(vi) *Vehicle routing problem with pick-up and delivering* (VRPPD)
In the VRPPD, a vehicle is allowed to deliver and collect goods. These include returned goods from unsatisfied customers or goods picked up to be delivered elsewhere. Here, goods must be picked up while the vehicle is still delivering other goods and all goods must fit in the same vehicle. Righini [71], provides approximation algorithms for this type of problem.

(vii) *Split Delivery Vehicle Routing Problem* (SDVRP)
In this extension of the VRP, a customer can be appointed to more than one delivery vehicle if it reduces the overall costs. This modification is also important when the order from the customer exceeds the vehicle capacity.

(viii) *Stochastic vehicle routing problem* (SVRP)
The stochastic VRP contain more random components. There are three types of stochastic VRPs:

- Stochastic customer: the customer V_i is present with probability p_i and absent with probability $1 - p_i$.
- Stochastic demand: the demand d_i of each customer is a random variable.
- Stochastic time: the stochastic unit-travel times t_{ij} are random variables.

(ix) *Periodic vehicle routing problem* (PVRP)
For periodic VRPs, the planning period is not only for one day but for a number of days, i.e., delivery is generalized to M number of days, where $M > 1$, as opposed to a single day like in the classical VRP.

1.3 Exact Methods and Metaheuristics

Solution methods for optimization problems can be classified as exact or inexact methods i.e., algorithms which yield exact solutions or approximate solutions,

respectively. Exact methods are well known to be exhaustive and impractical to use in the search for a solution of a COP when n is very large. This is because most COPs are NP-hard and the complexity of the problem increases exponentially as n increase. Alternatively, one can use approximation methods (heuristics) when exact methods fail. The word *heuristic* refers to an approach of solving problems. Most heuristics are designed to be problem dependent, and they do not guarantee an exact solution, but usually provide a good acceptable approximate to the exact solution, within reasonable computational time. When an optimization algorithm is problem independent, then it is referred to as *metaheuristic*. These techniques are often used when very little information about the problem at hand is known. Metaheuristics are implemented by employing some randomness to find an optimal solution to a problem. Most algorithms used in optimization make strong assumptions about the nature of the objective function f. These includes assumptions about the existence of the first and second derivatives of f. On the other hand, with metaheuristics weaker assumptions or no assumptions are made. Differentiability, linearity, convexity, and Lipschitz continuity are not really of importance in metaheuristics. This suggests that metaheuristics can be inefficient to use and should be used as the last resort. However, a large number of significant, large scale problems require metaheuristics. Sean Luke [56] explains that most metaheuristics exploit a heuristic belief about the space of candidate solutions. This includes the assumption that small changes will in the structure of the problem, result in small, well-behaved changes in the quality of the solution. Luke [56], calls this the *central defining features of metaheuristics*. According to the existing literature, the only methods capable of solving large VRPs ($>= 50$) are heuristic and metaheuristic methods. Very few exact algorithms exist for solving large scale VRPs. The $K-tree$ method proposed by Fisher [30], for instance, succeeded in solving a problem with 71 customers, however there are still instances of smaller cases not solved. Only a small amount of VRPs can be solved to optimality using exact methods. A good summary of the different solution methodologies for VRPs is shown in Figure 1.1.

An attractive feature in optimization algorithms is that different algorithms can easily be combined to form new, often improved algorithms which yield better performance. It is common practice to incorporate heuristics or local search techniques in metaheuristics to improve the quality of the solution of the algorithm. This feature in referred to as hybridization. Hybridization is used in algorithms investigated in this dissertation. We will also investigate two metaheuristics, viz., the genetic algorithm (GA) and ant colony optimization (ACO). We also investigate the performance of the Cutting Planes method. Although the Cutting Planes method is not shown in Figure 1.1. it can be classified under exact techniques.

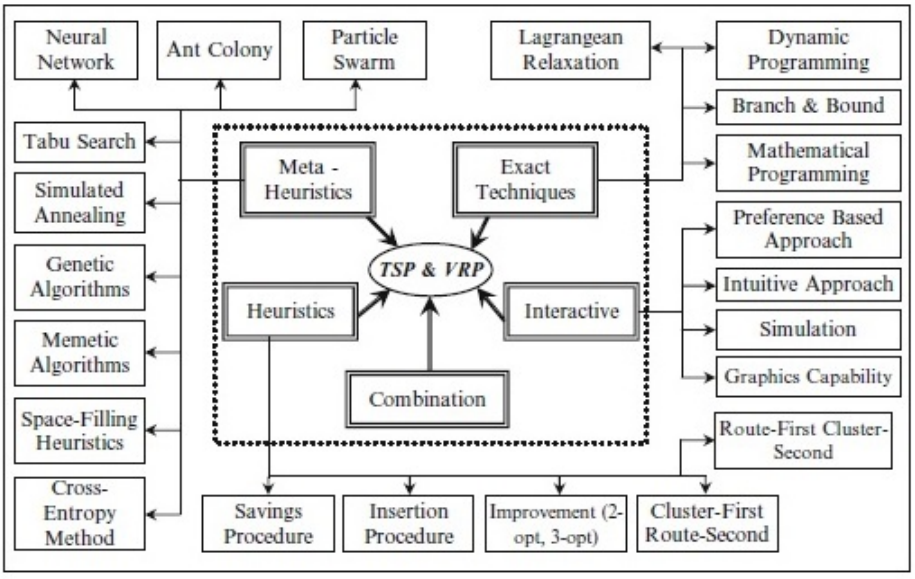

FIGURE 1.1: Solution methodologies of TSP and VRP, by Ganesh et al. [33]

1.4 Motivation

The focus of this dissertation is to understand the theoretical background of CVRPs. Metaheuristics have increased the scope of benchmark problems that can be solved to optimality. In despite of this, solving the same problems is almost impossible especially for large sized problems and also because very little is known about the CVRP polytope. The branch and bound method [51] is the most successful method used to solve CVRPs. Recently Padberg and Rinaldi [65], and Balas et al. [5] showed that generating new cuts, using Cutting Planes, can aid the branch and bound to extend the scope of problems that are solvable to optimality using the branch and cut method.

We also review two existing metaheuristics, the GA and ACO, by carrying out numerical testing on the performance of these methods. Both methods are implemented in Matlab. In order to do a comparison of these techniques, we use benchmark problems from the CVRP library [1].

1.5 Organization of the Dissertation

The rest of the dissertation is organized as follows. In Chapter 2, theoretical concepts pertaining to Cutting Planes are discussed. We begin with gomory Cutting Planes [35], which is followed by a review on polyhedral theory. This is done for TSPs, and then CVRPs, to highlight the complexity of the Cutting Planes method, when applied to CVRPs. In Chapter 3, we introduce the GA and discuss how it is applied to CVRPs.

We also investigate the effect of four recombination operators to CVRP. In Chapter 4, we investigate the ACO [25] by providing a detailed review and its successive improvements. Numerical results based on the methods discussed in Chapters 3 and 4, are presented in Chapter 5. The summary of the dissertation and possible future research options are discussed in Chapter 6.

Chapter 2

Exact Methods: Cutting planes

2.1 Overview

CVRPs are usually expressed as a linear programming problem (LP). Known LP methods can be used in an attempt to solve CVRPs. However, when the size of the problem increases, the number of inequalities defining the CVRP increase exponentially, resulting in too many inequalities to define. LP solvers are thus unable to solve the resultant CVRP. In despite of this, iterative approximation methods, which is a process characterized by relaxation, can be used to solve the CVRPs. For the symmetric capacitated vehicle routing problem (SCVRP), some relaxations may include the removal of capacity constraints or the subtour elimination constraint. This results in the Assignment problem or the B-matching problem. In this section, the method we will focus on looks at linear relaxation of integer programming problems (IPs), which results from dropping the integrality condition of the original problem.
In 1960, Land and Doig [51], proposed an automatic method for solving discrete programming problems, called the Branch and Bound method (B&B). This method, because of its success, has been widely used in solving CVRPs. In B&B, a linear relaxation of the problem is first solved. If the solution obtained is integral, then the problem is solved and the algorithm B & B terminates with the optimal solution. If the solution obtained is not integral, one fractional variable is selected from which two LPs will be solved each with an additional constraint; an upper bound and a lower bound of the fractional variable respectively. In B&B, if an optimum exists then the solution will be obtained in a finite number of iterations. The number of iterations usually tends to increase exponentially for large problems. In this dissertation, we explore independently, a different approach to B&B algorithm, called the *Cutting Planes* technique. This method can also solve LPs.

We will consider different relaxation techniques in this study, because each relaxation yields different results. When a relaxed enriched LP or a simple relaxed LP has many inequalities, the cutting planes method works as follows:
Let $h \geq 0$, be the subset of a number of constraints of the LP. If the optimal solution $x_h \in LP(h)$ is feasible to integer programming problems (IP), then we stop, as an optimal solution is reached. If the optimal solution is not integer-feasible, then an algorithm called the *separation algorithm* is used to find at least one constraint, if it exists, from the LP which is violated by x_h. The new constraint is added to $LP(h)$, and $LP(h+1)$ is solved to optimality. The new solution to $LP(h+1)$ forms a lower bound (if minimizing) to the IP:

$$Z_{LP(h)} \leq Z_{LP(h+1)} \leq Z_{LP} \leq Z_{IP}. \tag{2.1}$$

This process continues as long as x_h is not feasible. In practice, separation algorithms may not be exact, i.e., they may not return any violated inequalities even if some still exist. If cutting planes method has terminated without having reached an optimal solution, then the cutting planes can be combined with branching, using the *Branch and Cut* method (B&C) [65]. This is done to increase the probability of locating an optimal solution.

Cutting planes method may be unsuccessful in finding a solution to a given problem or perform poorly in the following situations:

1. When the linear program has too many constraints,
2. When the separation procedure has too many procedures,
3. When the cutting plane procedure does not produce good cuts for the specific type of problem.

While there is no remedy for case 1, and case 2, case 1 can be resolved by identifying the active and inactive constraints.

To understand the cutting planes method, we first study how the method can be applied to general IPs using the basic Gomory Cutting planes. Since the CVRP is the extension of the TSP, we investigate cutting planes in the context of the TSP before investigating cutting planes on CVRPs method.

2.1.1 Basic definitions

In the following subsection, we present some definitions which are vital to the understanding of the cutting planes.

Definition 1.1 (Convex set). A set V is convex if the line segment between any two points in V lies in V. i.e., $\forall x_1, x_2 \in V$ and $\forall \theta \in [0,1]$, we have $\theta x_1 + (1-\theta)\, x_2 \in V$.

This definition of the convex set can be generalized from 2 points to any number of points n.

Definition 1.2 (Convex Combination). A point of the form $\theta_1 x_1 + \theta_2 x_2 + \cdots + \theta_t x_t$ where $\theta_i \geq 0, i = 1, \cdots, t, \sum_{i=1}^{t} \theta_i = 1$, and $x_1, x_2, \cdots, x_t \in V$ is called a convex combination.

From definition 1.2, we can say that a set V is convex *iff* any convex combination of points is in V.

Definition 1.3 (Convex Hull). A convex hull of a set V is the smallest set of all convex combinations of the points in V. $Conv(V) = \{\theta_1 x_1 + \cdots + \theta_t x_t | t \geq 1, x_1, \cdots, x_t \in V, \sum_{i=1}^{t} \theta_i = 1, \theta_1, \cdots, \theta_t \in \Re \geq 0\}$.

Definition 1.4 (Hyperplane). A hyperplane is a set of the form $\{x \in \Re^n | a^T x = b\}, a \in \Re^n, b \in \Re$. This can be seen as a solution set of a nontrival linear equation among the components of x. A hyperplane divides $\Re^n$ into two spaces.

Definition 1.5 (Halfspace). A halfspace in $\Re^n$ is a set of the form $\{x \in \Re^n | a^T x \leq b\}, a \in \Re^n, b \in \Re$.

Definition 1.6 (Polyhedron). A polyhedron $P \subseteq \Re^n$ is a set of the form $\{x \in \Re^n | Ax \leq b\}$ for some $A \in \Re^{m \times n}$ and $b \in \Re$.

NB. It is important to see that this definition (1.6) is the intersection of finitely many halfspaces and hyperplanes or as the solution set of a finite number of inequalities. A *polytope* is a bounded polyhedron.

Definition 1.7 (Valid Inequality). An inequality $a^T x \leq b$ is called *valid* w.r.t. $S \subseteq \Re^n$ if S is contained in the halfspace defined by this inequality. i.e., $S \subseteq \{x \in \Re^n | a^T x \leq b\}$

Definition 1.8 (Face). $F \subseteq P$ is called a *Face* of P, if $\exists \quad a^T x \leq a_0$ valid to P, such that $F = \{x \in P | a^T x = a_0\}$.

These basic definitions are essential in the study of cutting planes. Lastly, the integer linear formulation of an LP for a maximization problem can be stated as follows:

Maximize

$$z = \sum_{j=1}^{k} c_j x_j, \tag{2.2}$$

subject to:

$$\sum_{j=1}^{k} a_{ij} x_j \leq b_i, \quad for \quad i = 1, 2, ..., m. \tag{2.3}$$

$$x_j \geq 0, \quad for \quad j = 1, 2, ..., k. \tag{2.4}$$

$$x_j \in Z, \quad for \quad j = 1, 2, ..., k. \tag{2.5}$$

A minimization problem can be converted into a maximization problem by multiplying the objective function z by -1, since:

$$-\text{ Minimum of } z = \text{Maximum of } (-z).$$

2.2 Pure Integer Cutting Planes

The use of cutting planes was introduced by Ralph Gomory [35] in 1958. Since then, different cutting planes have been studied extensively by COP practitioners. In this subsection, we present a general cutting planes for pure IP's. This method is independent of the problem structure and serves as a basis to understanding the general notion of cutting planes before we investigate cutting planes for CVRPs.

2.2.1 Problem formulation and algorithm

The following proposition serves as the basis to the method presented below.

Proposition 1. Any closed convex set V has a linear inequality description given by,

$$V = \cap_{j \in J} \{x | a_j^T x \leq b_j\},$$

where j is the index set that may be infinite. The above results can be found in convex analysis and optimization books. The idea behind cutting planes (CPs) is given in Algorithm 1 below.

In **Algorithm** 1 below, step 2 refers to the start of the while loop and step 3 states that the LP should be given as a maximization problem. The variable x^* refers to the

Algorithm 1 Generic cutting plane algorithm

1: Represent the problem as an LP, and go to 3.
2: **repeat**
3: Solve the LP $max\{c^T x : x \in P\}$,
4: **if** x^* is integral **then**
5: x^* is the optimal solution to the LP.
6: **else**
7: Find an inequality $w^T x \leq d$ such that $w^T x^* > d$.
8: **if** such an inequality $w^T x \leq d$ cutting off x^* is found **then**
9: Add the inequality to the system, that is, set $P = P \cap \{x : w^T x \leq d\}$, then go to step 3
10: **else**
11: We do not have the optimal. However, we have refined the original formulation. **stop**
12: **end if**
13: **end if**
14: **stop**

optimal solution to the LP, x,w are vectors in the convex set P and $d \in \Re$. Step 7 is the *separation procedure* step, and is usually the most difficult step to perform. This is because the polytopes structure of most problems is not fully known. We shall now give a fully detailed explanation using the Simplex algorithm to demonstrate this procedure.

Let $P_I = conv\,(P \cap \mathbf{Z})$ be the integral hull of P. When implementing cutting planes, we want to find the full description of P_I which enables us to solve:

$$max\{c^T x : Ax = b, x \geq 0, b \geq 0, x \in Z^n\},$$

the equivalence of equations (2.2)−(2.5). However, to find such a description is usually difficult. A general approach to tackling this problem is to approximate P_I by a rational polyhedron P which contains P_I, and for which the separation problem can be decided efficiently. We start off by solving the relaxed optimization model. If the extreme point of the feasible region has a fractional component, then the integrality condition is violated. To circumvent this, we solve the integer program as a sequence of linear programs and at each step we introduce a valid inequality (cutting plane), which tightens the relaxed model until optimality is reached. By finding such an inequality which removes portions of the region space, it can be shown that this will lead to the optimal integer solution (Fitness of the Gomory's Cutting-Plane method theorem) [54].

2.2.2 Gomory's Cutting Plane Algorithm

Consider the ILP mentioned above in (2.2) – (2.5). By setting the objective function to z we have:

$$z = \sum_{j=1}^{k} c_j x_{ij}, \tag{2.6}$$

where $i = 1, \cdots, m$. We define non-negative slack variables x_{k+i}. These variables are added to (2.3) to transform the ILP to standard form. The new constraints become:

$$\sum_{j=1}^{k} a_{ij} x_j + x_{k+i} = b_i. \tag{2.7}$$

The slack variables as well as the objective z take the integral form since A and b are both integers. Once the problem is set to standard form we now assemble the simplex tableau from which row operations will be performed to obtain the optimal tableau. To assemble the standard tableau let:

$$a_{i,k+j} := \begin{cases} 1 & \text{if} \quad i = j; \\ 0 & \text{if} \quad i \neq j; \end{cases} \tag{2.8}$$

for $1 \leq i,\, j \leq m$, $b_0 := 0$ and $a_{i,0} := 0$ for $i \leq j \leq m$. Finally, for $0 \leq j \leq n$:

$$a_{0,j} := \begin{cases} 1 & \text{if} \quad j = 0, \\ -c_j & \text{if} \quad 1 \leq j \leq k, \\ 0 & \text{if} \quad k+1 \leq j \leq k+m. \end{cases} \tag{2.9}$$

The resulting optimal simplex tableau after performing row operations is shown in Table 2.1 below.

TABLE 2.1: **Optimal Simplex Tableau**

Basis	z	x_1	$\cdots$	x_j	$\cdots$	x_i	$\cdots$	x_n	b
z	1	$\bar{c_1}$		$\bar{c_j}$		$\bar{c_i}$		$\bar{c_n}$	$\bar{z}$
$\vdots$	0	$\vdots$		$\vdots$		0		$\vdots$	$\vdots$
x_i	0	$\bar{a_{i1}}$		$\bar{a_{ij}}$		1		$\bar{a_{in}}$	$\bar{b_i}$
$\vdots$	0	$\vdots$		$\vdots$		0		$\vdots$	$\vdots$

If we define $n := k + m$, then we can partition the index set $(0, 1, 2, ..., n)$ into 2 parts;

non-basic indices $\eta = (\eta_1, \eta_2, \cdots, \eta_k)$ and basic indices $\beta = (\beta_0, \beta_1, \beta_2, \cdots, \beta_n)$. Thus, $\sum_{j=1}^{k} a_{ij}x_j = b_i$ for $i = 0, 1, \cdots, m$ can be written as:

$$\sum_{j=0}^{m} a_{i\beta_j}x_{\beta_j} + \sum_{j=1}^{n-m} a_{i\eta_j}x_{\eta_j} = b_i, \tag{2.10}$$

for $i = 0, 1, \cdots, m$. The basic variables can be represented in terms of the non-basic variables as:

$$x_{\beta_i} + \sum_{j=1}^{n-m} \bar{a}_{\beta_i\eta_j}x_{\eta_j} = \quad x^*_{\beta_i}. \tag{2.11}$$

This equation represents row i in the simplex tableau. By incorporating the integrality condition of it, we obtain:

$$x_{\beta_i} + \sum_{j=1}^{n-m} \lfloor \bar{a}_{\beta_i\eta_j} \rfloor x_{\eta_j} \leq \quad \lfloor x^*_{\beta_i} \rfloor. \tag{2.12}$$

Note that the cutting plane inequality is violated by the basic solution at hand whenever $x^* \notin \mathbf{Z}$. By subtracting (2.11) from (2.12) we obtain:

$$\sum_{j=1}^{n-m} \left(\lfloor \bar{a}_{\beta_i\eta_j} \rfloor - \bar{a}_{\beta_i\eta_j} \right) x_{\eta_j} \leq \quad \lfloor x^*_{\beta_i} \rfloor - x^*_{\beta_i}. \tag{2.13}$$

Equation (2.13) is the Gomory-cut inequality. The procedure to systematically generate these planes in the form of an LP, is shown in Algorithm 2. To conclude, we demonstrate the Gomory cutting planes by use of an example. To indicate a pivot row and element in the simplex tableau we use "$\rightarrow$" and "[...]" respectively.

The Gomory cutting planes are illustrated with the use of an example:
Example 1.

$$\begin{aligned} \text{Maximize} \quad & z = x_1 + x_2, \\ \text{Subject to:} \quad & 2x_1 - x_2 \leq 6, \\ & -x_1 + 4x_2 \leq 16, \\ & x_{1,2} \geq 0, \quad x_{1,2} \in Z. \end{aligned} \tag{2.14}$$

Before we can solve the LP in (2.14), we have to convert it to standard form. We do so by adding two non-negative slack variables x_3, x_4 to the above constraints. Slack variables are added because both constraints are '$\leq$' constraints. A similar, but slightly more complicated procedure is used to convert '$\geq$' inequalities to standard form. In this work, we are only interested in '$\leq$' constraints. Variables in the objective function are all transposed to the right hand side before the initial simplex tableau is

Algorithm 2 Gomory's cutting plane algorithm

1: Represent the problem as an LP in canonical form, i.e., $max\{c^T x : Ax = b, x \geq 0, b \geq 0, x \in Z^n\}$, and go to 3.
2: **repeat**
3: Solve the LP $max\{c^T x : Ax = b, x \geq 0\}$
4: **if** x^* is integral **then**
5: x^* is the optimal solution to the LP and satisfies step 1. **Stop**
6: **else**
7: Find the basis integer with the largest fractional part as a consistent rule in the optimal LP tableau, i.e. copy a row from the tables. This is parametrized as follows:

$$x_i = \bar{b_i} - \sum_{j \in N} \bar{a_{ij}} x_j.$$

8: Generate the Gomory-cutting plane:

$$\sum_{j \in N} \left(\lfloor \bar{a}_{\beta_i \eta_j} \rfloor - \bar{a}_{\beta_i \eta_j} \right) x_{\eta_j} \quad \leq \quad \lfloor x^*_{\beta_i} \rfloor - x^*_{\beta_i}.$$

9: Add a new non- negative integer slack variable s to the cutting plane:

$$\sum_{j \in N} \left(\lfloor \bar{a}_{\beta_i \eta_j} \rfloor - \bar{a}_{\beta_i \eta_j} \right) x_{\eta_j} + s \quad = \quad \lfloor x^*_{\beta_i} \rfloor - x^*_{\beta_i}$$

$$s \geq 0$$

$$s \in Z$$

10: Append this cutting plane equation to the bottom of the optimal simplex tableau
11: **end if**
12: Go to Step 3, but solve the LP using the **Dual Simplex iteration**.
13: **stop** when for all $\bar{b_i} \in Z^n$

set up, see Table 2.2. Table 2.3 shows the results after performing row operations:

TABLE 2.2: *Initial simplex tableau*

Basis	z	x_1	x_2	x_3	x_4	b	
z	1	-1	-1	0	0	0	
x_3	0	[2]	-1	1	0	6	$\rightarrow$
x_4	0	-1	4	0	1	16	

TABLE 2.3: *Optimal simplex tableau*

Basis	z	x_1	x_2	x_3	x_4	b
z	1	0	0	$\frac{5}{7}$	$\frac{3}{7}$	$\frac{78}{7}$
x_1	0	1	0	$\frac{4}{7}$	$\frac{2}{14}$	$\frac{40}{7}$
x_2	0	0	1	$\frac{1}{7}$	$\frac{2}{7}$	$\frac{38}{7}$

Here, [2] is the pivot element, and $x_1 = 40/7, x_2 = 38/7$ are non-integer optimal solutions to the relaxed problem. To obtain the optimal solution for the integer problem, we generate a Gomory cutting plane inequality, using the row in the optimal simplex tableau

with the highest fractional b. In this instance, this is row 2:

$$x_1 + \frac{4}{7}x_3 + \frac{2}{14}x_4 = \frac{40}{7}. \tag{2.15}$$

When condition (2.12) is applied to this equation, the resulting Gomory cutting plane is given by:

$$x_1 + \lfloor\frac{4}{7}\rfloor x_3 + \lfloor\frac{2}{14}\rfloor x_4 \leq \quad \lfloor\frac{40}{7}\rfloor, \tag{2.16}$$

Since $x_3 = x_4 = 0$, and $40/7 = 5$, this reduces to:

$$x_1 \leq \quad 5. \tag{2.17}$$

The Gomory cutting plane, as stated in step 8 of **Algorithm 2**, results when (2.17) is subtracted from (2.15), i.e.:

$$-\frac{4}{7}x_3 - \frac{2}{14}x_4 \leq -\frac{5}{7}. \tag{2.18}$$

This inequality is one of the many inequalities which tighten the IP formulation of this problem. When Steps 9 and 10 of **Algorithm** 2 are performed, the resultant tableaus are given in Tables 2.4 and 2.5, respectively.

TABLE 2.4: *New simplex tableau*

Basis	z	x_1	x_2	x_3	x_4	x_5	b
z	1	0	0	$\frac{5}{7}$	$\frac{3}{7}$	0	$\frac{78}{7}$
x_1	0	1	0	$\frac{4}{7}$	$\frac{2}{14}$	0	$\frac{40}{7}$
x_2	0	0	1	$\frac{1}{7}$	$\frac{2}{7}$	0	$\frac{38}{7}$
x_5	0	0	0	$[-\frac{4}{7}]$	$-\frac{2}{14}$	1	$-\frac{5}{7}$
				$\downarrow$			

TABLE 2.5: *Optimal simplex tableau*

Basis	z	x_1	x_2	x_3	x_4	x_5	b
z	1	0	0	$\frac{1}{4}$	$\frac{5}{4}$	0	$\frac{41}{4}$
x_1	0	1	0	0	0	1	5
x_2	0	0	1	0	$\frac{1}{4}$	$\frac{1}{4}$	$\frac{21}{4}$
x_3	0	0	0	1	$\frac{1}{4}$	$-\frac{7}{4}$	$\frac{5}{4}$

This process is repeated until $\bar{b_i} \in Z^n$. The final optimal simplex tableau is shown in Table 2.6. The remaining cutting plane to this problem is given by equation 2.19 :

$$-\frac{1}{4}x_4 - \frac{1}{4}x_5 \leq \frac{1}{4}. \tag{2.19}$$

The two cutting planes (2.18) and (2.19), in the original problem are $x_1 \leq 5$ and $x_2 \leq 5$ respectively. To get these results, substitute x_3, x_4 into (2.18) and x_4 and x_5 into (2.19). The graphical representation to this problem is illustrated in Figure 2.1 below. The highlighted region shows the feasible solution of the relaxed problem and the feasible integer points are represented by black dots. In this problem there are only two cutting plane inequalities resulting in a horizontal and vertical cut. This is not always the case; in some instances we can get slanted cuts.

TABLE 2.6: *Final optimal simplex Tableau*

Basis	z	x_1	x_2	x_3	x_4	x_5	x_6	b
z	1	0	0	0	0	1	1	10
x_1	0	1	0	0	0	1	0	5
x_2	0	0	1	0	0	0	1	5
x_3	0	0	0	1	0	-2	1	1
x_4	0	0	0	0	1	1	-4	1

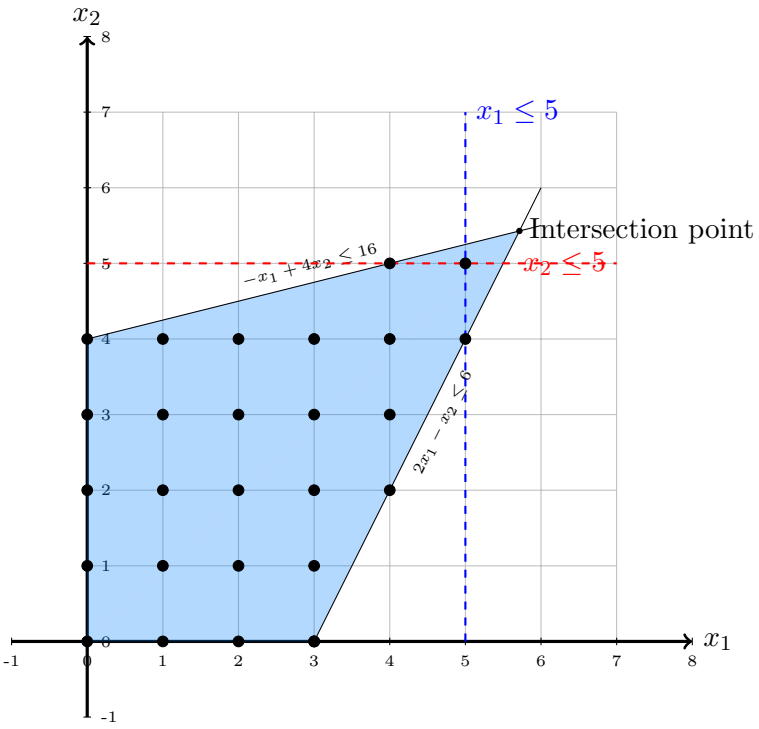

FIGURE 2.1: Polytope representation of Example 1.

Using Gomory cutting planes, we identified the inequalities needed to fully describe the integer polytope for the Example 1, in 2 iterations. Unfortunately, it is not as easy to find cutting planes for large problem instances. This is because, the number of iterations required to solve one problem to get its full description, is problem dependent. For CVRPs, more than one class of inequalities exists. Understanding the structure of the problem helps identify the type of inequalities involved. In the next section, we investigate cutting planes for the TSP since this is a special case of the VRP.

2.3 Cutting Planes Method Applied to the TSP

The TSP existed as early as the $1700's$ [6] during Euler and Vandermonde's time, although Rowan and Kirkman, in the 1800's may have been the first to consider

Hamiltonian cycles in the general context. The general form of the TSP was presented by Menger [53] in 1930. Since then, because of its complexity, it has been extensively studied and is one of the fundamental problems in Combinatorial Optimization. The TSP is seen as the basic problem to many routing problems. It is important that we introduce this section because the TSP is a special case of the VRP. The multiple TSP, where more than one salesman is allowed to be used in the solution, can be used to solve several types of VRPs [52, 53].

In the TSP we have a salesman and n cities which the salesman has to pass through before reaching their final destination. Beginning at the home base ($n = 1$), the salesman has to pass through all n cities ($n > 1$) exactly once before returning home. The objective of the problem is to minimize the total distance traveled by the salesman, for the entire trip. This can be interpreted as minimizing the cost of the whole trip or minimizing the travel time.

2.3.1 Mathematical formulation

There are many formulations to the TSP, just as there are many solution techniques available for solving TSPs. Each formulation has its own advantage and some formulations are more practical than others. For now we will only present the two index integer formulations for the asymmetric TSP (ATSP) formulation. We will then present the symmetric TSP (STSP) formulation. These formulations are also adopted in the VRP model. The symmetry of the STSP halves the number of possible solutions, while the ATSP yields paths which may not exist in both directions, resulting in a directed path.

The TSP can be defined on a complete undirected graph $G = (V, E)$ and by $G(V, A)$ on a directed symmetric graph. The variable $x_{ij} \in \{0, 1\}$ is called the decision variable. When $x_{ij} = 1$, the salesman goes directly from city i to city j, $x_{ij} = 0$ otherwise. c_{ij} is the corresponding distance to be minimized and it satisfies the triangle inequality i.e., $c_{ij} + c_{jk} \geq c_{ik}$, this is because every distance from i to j is a straight line. The asymmetric TSP can then be written as:

Minimize

$$\sum_{i=1}^{n} \sum_{j=1}^{n} c_{ij} x_{ij}, \tag{2.20}$$

Subject to:

$$\sum_{j=1}^{n} x_{ij} = 1 \qquad i = 1, \cdots, n, \tag{2.21}$$

$$\sum_{i=1}^{n} x_{ij} = 1 \qquad j = 1, \cdots, n, \tag{2.22}$$

$$\sum_{i \in S} \sum_{j \in S} x_{ij} \leq |S| - 1, \qquad 2 \leq |S| \leq n - 2. \tag{2.23}$$

where N is the number of customers and $S \subset N$. Equations (2.20)−(2.22) describes the well known *assignment problem*, hence this formulation is regarded as the relaxed problem to the TSP. The assignment problem combined with constraint (2.23) becomes the TSP. The inequality in (2.23) is known as the subtour elimination constraint, since it requires the whole trip to be connected. This means that for every $S \subset N$ subtour created, the number of edges connecting the cities should be 1 less than the number of cities in S. This results in an incomplete cycle; see Figure 2.2: (b):

FIGURE 2.2: The LHS image (a), $S = 3$ and the graph is connected. The RHS image (b), $S = 3$ and it is not connected and has as infeasible solution.

Inequality (2.23) can also be replaced by:

$$\sum_{i \in S} \sum_{j \in \bar{S}} x_{ij} \geq 2, \tag{2.24}$$

where the set $\left(S, \bar{S}\right)$, ($\bar{S}$ the complement of S) are the nontrivial partitions of $\{0, 1, \cdots, n\}$. The geometric interpretation of Equation (2.24) implies that every TSP solution must have at least one arc which points from S to its complement. In essence, S cannot be disconnected. For the visualization of how equation (2.24) works, refer to Figure 2.3 below. This constraint formulation, (2.24), was invented by Dantzig et al. [19].

There is a third formulation which differs from the two above. This formulation was invented by C.E. Miller et al. [60] in 1960, and it is called the MTZ formulation named after Miller, Tucker and Zemlim [60]. This subtour elimination constraint inequality blocks all tours not containing vertex 1 (home city), i.e., $S \subset V \setminus \{1\}$. The formulation is given as follows:

$$u_i - u_j + nx_{ij} \leq n-1, \quad for \quad i,j = 1,\cdots,n, \tag{2.25}$$

where u_i and u_j are uniquely defined arbitrary real numbers. Suppose that $1 \notin \{i_1, i_2, i_3, \cdots, i_k\}$ for $k \leq n$. Writing equation (2.25) for every arc of the subtour gives:

$$\begin{gathered} u_{i_1} - u_{i_2} + (n) \leq n-1, \\ u_{i_2} - u_{i_3} + (n) \leq n-1, \\ \vdots \\ u_{i_{k-1}} - u_{i_k} + (n) \leq n-1, \\ u_{i_k} - u_{i_1} + (n) \leq n-1. \end{gathered}$$

Summing up these constraints yields $k\,(n) \leq k\,(n-1)$, which is a contradiction. This proves that inequality (2.25) is valid.

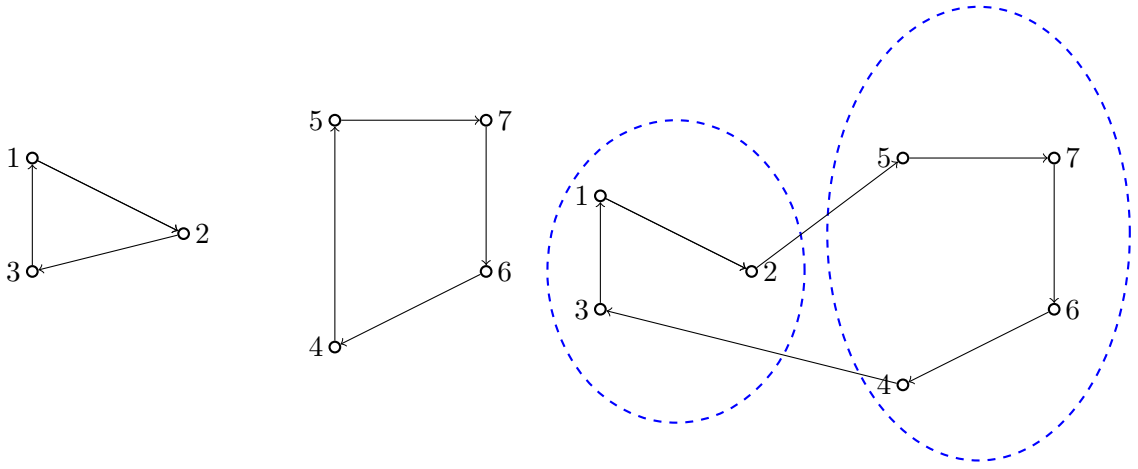

FIGURE 2.3: The LHS image (a), has a feasible solution, that is not a tour. The RHS image (b), is Figure (a) after applying the subtour elimination constraint

Most problems on large TSP's have been solved using the subtour formulation (2.25). No literature exists on computational studies using the pure MTZ formulation. Gabor Patiki [69] showed analytically that the MTZ has a much weaker LP relaxation, hence it is not popular.

2.3.2 The Complexity of the TSP

The formulation $(2.20)-(2.23)$ contains $n(n-1)$ binary variables, $2n$ degree constraints and $2^n - 2n - 2$ subtour eliminations constraints. This problem is known to be *NP-complete*. Papadimitrious et al. [68], listed two properties which any *NP-complete* problem satisfied:

1. No *NP-complete* problem can be solved by any known polynomial algorithm.

2. If there is a polynomial algorithm for any *NP-complete* problem, then there are polynomial algorithms for all *NP-complete* problems [1]. This proves that trying to solve the TSP using exact algorithms is hard.

Theorem 2.1. *The traveling salesman problem is NP-complete.*

Proof. First, we need to show that the TSP belongs to *NP*. To check the credibility of the tour, we can check if the tour contains each vertex only once. We then sum the cost of the edges and finally we check if the cost is minimum. This can done in polynomial time thus TSP belongs to NP.
Secondly, we prove that the TSP is $NP - hard$. One way to do this is to show that the Hamiltonian cycle $\leq_p$TSP, because it is well known that the Hamiltonian cycle is $NP - Hard$. We describe the reduction function. We construct the TSP from the graph $G = (A, E)$ which has a Hamiltonian cycle. We create a complete graph $G' = (V, E')$ where $E' = \{(i, j) : i, j \in V \quad \text{and} \quad i \neq j\}$ by defining the cost function as follows

$$t_{(i,j)} := \begin{cases} 0 & \text{if} \quad (i.j) \quad \in E, \\ 1 & \text{if} \quad (i.j) \quad \notin E. \end{cases} \tag{2.26}$$

Now suppose the Hamiltonian cycle h exists in G. The cost of the edges in h is 0 in G'. Thus if graph G has a Hamiltonian cycle then graph G' has a tour of 0 cost. Conversely, we assume that G' has a tour h' of cost at most 0. The cost of edges in E' are 0 and 1 by definition. So each edge must have a cost of 0 as the cost of h' is 0. We conclude that h' contains only edges in E. By showing this we have proven that G has a Hamiltonian cycle if and only if g' has a tour of cost at most 0. Thus TSP is $NP - complete$.

□

2.3.4 The Symmetric Traveling Salesman Problem

Let us now show the mathematical formulation of the symmetric traveling salesman problem (STSP). Let G be a complete graph with vertex set V and edge set E. For each edge $e \in E$, let c_e be the cost of traversing edge e. For any $S \subset V$, let $\delta(S)$ represent the set of edges in G with only one end vertex in S and $E(S)$ the set of edges with two end vertices in S. The variable x_e, such that $0 \leq x_e \leq 1$, takes values 1 if e is in the tour, and 0 otherwise. The variable $x(F)$, for any $F \subset E$ denotes $\sum_{e \in F} x_e$. The symmetric traveling salesman problem (STSP) can therefore be formulated as:
Minimize

$$\sum_{e \in E} x_e, \tag{2.27}$$

[1] These statements are still to be proved

subject to:

$$x(\delta(\{i\})) = 2 \quad \forall i \in V, \tag{2.28}$$

$$x(E(S)) \leq |S| - 1; \quad 3 \leq |S| \leq |V| - 3, \tag{2.29}$$

$$x_e \geq 0 \quad e \in E, \tag{2.30}$$

$$x \in Z^{|E|}. \tag{2.31}$$

The set $\delta(S)$ is usually referred to as the cut set, i.e., the edges which connect V to $V \setminus S$, and equation (2.28) is referred to as the degree constraint. In the study of exact algorithms, relaxations of the TSP are also important, as the implementation of the different relaxations yields different results.

Unlike the TSP mentioned in Section 2.3.1, obtaining the cutting planes for the STSP is non-trivial; it requires an understanding of the TSP polytope.

2.3.3 The TSP Polytope

Understanding the TSP polytope has lead to great progress in discovering new valid inequalities defining facets to the STSP. It is well known (also mentioned in definition 1.6) that there exists a finite set of linear equalities and linear inequalities defining the degree and facet constraints to the STSP. The optimal point to this Polytope is the optimal solution to the STSP. Removal of any of the elements in this set results in the relaxation of the TSP.
In the literature, we found that the TSP has both discrete and continuous feasible relaxation sets, and that every relaxation produced different results. Solving what is known as the subtour relaxation of the TSP is equivalent to solving Equations (2.27)−(2.30). Discrete relaxations can be partitioned into two types, i.e., *1-tree relaxation* and *2- matching relaxation*. [29, 48, 63].
Unfortunatly, the inequalities defining the facets of the STSP are too many and they grow exponentially with the size of the problem. Therefore, it is hard to determine all the inequalities defining the TSP, and the LP solver can not solve such large problems. This supports why we use relaxation methods. Moreover, M. Jünger et al. [48] explains that the full TSP description known in literature is only for TSP problems of dimension $n \leq 8$. For $n = 6$, 100 inequalities were used, for $n = 7$, 3437 inequalities were used and for $n = 8$, 494187 inqualities were used. Clearly, this is no good since most solvers can not handle many inequalities. It is also mathematically challenging to define all the facets of the TSP polytopes. Nevertheless, good relaxations still exist,

and the study of TSP polytope has shed some light into some facet defining inequalities that have helped researchers solve some large TSPs to optimality.

Grötschel [40] solved a 120 city TSP using only 96 inequalities defining facets. This motivates the reason to study facial structures of the polyhedron. Understanding other types of inequalities that strengthen this structure makes it possible to solve problems of higher cardinality. All such inequalities form the tightest formulation representation polytope of the TSP.

2.3.4 TSP Polyhedron

In the following subsection, we begin by defining some of the notation used for TSP polyhedrons. Some of the notation was derived from Lawler et al. [53]:

1. $G_n = (V, E)$ is a complete graph. A tour, $T \subseteq G_n$ is a tour if it is a cycle of length n in G_n, and a subtour, if it is a cycle of length $m \leq n$. We represent the set of all tours in G_n by T_n.

2. The characteristic vector or incidence vector denoted by x_e^T of a set $T \subseteq T_n$, is the vector indicating the membership or lack thereof, of an edge in T.

$$x_e^T = \begin{cases} 1 & \text{if} \quad e \in T, \\ 0 & \text{if} \quad e \notin T. \end{cases} \tag{2.32}$$

3. A denotes the incidence matrix of G_n. An incidence matrix A is such that each row represents the vertex of G_n and each column represents the corresponding edge in G_n.

4. The convex hull of the incidence vectors of all tours in G_n is denoted by Q_T^n

$$Q_T^n = conv\{x^T \in \Re^E | T \in T_n\}. \tag{2.33}$$

5. For any two subsets of E, U and W, whose intersections are not empty, we denote: $(U : W) = \{\ e \in E | e = [i, j] \text{ with } i \in U \text{ and } j \in W\ \}$.

6. An inequality $ax \leq a_0$ is called a *valid inequality* with respect to (w.r.t.) Q_T^n if $Q_T^n \subseteq \{x \in \Re^E | ax \leq a_0\}$. For an IP, this is an inequality that does not eliminate any integer feasible solution.

7. The *face* of Q_T^n is a valid inequality $ax \leq a_0$ s.t., $Q_T^n \cap \{x \in \Re^E | ax = a_0\} \neq Q_T^n$ yet it is not empty.

A tour, therefore, can be regarded as a point in the convex hull of vectors in $R^{|E|}$ satisfied by (2.26)−(2.30). The polytope formed by these points is represented by $STSP(n)$ and is called the symmetric traveling salesman polytope. A great amount of work has been done into understanding this polytope and its classes of valid and facet inducing inequalities. In the following subsection, we only highlight a few important discoveries that will lead us to introducing the CVRP polytope.

2.3.5 Valid Inequalities defining facets to the STSP

2.3.5.1 Trivial inequalities

The trivial inequality is of the form $x_e \geq 0$, where e belongs to the elements in the edge set. The trivial inequality is facet inducing and it is obvious that for any valid inequalities, $x_e \geq 0$. This result was proved by Grötschel and Padberg [41].

2.3.5.2 Subtour elimination inequalities

Subtour elimination constraints, discussed in section (2.31), are also valid inequalities which tighten the polyhedron structure and are satisfied by all tours. Intuitively, it is clear that for any valid tour, there should exist no subtour. Unfortunately, as we have already mentioned, there are a large number of these types of inequalities in the TSP, and this number increases with the size of the problem. Grötschel and Padberg [41], also showed that these inequalities are facet defining.

2.3.5.3 Comb inequalities

A comb (C) inequality is defined by H, and T such that, $H, T_i \subseteq V$, for $i = 1, 2, \cdots, k$, such that:

$$|H \cap T_i| \geq 1 \quad \text{for} \quad i = 1, \cdots, k, \tag{2.34}$$

$$|T_i - H| \geq 1 \quad \text{for} \quad i = 1, \cdots, k, \tag{2.35}$$

$$|T_i \cap T_j| = 0 \quad \text{for} \quad 1 \leq i < j \leq k, \tag{2.36}$$

$$k \quad is \quad \text{odd}. \tag{2.37}$$

The comb inequality is defined as:

$$x(E(H)) + \sum_{i=1}^{t} x(E(T_i)) \quad \leq \quad |H| + \sum_{i=1}^{t} |T_i| - \lceil \frac{3t}{2} \rceil. \tag{2.38}$$

In some texts this is equivalent to:

$$x(\delta(H)) + \sum_{i=1}^{t} x(\delta(T_i)) \geq 3t + 1. \tag{2.39}$$

Here, $x(E(H))$ describes the number of edges with both end vertices in H. Given say $d(H)$, the number of edges with only one end vertex in H, $x(E(H)) = |H| - d(H)/2$. Thus, we have:

$$x(E(H)) + \sum_{i=1}^{t} x(E(T_i)) = |h| + \sum_{i=1}^{t} |T_i| - \frac{1}{2}\left(d(H) + \sum_{i=1}^{t} d(T_i)\right). \tag{2.40}$$

To get equation (2.38), we have to show that $-\frac{1}{2}\left(d(H) + \sum_{i=1}^{t} d(T_i)\right)$ is greater than $3t$. The explanation is provided towards the end of this subsection when we give an intuitive explanation of the inequality. Grötschel and Padberg [42] proved that this inequality is valid for a STSP(n) with $n \geq 6$.

Figure (2.8) shows an example of a comb using sets. The comb inequality explained is a special case of a more general set of inequalities called *clique tree inequalities*. When $|H \cap T_i| = 1$, for $i = 1, \cdots, t$, then the comb inequality is referred to as the *Chvátal comb* inequality [16]. Furthermore, when (2.34) is an equality and $|T_i - H| = 1$, for $i = 1, \cdots, t$, the comb inequality reduces to a *2-matching inequalities* given by Edmonds et al. [29].

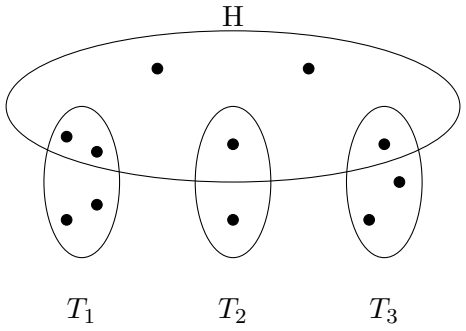

FIGURE 2.4: A Comb with handle H and teeth T_i.

2.3.5.4 Clique tree inequalities

Clique tree inequalities have also been proved to define facets, Grötschel and Pulleyblank [43]. These inequalities are more general than comb inequalities. To form a clique tree inequality, the teeth must contain at least one vertex which is not in any handle. Also, the teeth can intersect with more than one handle. Given p, the number of handles and t, the number of teeth, the clique tree inequality is defined as follows:

$$\sum_{j=1}^{p} x(\delta(H_j)) + \sum_{i=1}^{t} x(\delta(T_j)) \quad \geq \quad 2p + 3t - 1. \tag{2.41}$$

Notice that for $p = 0$ and $t = 1$, the above inequality becomes the connectivity constraint. Figure 2.9 shows a graphical representation of clique tree inequalities using sets.

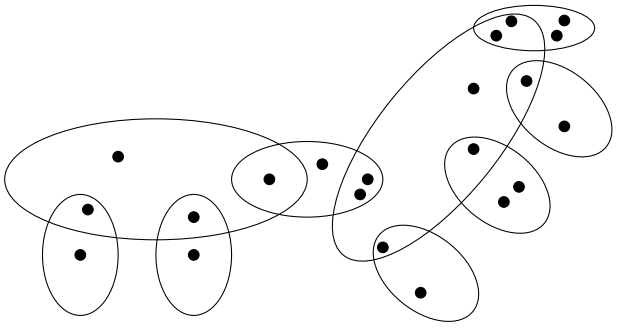

FIGURE 2.5: Handle and teeth of a clique -tree

2.3.5.5 Bipartition

A more general form of a clique tree inequality are Bipartitions by Boyd and Cunningham [9]. In Bipartitions, all the teeth and handles must be disjoint. The number of teeth intersecting a handle should be an odd number greater than or equal to 3. The interesting property of bipartitioning, is that a tooth may contain no vertex outside the intersection $H \cap T_i$, for $i = 1, \cdots, t$. Such teeth are referred to as degenerate. By letting d_j represent the number of handles intersecting tooth i, with $\beta_i = 1$, for degenerate teeth, we define:

$$\beta_i = \frac{d_i}{d_i - 1}, \tag{2.42}$$

as the non degenerate tooth. For Bipartition, the following inequality holds true:

$$\sum_{j=1}^{p} x(\delta(H_j)) + \sum_{i=1}^{t} \beta_i x(\delta(T_i)) \geq p + \sum_{i \in ND} (d_i + 2) + \sum_{i \in D} (2\beta_i - 1) d_i, \tag{2.43}$$

where ND and D describe or define the non degenerate and degenerate cases, respectively. An integral bipartition is when the teeth intersect with more than 3 handles and it is non degenerate.

Figure 2.10 illustrates the existing types of inequalities, listed by Letchford [55]. In this diagram, the inequality at the tail of the arrow is a special case of the inequality pointed by the arrow.

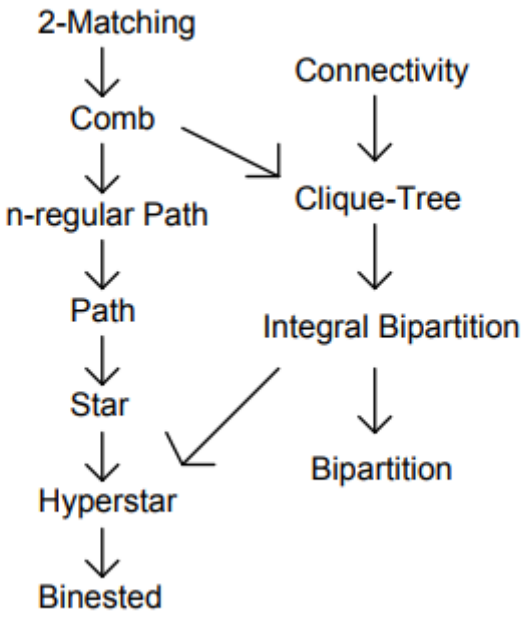

FIGURE 2.6: The existing classes of handle-tooth-cutset inequalities [55].

So far, we have only discussed a few of the inequalities listed on the right hand side of Figure 2.10. To describe path inequalities, we need to introduce the graphical traveling salesman problems (GTSP).We briefly present GTSPs. Cornuéjols, Fonlupt, and Naddef [18], were the first to publish a paper on the GTSP, though Fleischmann [31] did a similar study around the same time. The GTSP was introduced to overcome problems for which the shortest path in the TSP required an edge to be traversed or a node to be visited more than once, or when a graph $G(V, E)$ is not fully connected, and the Hamiltonian cycle does not exist. Therefore, the GTSP consists of finding a tour of G, whose length of a connected cycle going through *at least* once in each node is minimal. The graph G is associated with an integral vector $x = (x_e : e \in E)$, and x_e represents the number of times an edge e is traversed. The formulation can be given as follows:

$$Minimize \quad \sum_{e \in E} x_e, \tag{2.44}$$

subject to:

$$x(\delta(i)) \quad \text{is a positive even integer} \quad \forall i \in V, \tag{2.45}$$

$$x(\delta(S)) \geq 2 \quad \forall S \subset V, \tag{2.46}$$

$$x_e \geq 0 \quad \forall e \in E. \tag{2.47}$$

A vector $x \in R^E$ is a tour if (2.45) − (2.47) is satisfied and the convex hull is represented by GTSP(G). Cornuéjols et al.[18] proved that the polyhedron GTSP(G) is full dimensional when G is connected. An interesting observation is that the GTSP(G) is also an unbounded polyhedron that can be related to the TSP polyhedron in that, the $TSP(G) = GTSP(G) \cap \{x \in \Re^E : x(E) = |N|\}$, i.e., Q_T^n is a face of the GTSP(G). It is these observation that led Cornuéjols et al. to discover path inequalities which are also valid to Q_T^n and they generalize the comb inequality. A number of inequalities in the HTC for Q_T^n are valid in the GTSP(G), although they might not necessarily be facet inducing.

2.3.5.6 Path, Wheelbarrow and Bicycle inequalities

Path inequalities are also valid facet inducing to the STSP(n), but for the GTSP(G) Cornuéjols et al. defines path inequalities as follows:

A *K path configuration* is a partition of V into $\{A, Z, V_j^i \quad \text{for} \quad i = i, \cdots, k \quad \text{and} \quad j = 1, \cdots, n_i\}$ where $k \geq 3$ is an odd integer and $n_i \geq 2$ for $i = 1, \cdots, k$. The variable A and Z result from V_j^i when $j = 0$ and $j = n_i + 1$ respectively. The edges (V_j^i, V_{j+1}^i) are connected. This path inequality is given as follows:

$$\sum_{e \in E} f_e x_e \geq 1 + \sum_{i=1}^{k} \frac{n_i + 1}{n_i - 1}, \tag{2.48}$$

where

$$f_e = \begin{cases} 1 & e \in (A, Z), \\ \frac{|j-p|}{n_i - 1} & \text{for} \quad e \in (V_j^i, V_p^i), i = 1, \cdots, k \quad \text{and} \quad j \notin p, \\ & \text{such that} |j - p| \leq n_i. \\ \max\{\frac{p}{n_r - 1} - \frac{j-2}{n_i - 1}, \frac{j}{n_i - 1} - \frac{p-2}{n_r - 1}\} & \text{for} \quad e \in (V_j^i, V_p^r) \quad i \neq r \quad \textit{and} \quad j = 1, \cdots, n_i, \\ & p = 1, \cdots, n_r. \\ 0 & \text{Otherwise.} \end{cases} \tag{2.49}$$

When $n_i = n \geq 2$, the inequality (2.48) and (2.49) can be simplified as follows:

$$\sum_{e \in E} g_e x_e \geq kn + n + k - 1, \tag{2.50}$$

where:

$$f_e = \begin{cases} n-1 & e \in (A, Z)\,, \\ |j-p| & \text{for} \quad e \in (V_j^i, V_p^i), i = 1, \cdots, k \quad \text{and} \quad j \notin p, \\ & \text{such that} \quad |j-p| \leq n_i. \\ |j-p|+2 & \text{for} \quad e \in (V_j^i, V_p^r) \quad i \neq r \quad \text{and}, p = 1, \cdots, n_r. \\ 0 & \text{Otherwise.} \end{cases} \tag{2.51}$$

Such a configuration was referred to by Cornuéjols et al., as a *n-regular* path. When $k = 3$ and $n = 2$, the n-regular path can be shown to be 3-star inequality given by Fleischmann [31]. Star inequalities wont be covered in the scope of this research. Cornuéjols et al. also showed that this inequality is valid for the G TSP(G), that it defines facets of the GTSP(G), and more interestingly that the path inequality generalizes the comb inequalities such that $H \setminus \cup_{i=1}^{k} T_i \neq \emptyset$ and $\cup_{i=0}^{k} T_i \neq N$, with the 2-regular path inequality giving the same faces for the TSP polytope. Lastly, he showed that the two path combined with the wheelbarrow and bicycle inequalities generalizes the comb inequalities.

The bicycle and wheelbarrow inequalities are defined as follows:

$$\sum_{e \in E} f_e x_e \geq 1 + \sum_{i=1}^{k} \frac{n_i + 1}{n_i - 1}. \tag{2.52}$$

f in this instance is defined by:

$$f_e = \begin{cases} \frac{|j-p|}{n_i - 1} & \text{for} \quad e \in (V_j^i, V_p^i), i = 1, \cdots, k \quad \text{and} \quad j \notin p, \\ & \text{such that} |j-p| \leq n_i. \\ \max\{\frac{p}{n_r - 1} - \frac{j-2}{n_i - 1}, \frac{j}{n_i - 1} - \frac{p-2}{n_r - 1}\} & \text{for} \quad e \in (V_j^i, V_p^r) \quad i \neq r \quad \textit{and} \quad j = 1, \cdots, n_i, \\ & p = 1, \cdots, n_r. \\ 0 & \text{Otherwise.} \end{cases} \tag{2.53}$$

here, the wheelbarrow configuration is defined by an odd integer $k \geq 3$ and an integer $n_i \geq 2$ for $i = 1, \cdots, k$ with node set V partitioned into $A = V_0^i$, $i = 1, \cdots, k$ and V_j^i for $j = 1, \cdots, n_i$. The edge set (V_j^i, V_{j+1}^i) and $(V_{n_i}^i, V_{n_{i+1}}^{i+1})$ is nonempty for $i = 1, \cdots, k$ and $j = 1, \cdots, n_i - 1$.

The bicycle configuration is also defined by an odd $k \geq 3$ and an integer $n_i \geq 2$ for $i = 1, \cdots, k$ with node set V partitioned into V_j^i, for $i = 1, \cdots, k$ and $j = 1, \cdots, n_i$, The edge set (V_j^i, V_{j+1}^i), $(V_{n_i}^i, V_{n_{i+1}}^{i+1})$, and $(V_1^i, V_{n_{i+1}}^{i+1})$is nonempty for $i = 1, \cdots, k$ and $j = 1, \cdots, n_i - 1$. Figures 2.11 - 2.13 illustrate the K-path configuration, the bicycle inequality, and the wheelbarrow inequality, respectively.

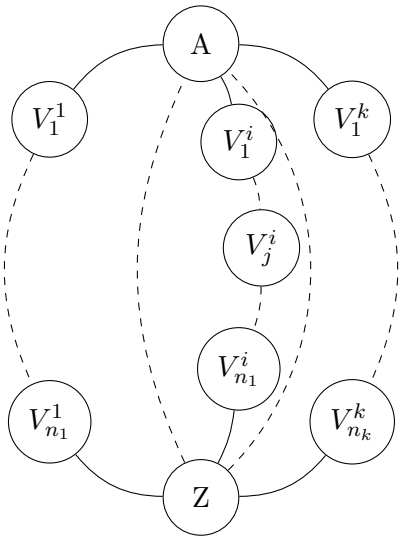

FIGURE 2.7: A K-path configuration

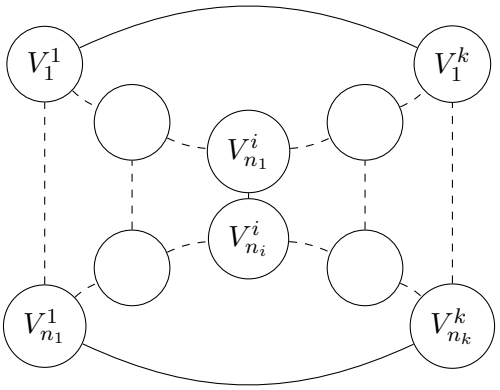

FIGURE 2.8: The bicycle inequality

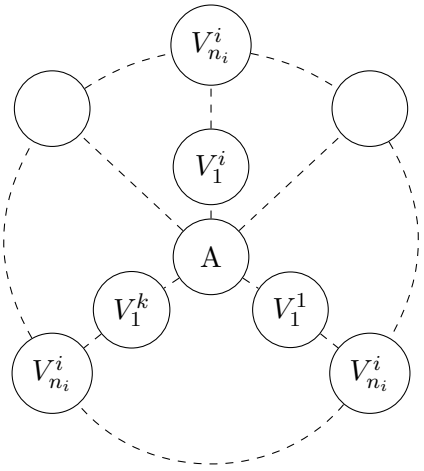

FIGURE 2.9: The wheelbarrow inequality

Apart from the subtour elimination constraint, the constraints presented in section 2.3.5 are not trivial. The validity of these constraints, in closed set form, is discussed by

Naddef and Pochet [61]. An inequality is said to be in closed set form if it can be written as follows:

$$\sum_{i=1}^{p} \alpha_i x(\delta(S_i)) \geq r(S), \tag{2.54}$$

where $S = \{S_1, \cdots, S_i, \cdots, S_p\}$ are distinct subsets of V, $\alpha_1, \alpha_2, \cdots, \alpha_i, \cdots, \alpha_p$ are positive integers and $r(S)$ is a function on S. Note that, if S is a proper subset, $\alpha = 1$ and $r(S) \geq 2$, then equation (2.54) becomes $x(\delta(S)) \geq 2$, which is the subtour elimination constraint defined equation (2.24)

Naddef et al.[61] also explains why a set S is divided into a set of handles, H, and teeth, T_i, with non-zero zero integers $\alpha_1, \cdots, \alpha_i, \cdots, \alpha_h$. The known closed set inequalities are in the following form:

$$\sum_{i=1}^{h} \alpha_i x(\delta(H_i)) + \sum_{j=1}^{t} \beta_i x(\delta(T_j)) \geq A + 2\sum_{j=1}^{t} \beta_j \tag{2.55}$$

and can be arranged as follows:

$$\sum_{i=1}^{h} \alpha_i x(\delta(H_i)) \geq A - \sum_{j=1}^{t} \beta_j [x(\delta(T_j)) - 2]. \tag{2.56}$$

From their procedure, the values for A is:

$$A = \min_{\Gamma}\{\sum_{i=1}^{h} \alpha_i |\Gamma \cap H_i| : |\Gamma \cap \delta(T_j)| = 2 \quad \forall j = 1, \cdots, t\} \tag{2.57}$$

The function Gamma (Γ) defines a hamiltonian, and $|\Gamma \cap \delta(T_j)| = 2$ means that the edges in the coboundry of a tooth for a Hamiltonian are exactly 2. A set S is said to be *tight* when a hamiltonian cycle $|\Gamma \cap \delta(S)| = 2$ or $x(\delta(S)) = 2$. With this explained, it is observed that equation (2.57) is derived from (2.56) given it is in tight formulation for a hamiltonian, i.e., $x(\delta(T_j)) - 2 = 0$ such that (2.56) reduces to $\sum_{i=1}^{h} \alpha_i x(\delta(H_i)) \geq A$. β_j, $j = 1, \cdots, t$ is defined by the following equation:

$$\beta_k = \max_{l>1} \frac{(b_k^1 - b_k^l)}{(2l-2)}, \tag{2.58}$$

where

$$b_k^l = \min_{\Gamma \in \mathscr{G}}[\sum_{i=1}^{h} \alpha_i x^{\Gamma}(\delta(H_i)) + \sum_{j\leq} \beta_j [x^{\Gamma}(\delta(T_j)) - 2]]. \tag{2.59}$$

This lifting procedure requires $\Gamma \in \mathscr{G} = \{\Gamma : x^{\Gamma}(\delta(T_j)) = 2 \quad \forall j \geq k, x^{\Gamma}(\delta(T_k)) = 2l\}$, where $b_k^l = \infty$ if $\mathscr{G} = \emptyset$, $b_k^1 = A \quad \forall k \in 1, \cdots, t$.

Essentially, A can be regarded as the minimum number of times a tour crosses the borders of the handles when all teeth are tight, while β_j can be seen as the best gain per unit of increase of the number of crossing of the border of T_j.

Following the explanation by Naddef et al., [61], we note that the comb inequality given in (2.38) can be rewritten as:

$$x(\delta(H)) \geq t + 1 - \sum_{j=1}^{t} [x(\delta(T_j)) - 2], \tag{2.60}$$

and if all teeth are tight, we understand that a hamiltonian cycle will cross the boundaries of H in a minimum of $t + 1$ edges. With this understanding, proofs about the validity of these inequalities are and their formulation are easier to understand.

2.3.6 The STSP Cutting Planes

The STSP cutting planes, is the separation process of finding violated inequalities. We know that these inequalities should tighten the polyhedral structure of the STSP(n). Thus, the separation problem to the STSP(n) are algorithms that would find any of the aforementioned facet defining inequalities for a STSP(n). From the literature, we note that there exists two types of separation techniques. These are exact and heuristic separation algorithms. Exact separation algorithms are guaranteed to locate all existing violated inequalities, while heuristic separation algorithms are not. We now discuss some exact and heuristic separation algorithms before investigating the CVRP cutting planes.

Let us first define a support graph as $G_z = (V_n, E, z)$, of which z is a weighted graph whose edge set E contains all edges of E_n corresponding to a positive component of z. The weight associated with $e \in E$ is z_e. The edges of G_z with weight $z_e = 1$ are called 1-edges.
For the *subtour elimination inequality*, Gomory and Hu [36] proposed an algorithm to solve the minimum weight cut in a non negative edge weighted graph. This algorithm had the complexity of $O(|v|^2|E|log(|v|^2/|E|))$. Padberg and Rinaldi [64], proposed a much faster exact algorithm, which has the same complexity, but as the worst case scenario. A number of these algorithms exist, but for much larger STSP instances this order is undesirable. Heuristics are thus preferred, some of which are algorithms by Hao [44] and Nagamochi [62].

In the *2-matching inequalities*, a new supporting graph G'_z is created, with double the number of edges in G. This is generated by splitting each edge into two and joining them

with a new node such that $G_z^{'}$ has $n + |E|$ nodes. With this new approach, Padberg and Rao [66] proposed an algorithm that could solve an STSP in polynomial time for the 2-matching inequalities. An improvement on this can be found on Padberg and Grötschel (1985), and Padberg and Rinaldi (1990).

Carr [13] developed an exact algorithm which in practice is able to identify a comb with three teeth. Padberg and Rinaldi (1990) developed a separation algorithm for the $2 - matching$, combs and clique-tree inequalities. This algorithm uses the idea of shrinkable sets to identify violated inequalities. A set S is called shrinkable if the graph obtained by contracting S is not the supporting graph of a point z. In comparison, not much has been done for cliques tree inequalities compared to comb inequalities. Different studies on the comparison of different cutting planes has been previously studied and one of the few studies was undertaken by Jünger et al. [48].

2.4 Cutting Planes Method Applied to CVRP

In the preceding subsections we discussed cuts for general IPs and for the TSP. In the following subsection we will present cutting planes for the CVRP.

2.4.1 The CVRP mathematical formulation

For the CVRP, the graph $G(V, E)$ is given with V having $n + 1$ nodes, i.e., node $(0, 1, 2, \cdots, n)$ such that node 0 or v_0 represents the *depot* while the remaining nodes represent customers. Every customer is alloted a positive demand d_i and the depot contains K number of vehicles, all with the same capacity Q. For every $F \subseteq G$, $G(F)$ is called the subgraph of $(V(F), F)$ induced by F, where $V(F)$ is the set of nodes incident to at least one edge of F. Therefore, a feasible *route* may be defined as a nonempty subset R of E such that the induced subgraph $G(R)$ is a simple cycle containing the depot node v_0 and such that the total customer demand does not exceed maximum capacity per vehicle. In vehicle routing we want to minimize such cycles $R_1, R_2, \cdots, R_k$ i.e., to minimize the total number of cars used and the sum costs of all the routes such that each customer is visited by exactly one vehicle, and all customers are visited. To measure the quality of the objective function, there is a cost associated with every feasible solution. Typically, this function is made up of, the total length of routes of the tours and the number of vehicles used. i.e., for any given solution, a route may be traversed or not, and the same is true for vehicles.
Different formulations with their advantages and disadvantages exist to model the CVRP. In this section we begin by introducing the bin packing problem (BPP), which is pivotal to our discussion of CVRPs. We then introduce the CVRP integer formulation which is the extension of the formulation presented in subsection (2.3.4).

The bin packing problem can be described using terms found in the knapsack problem. i.e., we have knapsack (bins) of capacity W and n items of different weights (w_i). The bin packing problem consists of packing the varying weights into the bins without exceeding the bin capacity, in a manner that minimizes the number of bins used. Amongst the mathematical programming formulations that exist for the bin packing problem, below we present Martello and Toth's [58] formulation:

Minimize

$$z = \sum_{k=1}^{K} y_k, \tag{2.61}$$

subject to:

$$\sum_{k=1}^{K} x_{ik} = 1, \quad i = 1, \cdots, n, \tag{2.62}$$

$$\sum_{i=1}^{n} w_i x_{ik} \leq W y_k, \quad k = 1, \cdots, K, \tag{2.63}$$

$$y_k \in \{0, 1\}, \quad k = 1, \cdots, K, \tag{2.64}$$

$$x_{ik} \in \{0, 1\}, \quad \imath = 1, \cdots, n, k = 1, \cdots, K. \tag{2.65}$$

where,

$$y_k = \begin{cases} 1 & \text{if bin } k \text{ is used,} \\ 0 & \text{otherwise;} \end{cases} \tag{2.66}$$

$$x_{ik} = \begin{cases} 1 & \text{if item } i \text{ is assigned to bin } k, \\ 0 & \text{otherwise.} \end{cases} \tag{2.67}$$

K in this instance is the maximum number of bins needed. Using this formulation Motello and Toth [58] showed that an obvious lower bound could be written as:

$$\lceil \sum_{i=1}^{n} \frac{w_i}{W} \rceil, \tag{2.68}$$

and, can be obtained in $O(n)$. The BPP has been shown to be strongly NP-Hard and for a thorough analysis of results using approximation algorithms on the BPP, we refer the reader to a book by Coffman Jr et al. [17].

We introduce this formulation since the BPP can be regarded as the varying demands

(d) of n customers needed to be packed in the vehicles at the depot in such a manner that minimizes the total use of all vehicles. Therefore one formulation of the CVRP can be given as follows:

Minimize

$$z = \sum_{e \in E} c_e x_e, \tag{2.69}$$

subject:

$$x(\delta(\{i\})) = 2 \quad \forall i \in V, \tag{2.70}$$

$$x(\delta(\{0\})) = 2K, \tag{2.71}$$

$$x(\delta(S)) \geq 2 \left\lceil \frac{d(S)}{C} \right\rceil \quad \forall \emptyset \neq S \subset V, \tag{2.72}$$

$$0 \leq x_e \leq 1 \quad \forall e \in E \backslash \delta(0), \tag{2.73}$$

$$0 \leq x_e \leq 2 \quad \forall e \in \delta(0), \tag{2.74}$$

$$x_e \quad \text{integer} \quad \forall e \in E. \tag{2.75}$$

Equation (2.70) represents the connectivity of the graph and states that each edge may only connect to two nodes. Equation (2.71) refers to the number of vehicles that leave and enter the depot. Equation (2.72) is known as the capacity inequality. The CVRP allows $x_e = 2$ only for the degenerate case shown by (2.74) and that is when a vehicle visits only one customer and returns to the deport. eg: $(0, j, 0)$. The formulation (2.69) to (2.75) is known as the two index flow formulation. Though it is an LP formulation, it has an exponential number of constraints and thus it is difficult to solve.

2.4.1 The CVRP Polytope

The CVRP polytope has not received as much attention as the STSP(n) polytope. Unlike the STSP(n) polytope, which depends on the size n only, the CVRP polytope is more complex in that it depends on n, d, Q and K which all depend on each other. Therefore, the CVRP(d, n, Q, K) polytope is the convex hull of the representative vectors of all the K-routes. A K-route is therefore a feasible solution that can further be partitioned into disjoint subsets such that each partition total customer demand is less than Q and each partition contains the depot node.

A solution can be associated with a vector $x^R \in \Re^E$. This is a real vector with elements in E, such that, the value components x_e^R associated with e is the number of times e

appears in the solution where $x_e^R \in \{0, 1, 2\}$, as shown in (2.69)-(2.75). The CVRP is made up of two NP-Hard problems, the BPP and the TSP. Cutting planes can therefore result from cuts designed for any of the two problems, as long as they do not violate the CVRP formulation.

2.4.2 Inequalities from Capacity Constraints

Inequality (2.72) is referred to as the *rounded capacity inequality*. With this rounding, it is possible for the resulting inequality to be infeasible. There is a hierarchy for capacity inequalities. This is determined by the fact that there can be many expression sharing the same right hand side (RHS) term of (2.72). When the ceiling function of the RHS term is dropped, (2.72) is referred to as the *fractional capacity inequality*. For a more general RHS term, the capacity constraint can be defined as solving for,

$$R(S) = \min_{P \in \mathscr{P}} \beta(P, S), \tag{2.76}$$

where R is a function that maps the elements of the power set formed by clients to the positive integer set, P is the set of partitions (S_i), for $S_i \in S$ the set of all clients for $i = 1, \cdots, K$ formed by a K-route, and $\mathscr{P}$ is the set of all possible partitions of a K-route. Here,

$$\beta(P, S) = \{i : S_i \cap S \neq \emptyset\}. \tag{2.77}$$

Clearly $\beta(P, S)$ outlines all the feasible paths of a K-route and as a result it is the number of vehicles needed to satisfy all the customers in S. Notice therefore, that the *capacity inequality* has to have an even number, i.e., $2R(S)$. When the minimum of (2.76) can be obtained, we refer to it as a *tight* feasible partition. In reality R$(\cdot)$ is hard to compute as an inequality for cutting planes [74]. Weaker versions are used instead. These weak inequalities include *fractional capacity constraints*, *framed capacity constraints* and *weak generalized capacity constraints*.

2.4.3 Inequalities from Routing Constraints

Naddef and Rinaldi [74] showed that every valid STSP inequality put in tight triangular (TT) form is valid for the CVRP polytope. An inequality $ax \geq a_0$ is in TT if for all triples of distinct nodes $i, j, k \in V$, we have $a_{jk} \leq a_{ij} + a_{ik}$, and for every $i \in V$ there exist $j(i) \in V$ and $k(i) \in V$ such that $a_{j(i)k(i)} = a_{ij(i)} + a_{ik}$.

Theorem 2.1 *All the valid inequalities for the STSP polytope, written in TT form, are valid for the CVRP polytope.*

Proof. Assume a valid inequality $ax \geq a_0$, for the STSP polytope in TT form, is not valid for the CVRP polytope. Consider one K-route, say R. Its incidence vector x^R satisfies $ax^R \leq a_0$. If we consider two edges of R, $(0, i)$ and $(0, j)$, incident to the depot, then removing these two edges and adding edges (i, j) yields a $(k-1)$-route $R^{'}$ with $ax^{R^{'}} \leq a_0$ (because the triangle inequality holds). By doing so, this might violate the capacity inequality. Now, repeating this process, we end up with a Hamiltonian cycle Γ such that $ax^{\Gamma} \leq a_0$, which contradicts the fact that $ax \geq a_0$ is a valid inequality for the STSP polytope, this completes the proof. □

In the CVRP polytope, the BPP inequalities deal with the capacity part of the problem, while inequalities from the STSP polytope deal with the routing part of the problem. This inheritance is not straight forward, however, it is not surprising that these inequalities are valid although they might not necessarily define facets.

The STSP inequalities can also be strengthened for the CVRP polytope by adding the BPP inequalities in their formulation, this is done by understanding the role that these inequalities play in the polytope [61]. This was discussed under Section (2.3.5.6).

Toth et al. [74], strengthens the comb inequality by reformulating the comb inequality in a manner in which it force edges out of the handle using the capacity constraint. This is understood from the intuition given by Nadeff et al.[61], that the function of the teeth is somehow to force edges out of the handle in t minimum number of ways, and out of a hamiltonian in $t+1$ minimum number of ways. In a tight formulation, the intersection of the coboundary of a hamiltonian and of the handle or tooth is at least 2. In the CVRP, this is at least $2R(S)$ where $R(S)$ is given in (2.76). $R(S)$ can, however, be relaxed to $r(S)$ which is a rounded capacity constraint, where the integer restriction has been relaxed. This strengthened comb formulation is as presented. Given $H, T_i \subset V$ for $i = 1, 2, \cdots, k$, let no depot be contained in the tooth and let:

$$r(T_i \setminus H) + r(T_i \cap H) \geq r(T_i), \tag{2.78}$$

then the following inequality is valid ([74]):

$$x(\delta(H)) + \geq t + 1 - \sum_{i=1}^{t} (x(\delta(T_i)) - 2r(T_i)). \tag{2.79}$$

The term $r(T_i)$ can be thought of as a small estimate of the number of vehicles needed, which is determined by a heuristic. Inequality (2.79) can be thought of as a proper generalization of the comb inequality. If the depot was to be in any of the teeth, say, without loss of generality, $v_0 \in T_1 \setminus H$, then the demand of other sets outside T_1 would determine the number of edges needed in the co-boundary of T_i.

2.4.3.1 Valid Inqualities Combining Bin Packing and the STSP

Another form of inequality strengthening, resulting from the path bin support, is defined by a handle H, by teeth $T_1, T_2, \cdots, T_t$, and by spots $S_1, S_2, \cdots, S_i$. For notational simplicity, we let T_{t+i} stand for S_i such that:

$$H, T_1, T_2, \cdots, T_{t+s} \subset V, \tag{2.80}$$

$$d(T_j) \leq C \quad \forall 1 \leq j \leq t+s, \tag{2.81}$$

$$S_j \subset H \quad \forall 1 \leq j \leq s, \tag{2.82}$$

$$T_j \cap H \neq \emptyset \quad \forall 1 \leq j \leq t, \tag{2.83}$$

$$T_j \setminus H \neq \emptyset \quad \forall 1 \leq j \leq t, \tag{2.84}$$

$$T_i \cap T_j = \emptyset \quad \forall 1 \leq i \leq j \leq t+s, \tag{2.85}$$

$$t + s \geq 1. \tag{2.86}$$

Here, the tooth or the spot is forced to be less than or equal to the capacity constraint, and t need not be an odd number. The above formulation of a tooth can represent the vehicle capacity. By adding an extra constraint that a bin can contain items corresponding to at most two teeth, Toth et al [74] provided another valid inequality. For any valid path bin support $(H, T_1, \cdots, T_t, S_1, \cdots, S_s)$ associated with path bin inequalities and valid for the CVRP polytopes:

$$x(\delta(H)) \geq 2r'(H|T_1, \cdots, T_{t+s}) - \sum_{j=1}^{t+s}(x(\delta(T_j)) - 2). \tag{2.87}$$

2.4.4 Summary

This brief introduction in section 2.4.3.1 was used to demonstrate that we can also use different inequalities to get new valid inequalities. In this chapter, we presented many important types of cutting planes. Currently, cutting planes are used in conjunction with the branch and bound to solve problems to optimality, and this has resulted in an increase in the number of solved problems.

Chapter 3

Metaheuristics: Genetic Algorithm

In this chapter, a description of the Genetic Algorithm (GA) metaheuristic, and an overview of some approaches on how to solve CVRPs using the GA will be discussed. Using Matlab, we will also build our own simple GA Matlab code for CVRPs and test it on CVRP benchmark instances. In the code we study the effectiveness of different recombination operators to determine the best performers for CVRPs. In particular, the partial mapped crossover (PMX), the order crossover (OX), the alternative edge crossover (AEX), and the heuristic greedy crossover (HGX) are compared. This is done so as to determine the best combination of genetic operators for solving CVRP's. The results are later used for comparison against the ant colony optimization algorithm.

3.1 The main features of the Genetic Algorithm (GA)

The genetic algorithm was invented by John Holland [46] in 1970. It involves solving optimization problems by implementing processes observed during natural selection. The GA follows the idea that fit, clever and strong individuals (genes) have a higher chance of survival over unfit, and weak individuals. The later generation is therefore assumed to be better than the previous one. This population is created either by following a generational or the steady state approach. In the generational approach, the entire sample population is updated once per iteration, whereas in the steady state approach few candidates of the population are updated per iteration, respectively. By doing so, a stronger, fitter population than the previous one is being created at each iteration (or generation). This is equivalent to seeking for the better objective value in the optimization space at each time step.

A list of biological terms are used within the GA. These terms and their meaning are listed below:

- *Population* - a set of candidate solutions.
- *Genotype* - the individuals in the population.
- *Genes* - certain characteristics of an individual.
- *Chromosome* - encode the solution to the problem and are made up of different units (genes).
- *Loci* - position or location in the chromosome.
- *Parents* - the current set of candidate solutions.
- *Child* - new individuals formed by two parents.
- *Recombination* - asexual breeding.
- *Crossover* - special tweaks of two parent chromosomes to form two new children in the poplulation.
- *Generation* - when one cycle of fitness assessment, breeding and population reassembly has happened - we have a new generation.

3.1.1 Biological Optimization

Holland [46] thought of the GA during his studies in genetics and evolution of natural organisms. Biologists refer to these two studies as the study of synthetic theory of evolution [45]. In the synthetic theory of evolution, new individual species are formed through genetic changes from one population to the next. Through these changes, new individuals differ from old. Darwinian evolution [21] merges with Mendeline genetics theory [75] and makes a unified theory of evolution.
A tremendous diversity of species exist in nature. These species depend on one another for survival while others fight against each other. Survival is very important for all species in order for them not to be extinct. For example, rabbits are prey to many species such as snakes, eagles, owls, foxes, raccoons and even wild dogs, not to mention humans. For survival, rabbits need to out-smart and out-run the predator, have the ability to adapt in new environments, and to reproduce. Reproduction, therefore, plays a big role for the survival of rabbits and for their existence in the next generation. During reproduction, the traits aforementioned are passed on to new baby rabbits through genes

carried by chromosomes in the deoxyribonucleic acid (DNA). The DNA is known as the genetic code of individuals. These genes carry different characters passed on from parents to children. When these genes are exchanged some are dominant and others are recessive with dominant genes manifested more in the new generations. This means that some poor qualities can also be carried to the next generation, yielding weaker individuals than the previous generation. In general, however, it is assumed that the greater majority of weak rabbits do not survive the predator and hence constitute a smaller portion of the later generation. Therefore, individuals in the new generation have selected good qualities from both parents, making them potentially stronger and fitter than their parents.
During reproduction genes are copied from parents to children, this process is called *mitosis*. In this phase, chromosomes are copied and passed from parents onto their offspring. *Meioses* is the process of cell division that occurs were half the chromosomes from the mother and half the chromosomes from the father are co-joined through a process called *crossing over*. It is through this process, genetic material from both parents is copied to form the new unique child. *Mutation* then occurs to prevent the child from being identical to the parent.
The popular GA proposed by Holland [46], is built on these principles. In the following subsections, we present the GA, as well as a basic implementation using a peudo-code.

3.1.2 The Genetic Algorithm

The GA follows the principles of genetics and natural selection in order to solve for discrete or continuous optimization problems. It was first made popular by Goldberg in 1989 and is now popular for its suitability to solve hard complex optimisation problems, and also for its easy implementation. The following steps describe the GA procedure:

1. An initial population of individuals is generated.
2. All the individuals are evaluated and ranked with a fitness function appropriate to the problem at hand.
3. A breeding population is formed by selecting top-ranked individuals, using natural selection.
4. These top ranked individuals undergo certain transformations through genetic operators to reproduce children. These transformation include recombination and mutation.

When these four steps are repeated, the population increasingly becomes fitter. New children can either replace the entire population or portions of the population. This

is done by either replacing weak individuals in the population by children or letting the children compete with their parents for a spot in the next generation. Algorithm 3 below, shows these simple steps for GA procedure at high level.

Algorithm 3 A Simple Genetic Algorithm (GA) Process

1: `Initialization of variables`
2: Pop $\leftarrow$ {}, `create an empty population`
3: **for** `PopSize times` **do**
4: `Pop` $\leftarrow$ `Pop` $\cup$ {`new random individual`}
5: **end for**
6: Best $\leftarrow$ {} , `create an empty set of the best individual`
7: **repeat**
8: **for** `each` $Pop_i \in$ `Pop` **do**
9: AssessFitness(Pop_i)
10: **if** Best = {} or Fitness(Pop_i) > Fitness(Best) **then**
11: Best $\leftarrow Pop_i$
12: **end if**
13: **end for**
14: Q $\leftarrow$ {}, `create an empty set of` n `fittest individuals in Pop`
15: **for** $\frac{Pop-n}{2}$ `times` **do**
16: Parent $Pop_a \leftarrow Pop_i$, `Select other options, an individual from (Pop)`
17: Parent $Pop_b \leftarrow Pop_j$, `Select other options, an individual from (Pop)`
18: Children C_a, $C_b \leftarrow$ `Crossover(Copy(`Pop_a`),Copy(`Pop_b`))`
19: Q $\leftarrow$ Q $\cup$ {`Mutate(`C_a`),Mutate(`C_b`)`}
20: **end for**
21: Pop $\leftarrow$ Q
22: **until** Best solution is found or we have run out of time.
23: **return** Best

Algorithm 3 is already explained within the steps, but it is worth mentioning that step 1 variables, are variables the user might need for their code. These variables can be the population size, time, mutation rate, the percentage of the new population, and crossover rate. These are variables needed depending on the style of implementation. In general many variants of the GA exist. In the following subsection we will present an in-depth discussion of the GA within the context of CVRPs.

3.2 Genetic Algorithm for CVRP's

In this section, important concepts regarding the GA and its implementation on the CVRP will be discussed. Current research shows that the GA can be applied to solve CVRPs [3]. Also, there is no specific type of GA developed for CVRPs as discussed in chapter 1, but the implementation of the GA can be manipulated to suit the user's requirements. Hence, researchers either develop new approaches to the GA algorithm or experiment on existing ones. These experiments are based on improving existing GAs by either improving their operators or proposing new combinations to the GA, to hybridize it. We will investigate some operators, and the quality of results they produce. We will use the results they produce for comparison, in Chapter 5.

3.2.1 Problem Description and Mathematical Model

In section 1, we introduced the two index vehicle flow formulation to solve CVRPs, This formulation is not preferred for complex versions of VRP's. Instead, the three index vehicle flow formulation is used, the details of which are now presented [74]. With the three index formulation, we can explicitly indicate the vehicle which traverses an arc in the solution, where as with the two index formulation if we had different sized vehicles we could not indicate this. It is for this reason that the three index formulation is used. The three index formulation has $O\left(n^2m\right)$ variables, x_{ijm}, and $O\left(nm\right)$ variables y_{im}. $x_{ijm}\,(i \neq j)$, is equal to 1 in the optimal solution if arc $(i,j) \in A$ is traversed by vehicle M. Otherwise $x_{ijm} = 0$. We also have other binary variables given by $y_{im}\,(i \in V; m = 1, \cdots, M)$ which are set to 1, if customer i is served by vehicle m in the optimal solution or 0 otherwise. The three index model for the asymmetric CVRP is given as follows

$$\texttt{Minimize} \quad \sum_{i \in V} \sum_{j \in V} c_{ij} \sum_{m=1}^{m} x_{ijm} \tag{3.1}$$

`Subject to`

$$\sum_{m=1}^{M} y_{im} = 1 \qquad \forall i \in V \setminus \{0\}, \tag{3.2}$$

$$\sum_{m=1}^{M} y_{0m} = M, \tag{3.3}$$

$$\sum_{j \in V} x_{ijm} = \sum_{j \in V} x_{jim} = y_{im} \qquad \forall i \in V, m = 1, \cdots, M, \tag{3.4}$$

$$\sum_{i \in V} d_i y_{im} \leq C \qquad \forall m = 1, \cdots, M, \tag{3.5}$$

$$\sum_{i \in S} \sum_{j \notin S} x_{ijm} \geq y_{hm} \qquad \forall S \subseteq V \setminus \{0\}, h \in S, m = 1, \cdots, M, \tag{3.6}$$

$$y_{im} \in \{0, 1\} \qquad \forall i \in V, m = 1, \cdots, M, \tag{3.7}$$

$$x_{ijm} \in \{0, 1\} \qquad \forall i, j \in V, m = 1, \cdots, M. \tag{3.8}$$

When arc (i, j) is not in the solution then $x_{ijm} = 0, \forall M$. Equation (3.2) imposes that a customer is visited exactly once. Equation (3.3) imposes that M vehicles leave the depot, and (3.4) makes sure that the same vehicle which enters a node leaves the node. The inequalities in (3.5), represent the capacity restriction per vehicle, and equation (3.6) ensures the connectivity per circuit formed. These types of constraints were explained in (2.23) to (2.24). The three index versions of those constraints are:

$$\sum_{i \in S} \sum_{j \in S} x_{ijm} \leq |S| - 1 \qquad \forall S \subseteq V \setminus \{0\}, |S| \geq 2, m = 1, \cdots, M, \tag{3.9}$$

and the MTZ subtour elimination constraints are:

$$u_{im} - u_{jm} + C x_{ijm} \leq C - d_j \quad \forall i, j \in V \setminus \{0\}, i \neq j, \tag{3.10}$$

$$d_i \leq u_{im} \leq C \qquad \forall i \in V \setminus \{0\}, m = 1, \cdots, M. \tag{3.11}$$

3.2.2 Chromosome Representation

Implementation is somewhat different from the above formulation. In general, different approaches to implementation exist. We represent chromosomes for the CVRP as a unique random permutation of the size of the problem, i.e., the initial population is seeded randomly. Variations to the random permutation may include using local heuristics to create the initial population or defining integer distributions depending on the customer's requirements or the layout of the cities. Using local heuristics may be

advantageous in that they may reduce the computational time for the whole GA process if they generate a good initial population close to the optimum. The disadvantage of local heuristics is that they might have a negative effect on the GA, causing it to be trapped in local optima. It is, in this case, when the effectiveness of recombination operators is not visible in the improvement of the population fitness [56].

TABLE 3.1: Representation of an individual

Pop_i	1	2	4	3	8	7	6	5	9	10

Table 3.1 shows an instance of a ten city chromosome representation, where each number represents the customer or city, and in the order in which they must be visited. By using such a representation, a giant TSP tour starting and ending at the depot using the GA is generated. This solution is not the desired CVRP solution, as the explicit routes per vehicle are unknown. To get the final desired solution, Prins et al. [70] introduced an $O\left(n^2\right)$ approach called the optimal tour splitting techniques. This technique uses the Bellmen Ford shortest path [70]. However, in this paper we opt for a simple heuristic of $O\left(n\right)$. This method may not necessarily yield the optimal split but gives good reasonable results. The algorithm works as follows: given $w = [30, 20, 10, 50, 15, 35, 20, 20, 5, 10]$ the demand per customer, $W = 50$, and the chromosome(Pop_i) given as above. Then to allocate vehicles to customers, we allocate in the order of the solution Pop_i, taking into account the capacity constrains. If R represent a circuit tour. Then, $R_1 = v_0v_1v_2v_0$, is the first circuit, v_4 is not included since $\sum_{i=1}^{3} Pop_i \geq W = 50$. Thus the final solution is shown by Figure 3.1, a and b, and the independent routes are as follows:
$R_1 = v_0v_1v_2v_0$, $R_2 = v_0v_4v_0$, $R_3 = v_0v_3v_8v_7v_0$, $R_4 = v_0v_6v_5v_0$, $R_5 = v_0v_9v_{10}v_0$.

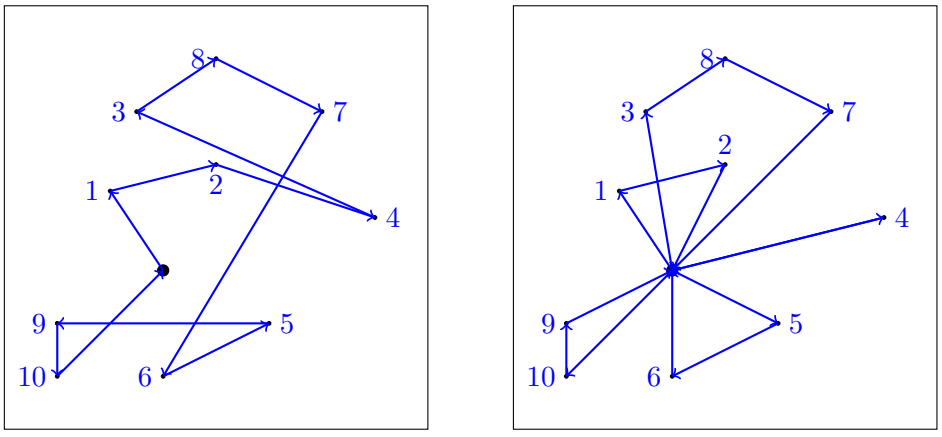

FIGURE 3.1: (*a*) Figure on the left represents the feasible solution as a TSP, capacity restrictions not considered. (*b*) The figure on the right represents the desired feasible solution for the CVRP, after capacity restriction are considered

3.2.3 Fitness Function

Using the above representation, R, to represent tours. R_i, for $i = 1, \cdots, n$, can be written as $v_{i0} \to v_{i1} \to \cdots \to v_{ki+1}$, where $v_{i0} = v_{ki+1} = v_0$. With this representation the fitness function can be described as:

$$f = \sum_{i=1}^{n} \sum_{j=1}^{k_1+1} \sqrt{(x_{i,j} - x_{i,j-1})^2 + (y_{i,j} - y_{i,j-1})^2}. \tag{3.12}$$

The fitness value measures the quality of the solution, and can thus be constructed depending on the users solution approach.

3.2.4 Selection Operator

Identifying the appropriate selection technique and crossover is critical when using the GA. It directly influences the speed or the overall performance of the algorithm and the quality of the final outcome. Selection techniques improve the population's fitness over successive generations or iterations in the GA.
If the selection process and crossover were not available in the GA, then the search would be random, and a good solution would not be guaranteed. John [38], describes selection as a three step process:

- Mapping the objective function to fitness.
- Creating a probability distribution based on the fitness.
- Drawing a sample from the distribution.

Within this process *Selective pressure* can be used. Selective pressure is a process of favouring certain individuals over others in the selection process. When the selective pressure is too high, GA lacks genetic diversity and thus can lead the algorithm to premature convergence. It is therefore important that the right pressure is selected and at the same time favour good individuals that can lead the GA to optimality. The reader may refer to Pandey et al. [67], who did a comparative review on approaches for handling premature convergence in the GA. At this stage in the process, fitness values of every chromosome in the initial population should have been calculated. These values can be represented as an $n \times 1$ vector, where n denotes the number of solutions. These fitness function values are used to determine which chromosomes to choose for recombination. Below we name the popular types of selection procedures and how they work:

- Roulette wheel selection,

- Truncation selection,
- Rank based fitness assignment,
- Stochastic universal sampling,
- Tournament selection.

3.2.4.1 Roulette wheel selection. In the roulette wheel procedure, like the name suggests, we have a roulette wheel where the circumference is made up of fitness values of individuals in the population. An individual is selected through spinning the roulette wheel. Mathematically, this is represented using a probability function given as follows:

$$P(i) = f(i)/\sum_{j=1}^{n} f_j. \tag{3.13}$$

From equation (3.13), the probability of an individual being selected is directly proportional to the fitness value of that individual. A parent chromosome is therefore selected by spinning of the roulette wheel. With this procedure, fitter individuals, therefore, have a higher chance of being selected because they occupy more space on the roulette wheel. If the fitness values of individuals in the population are equally distributed, then all members in the population have a fair chance of being selected.

3.2.4.2 Truncation selection. Truncation selection procedure refers to the process of eliminating certain members in the population in order that the leftover population is the one to undergo reproduction. There are different percentages of truncation selection that can be used. For instance, 50% truncation selection means that 50% of chromosomes are selected for recombination and the other 50% is eliminated and replaced by the children. This selection of individuals can be done randomly, or by eliminating the weak members in the population, or by any defined process set by the owner.

3.2.4.3 Rank based fitness assignment. The rank based fitness assignment procedure ranks individuals in the population before the selection is done. This method was introduced by Baker [4], to eliminate the scaling problems of highly fit individuals. Highly fit individuals constituted a bigger portion of the circumference of the roulette wheel resulting in a lack of diversity in this procedure. There are different types of rankings, i.e., there is linear ranking, non-linear ranking, and exponential ranking. Ranking ensures that there is a uniform distribution in the population and provides means to handle the selective pressure. Firstly, the population is rearranged according to their objectives values. The fitness value per individuals in the population

is then assigned based on the position of the individual and not on the actual objective value. The disadvantage here is that all chromosomes have an equal chance to be selected which can impact on the time it takes to find the optimal value.

3.2.4.4 Tournament Selection. In tournament selection, individuals in the population are made to compete with one another and the best individual per tournament is selected for reproduction. The number of participants for the tournament is determined by the user. This is known as the tournament size. Some variants also take the second best individual for reproduction. This is probably one of the most favoured methods because it preserves diversity by giving individuals the opportunity to compete against each other. In this dissertation, we use a binary tournament procedure, by selecting two individuals randomly and letting them compete one on one. The winning parent is selected to participate in reproduction. The tournament takes place until we have the desired number of parents for reproduction. **Algorithm** 4 below provides an illustration on how tournament selection can be implemented.

Algorithm 4 Tournament Selection

```
1: Initialization of variables
2: Pop ← population
3: Best ← random individual picked from Pop with replacement
4: for i from 2 to t do
5:    Next ← random individual picked from Pop with replacement
6:    if Fitness(Next) > Fitness(Best) then
7:       Best ← Next
8:    end if
9: end for
10: return Best
```

3.2.5 Crossover Operator

Crossover is the distinguishing feature in GA. It affects both the speed of the code and quality of the solution. Crossover involves mixing and matching parts of two selected individuals called parents, to breed and form two different children. Depending on the solution representation, there are many ways to perform crossover. The classical crossovers for binary GA using vector notation are the, Single-Point, Two-Point and Uniform Crossover [45]. Some of the well known crossovers for ordered list representation that are used in the TSP are:

- Partially mapped crossover (PMX),

- Order crossover (OX),
- Alternative edge crossover (AEX),
- Heuristic greedy crossover (HGreedyX),
- Cyclic crossover (CX),
- Sequential construction crossover (SCX),
- Edge recombination crossover (ERX),

The list given above details a few of many crossover techniques which exist. In this dissertation, we only test the first four of these operators. Namely, PMX, OX, AEX, and HGreedyX.

3.2.5.1 Partially mapped crossover. The partially mapped crossover was originally designed for TSP by Goldberg et al, [34]. Two parents are selected from the list of individuals in the population to go through reproduction. From these two individuals, two crossover points within the length of the chromosome are chosen arbitrarily. The genes between cut points of parent one and parent two are then copied directly to the second and first child, respectively. The remaining genes from the parents are copied into empty spaces in the same two children following a mapping relation such that two genes are not repeated in the same chromosome. We illustrate this mapping rule with an example.
Consider $P1$ and $P2$ the two parent individuals to undergo crossover, and let $C1$, and $C2$ be the resulting children. We denote the two cut points by "|".

$P1$	1	9	7	8	4	10	6	5	2	3

$P2$	4	3	9	7	5	8	2	1	10	6

Genes from the crossover zone in $P1$ are directly copied to $C2$ and $P2$ crossover zone genes are copied to $C1$, " $*$" denotes the positions not yet filled.

$C1$	*	*	*	7	5	8	2	*	*	*

$C2$	*	*	*	8	4	10	6	*	*	*

The mapping relation formed by the two parents crossover zones is therefore given as follows: $7 \leftrightarrow 8 \leftrightarrow 10$, $4 \leftrightarrow 5$, $6 \leftrightarrow 2$. Genes outside the crossover zone from P1 are now copied directly to C1, and P2 to C2, respectively, taking into consideration the mapping relation. The new PMX children formed are as follows:

$C1$	1	9	10	7	5	8	2	4	6	3

$C2$	5	3	9	8	4	10	6	1	7	2

3.2.5.2 Order Crossover. Order crossover was first proposed as a TSP crossover technique. This method is similar to PMX in the sense that it generates two cut points. In OX, genes in the crossover zone from $P1$ are copied directly to $C1$, and $P2$ to $C2$, respectively. The remaining spaces in the child are filled in by elements from P1 and P2. Genes in $P2$ are copied to $C1$ and $P1$ to $C2$, respectively, in an ordered fashion. The first element is copied from the first gene directly after the second cut point from parent 1 to child 2 at same position. Moving from left to right, genes are copied to the child until all the elements are present in the new child.
For instance, Let the two parents and the two cut points be the same as in the PMX. $C1$ and $C2$ using OX are as follows:

$C1$	7	5	2	8	4	10	6	1	3	9

$C2$	4	10	6	7	5	8	2	3	1	9

3.2.5.3 Alternative edge crossover. Alternative edge crossover is an edge preserving operator. An edge is selected at random from either parent to be the starting edge (a, b) in the child. From there, edge (b, c) from the *alternative* parent is selected. The process continues in this way by alternatively selecting edges from alternative parents. When a new selected edge forms a cycle in the child, then a new edge which is randomly selected is used to break this cycle. The process continues until all genes have been filled in the new child. For instance, let $P1$ and $P2$ be as follows:

$P1$	6	3	7	1	5	10	9	8	2	4

$P2$	10	5	8	3	9	2	7	6	1	4

If arc $(6, 3)$ is selected first from $P1$, then, the next arc will be $(3, 9)$ from $P2$ followed by arc $(9, 8)$ from $P1$.

$C1$	6	3	9	8	*	*	*	*	*	*

The arc going out from 8 should be chosen from $P2$, but this choice is infeasible since this will form a cycle too early. To avoid this, any random edge in $P2$ which is not yet in $C1$ is selected, and the process is continued. The complete child $C1$ and $C2$ is as follows:

$C1$	6	3	9	8	1	5	2	4	10	7

$C2$	10	5	1	4	6	2	3	9	8	7

3.2.5.4 Heuristic greedy crossover. Most crossover operators don't exploit the distance between cities. It is for this reason we have an interest in the heuristic crossover. The heuristic greedy crossover does a bit of exploiting of the domain space. In fact, it is common for the genetic approach to avoid any heuristic information about the domain of a problem. The algorithm has some resemblance with AEX. Here, the child is formed by choosing, from each vertex, the cheapest of the two parent arcs. Grefenstette [39] lists the following steps to the heuristic greedy crossover:

1. Pick a random starting city from one of the two parents.
2. Compare the edge leaving the current city in both the parents, and select the shorter edge.
3. If the shorter parental edge introduces a cycle in the partial tour, then extend the tour with a random edge that does not introduce a cycle.
4. Repeat step 2 and 3 until all the cities are included in the tour.

Considering $P1$ and $P2$ to be the same as in the previous AEX example. Assume that the costs associated with the arcs are as follows:

$P1 \cdots c_{63} = 15, c_{37} = 5, c_{71} = 25, c_{15} = 20, c_{510} = 10, c_{109} = 5, c_{98} = 30, c_{82} = 20, c_{24} = 2, c_{46} = 20.$

$P2 \cdots c_{105} = 10, c_{58} = 20, c_{83} = 4, c_{39} = 30, c_{92} = 30, c_{27} = 20, c_{76} = 2, c_{61} = 15, c_{14} = 5, c_{410} = 20.$

To create a child following the above procedure, take vertex 6 as our starting city. The arcs leaving from node 6 in $P1$ and $P2$ are considered, i.e. $6 \rightarrow 3$ with cost $c_{63} = 15$ and $6 \rightarrow 1$ with cost $c_{61} = 15$. In this particular case, both arcs have the same cost so we chose one randomly. The child is initialized as,

$C1$	6	1	*	*	*	*	*	*	*	*

Next, $1 \rightarrow 5$ with cost $c_{15} = 20$ in $P1$ and $1 \rightarrow 4$ with cost $c_{14} = 5$ in $P2$. $1 \rightarrow 4$ is cheaper. Therefore, the successive vertex is 4.

$C1$	6	1	4	*	*	*	*	*	*	*

Whenever an infeasible arc is found, a random arc with the least cost is chosen and the cycle continues until $C1$ is completed.

3.2.6 Mutation

Mutation, even though not implemented in this work, is usually done after crossover on new individuals in the GA. It improves performance, by helping the crossovers to escape from local minima. This is done by altering the new created chromosome, after crossover, by a certain percentage, from the list of genes in that chromosome. Mutation depends on the encoding used, and because we have used permutations to represent the CVRP the following mutations apply:

- Mutation by inversion,
- Mutation by reinsertion,
- Mutation by swapping,
- Mutation by scramble.

We refer the reader to Soni and Kumar [72] for the explanation of the above mutations. Though mutation helps escape the local optimum, by helping the GA from converging to popular solution, it can prohibit the GA from converging too fast. After mutation and validation of the new offspring, the associated cost of the muted offspring and the original offspring is calculated and compared. The best chromosome is then either injected directly in the population or has to compete with its parents.

Chapter 4

Metaheuristics: Ant Colony Optimization

In this Chapter, we will discuss the ant colony optimization (ACO) metaheuristic. We explore this metaheuristic by firstly discussing its origin (real ants), and then transition from there onto artificial ants. We thought it worthwhile to discuss it in some depth. We then perform numerical testing on benchmark CVRPs, using our own implementation of the ant colony system (ACS). Lastly, we compare our results with that of Bullnheimer [12].

4.1 The main features of the Ant Colony Algorithm

There are no real ants in ACO. ACO was inspired through the study of ants interacting in an ant colony. Although Dorigo et al., [27] was the first to introduce this algorithm, research had already started about the interaction of I. humilis, linepithema humile and lasius niger ants. Ants stigmergy can be defined as the indirect communication of ants modified now and again through interaction with one another and the environment. Dorigo et al., [27] discovered that communication between the ants and the environment is based on a chemical called *pheromone* produced by ants. Dorigo et al., [27] also discovered the following behavior in ants: the division of labour; brood sorting; foraging; and cooperative transport. Ants deposit pheromone on the ground or path they take when searching for food so that they are able to trace back the path they took and also for foragers to follow this pheromone trail. This is why we usually see ants walking in a line as if they are marching.

ACO, which is an evolutionary algorithm, is part of a larger field of Swarm Intelligence algorithms. this includes the studying of bees, birds, fish and other social insects, to try and simulate their behaviour processes in order to solve hard optimization problems. Exact algorithms have computational complexity problems, as mentioned in chapter 2, when solving for large CVRPs. For this reason, we investigate the performance of ACO, for solving large CVRPs

4.1.1 Real Ants - The double bridge experiment

Though Dorigo was the first to introduce the ant system (AS) which is the first ACO algorithm, Deneubourg et al. [22] and Goss et al. [37] are amongst the earliest researchers to model the behaivour of ants as observed under controlled conditions using the double bridge experiment. The bridge they used consisted of two branches of varying lengths connecting the nest and the food source for Argentine ants. Figure 4.1 illustrates this bridge. The varying branch ratio is given by, $r = b_l/b_s$ where b_l is the length of the longer branch and b_s is the length of the shorter branch. Argentine ants were allowed to cross the two branches in search of food and were not restricted to one branch. The ants, however, had to make a choice about which bridge to use when at the crossroads. In their experiment, Deneubourg et al. [22, 37], tested for varying lengths of the branch, including when $b_l = b_s$, i.e., when $r = 1$ as shown in Figure 4.1. They also allowed the ants to wonder about on the bridge for a certain time t. These ants moved from the nest to the food using the bridge. The following conclusions about these observations were made.

For the duration of the entire experiment, most ants used the short branch. As time passed by, more and more ants were attracted to this path until almost all of the ants were using this single path. The reason for this is that the pheromone secreted by the ants became more dense and thus attracted more ants to the path. The less the ants used a single path, the less pheromone was secreted on it. The remaining pheromone concentration, therefore, did not increase nor remain the same, but it evaporated at a rate due to exposure to the sun. Becker [50], contributed to this finding by observing that some ants, when searching for food, secrete more pheromone when a particular food source is found. The ant becomes more excited, and secretes more pheromones in order to attract more foragers. Another interesting observation which was found through the double bridge experiment was that some ants still used the long path even when most ants preferred the shorter branch. This is defined as exploration, and is useful in the event that ants have converged to a local optimum path. Further-more, it was observed that ants preferred the long branch over the short one, if initially they

were exposed to the long branch for some time before the shorter branch had been added to the system.

FIGURE 4.1: Experimental setup for the double bridge experiment. (a) The left image has equal branch lengths. (b) The right image has different branch lengths. ([28], 2004, p.3)

We now consider artificial ants, and introduce a simple stochastic model to aid in the understanding of the ACO metaheuristic. This is presented to facilitate understanding of the ACO metaheuristic when we describe artificial ants. The following simple mathematical model explains the dynamics of the ant colony as observed under the double bridge experiment [22, 28, 37].

4.1.1.1 The simple stochastic model.

The simple mathematical model of the real ants experiment was developed by Deneubourg et al. [22], Goss et al. [37], and Dorigo [28]. This model consists of, ψ, the number of ants that cross the bridge (double bridge) per second in both directions at a constant speed v cm/s. The time t_s ($time = distance/speed$) it takes for an ant to traverse the short branch is given by $t_s = l_s/v$ seconds, while for the long branch $t_l = r \cdot t_s$ seconds, where $r = l_l/l_s$, and l_l and l_s are the respective lengths of the long branch and the short branch in the double bridge experiment. The total amount of pheromone on the branch at time t is the function ψ_{ia}, where $a \in \{l, s\}$. Now, the probability of an ant at decision point $i \in \{1, 2\}$, using the short path $s \in a$, is denoted by P_{is} and given as follows:

$$P_{is}(t) = \frac{(t_s + \varphi_{is}(t))^\alpha}{(t_s + \varphi_{is}(t))^\alpha + (t_s + \varphi_{il}(t))^\alpha}, \tag{4.1}$$

where α represents real values. The parameter α has been studied for different values to see how it influences the model. In their experiments, Deneubourg et al. [22, 28, 37], found that $\alpha = 2$, yielded the best results. The probability that an ant, at decision point $i \in \{1, 2\}$ uses the long path, i.e., $P_{il}(t)$, can be computed such that $P_{is}(t) + P_{il}(t) = 1$. The differential equation which describes the evolution of the stochastic system is given by:

$$\frac{d_{\varphi_{is}}}{d_t} = \psi p_{js}(t - t_s) + \psi p_{is}(t), \qquad (i = 1, j = 2; i = 2, j = 1), \tag{4.2}$$

$$\frac{d_{\varphi_{il}}}{d_t} = \psi p_{jl}(t - r \cdot t_s) + \psi p_{il}(t), \qquad (i = 1, j = 2; i = 2, j = 1). \tag{4.3}$$

Here, equation (4.2) can be interpreted as the instantaneous variation at time t of pheromone on s at i, equal to the flow of ants given by ψ multiplied by the probability of the ant at decision point 1 and 2. Equation 4.3 follows the similar interpretation but with ants using the long branch. The reader is referred to Dorigo et al. [28] for more details about this formulation and the outcome of the results. This discussion was just presented to help the reader understand the formulations of the artificial ants, as well as the idea behind the new assumption on the ACO algorithm that will be discussed next. We now present an in depth discussion on artificial ants.

4.1.2 Artificial Ants

In the preceding subsection we presented the double bridge experiment. In this section, we look at discrete data sets, because the CVRP, which we want to solve, is described as a graphic problem. In order to incorporate the CVRP as a graphical problem, the bridge experiment has to be translated to a graphic modeling problem. Additional assumptions are needed from the continuous double bridge experiment. This is done to ensure that the ants find the shortest path between two nodes, and in general, to find the shortest path between the source node and the destination nodes for any n sized connected graph. A direct extension of the stochastic model for artificial ants has drawbacks. These drawbacks are that ants can form loops while building the solution because of pheromone trail deposits. Such loops are hard to escape, and may trap many ants. This phenomena is also seen in reality and is called an ant mill. Even if the ants escape such a loop, the overall pheromone trail distribution becomes such that the short paths are no longer favoured. Because of these drawbacks, we need to introduce an artificial ant with more assumptions and extensions.
The first assumption is that time is discrete, i.e., at each time step an individual ant can move to the respective neighbour node at a constant speed of one unit of length per time unit. Secondly, an ant is also assumed to deposit one unit of pheromone on any arc it has traversed. Thus the model can be viewed in two ways:

1. A connected graph of equal edge length, where longer paths have more edges.
2. A connected graph of unequal edge lengths.

The first artificial ant model was developed by Marco Dorigo [25] 1991, in fulfillment of his Phd Thesis, to solve COP problems. This model was named the *Ant System* (AS). This model is explained next.

4.1.2.1 Simple artificial ant model. Today the AS together with its extensions is what we refer to as ACO algorithms. The AS is a construction metahueristic, i.e. ants have a starting point and they build a solution by moving around on the connected graph $G = (V, A)$ with edge weights. The problem constraints are built into the ant constructive heuristics. Ants are also allowed to construct infeasible solutions much like in the GA, if the user permits this. For an artificial ant, all nodes $v_i \in V$ and edges $a_{ij} \in A$ are assigned a *pheromone trail* τ (τ_i if associated with the node, τ_{ij} if associated with connections), and a *heuristic value* η(η_i if associated with the node, η_{ij} if associated with connections). The values τ and η are the factors which influence an ants behavior when making a probabilistic decision on how to move along the graph. In the artificial ant model, ants deposit pheromone on the path taken, and evaporation is applied on all edges in G. The updating of pheromone and heuristic information on G is done once an ant has completed a partial solution. Ants will move on the graph $G(V, A)$ interacting in this way to improve the solutions which they build at each iteration. Note that if no ants traverse a particular path for many iterations, then the pheromone on that path decreases drastically, i.e., pheromone updates only take place on the arcs which the ants have traversed. By doing so, the most popular paths will receive more pheromone and will attract more ants. In this manner, the ant system will converge. It is interesting to note that even when the system has converged, some ants still prefer to wonder about the graph.
All arcs are given a higher initial pheromone at the beginning of the algorithm to allow more exploration of arcs on the system and to avoid premature convergences. Moreover, the following properties of an artificial ant k in the colony, at any step, hold [28]:

- An ant exploits the construction graph $G = (V, A)$ in the search for an optimal solution $s^* \in S$, where S is the subset of candidate solutions of X and s^* is the minimum cost feasible solution.
- An ant has a memory M^k that it can use to store information about the path it has followed so far. This memory is useful to build feasible solutions, retrace the ant path, evaluate the solution, and calculate the heuristic value η.
- An ant has a start state x_s^k and one or many termination conditions e^k[1].
- When the ant is in state $x = \langle x_{r-1}, i\rangle$, it will move to node j in its neighborhood $N^k(x_r)$ given the termination condition is not met, and $x = \langle x_{r-1}, i\rangle \in X$.

[1]The general termination criterion is set by the user, and is commonly defined in terms of the maximum CPU time, a percentage deviation of the obtained optimum from the current optimum or a maximum number of iterations.

- An ant selects a move using probability. This probability decision rule is a function of the locally available pheromone trails and heuristic value, the ants private memory stores its current state, and the problem constraints.
- When adding a component v_j to the current state, it can update the pheromone trail τ associated with the corresponding connection.
- An ant can retrace the same path it took to reach the destination, when going to the source node, and update the pheromone trails of the used components.

Implementation of the ACO, with the aforementioned assumptions, can be described in terms of three procedures:

1. Constructing solutions.
2. Updating pheromone.
3. Applying a local search (optional).

The high level pseudo code for these steps is presented in **Algorithm** 5 below.

Algorithm 5 Simple ACO pseudo-code

```
1: Procedure ACO Metaheuristic
2: While termination not met
3: GenerateSolution
4: UpdatePheromone
5: ApplyLocalSearch
6: end
```

Step 3 and 4, respectively, are the most important steps in the ACO algorithm. These two steps are enough to simulate the basic ant system. The local search step is an optional step which is used to help the ACO improve the quality of the solution in the search for a global optimum.

4.2 ACO Mathematical Formulation and Algorithm

A large amount of work has been done on improving the AS metaheuristic since 1990 when it was first implemented. In general, the performance of ACO algorithms differ depending on the type of problem which is being solved. Usually these algorithms are tested on available benchmark problems. Therefore, their performance is measured on the average performance based benchmark problems. For the sake of completeness, a

brief overview of the ACO metaheuristic designed for TSPs is discussed. We also discuss the implementation of ACO for CVRPs. Lastly, we will present numerical results based on our implementation

4.2.1 Travelling Salesman Problem using ACO

Due to the relationship between the TSP and the CVRP we present the mathematical formulation of the AS using the TSP. The TSP has played an important role in the research of ACOs. In fact, AS was first tested on TSP examples. The TSPs have become the standard testing problem for many new algorithms developed for COP. This is because TSPs are easily understood and can be easily implemented. The TSP also has many benchmark test problems with solutions for different sized problems. When the AS was introduced, its performance was inferior to the existing TSP solvers. It was therefore not popular. Since 1991 many researchers interested in metaheuristics for COP hard problems have studied and improved this algorithm including Dorigo [27]. Some improvements to the original AS include *Elitist AS* [26], *Ant Q* [23] *Rank-based AS* [11], *Ant Colony System* [24], *Max-Min AS* [73], *ANTS* [57] and *Hyper cube AS* [8].
Differences between AS and these extensions are based on the way pheromone updates are performed, as well as some additional details in the management of pheromone trails. A small subset of these algorithms contain other unique features in their implementation.We present some of these differences in the next subsection.

4.2.2 Mathematical formulation

In chapter 2, the description of the TSP was given. The TSP is an NP-hard combinatorial optimization problem and to solve it one is required to find the shortest tour (a Hamiltonian cycle) across all cities in the problem. In this subsection, we will restrict our attention to the ATSP. To solve the TSP using AS, we first need to assign pheromone values to all arcs on graph G. Initial pheromone trails can be estimated in many different ways. One of the ways to do so is by setting $\tau_{ij} = \tau_0 = m/C^{nn}$, where m is the number of ants in graph G, and C^{nn} is the length of a tour in G generated by the nearest-neighbour heuristic. Recall that τ_{ij} describes the desire, for an ant, to visit city j when in city i. Furthermore, the heuristic information η_{ij} is usually given as an inverse of the distance between city i and the desired city j, i.e., $\eta_{ij} = 1/d_{ij}$, where d_{ij}

is the distance between city i and city j and $d_{ij} \neq 0$. If $d_{ij} = 0$, η_{ij} is usually assigned a very small value.

4.2.2.1 Tour construction. In AS, the number of artificial ants in the system, m, are responsible for the construction of the solution. Ants are assigned an initial city or can be scattered randomly across all cities. For an ant k at city i to move to the next city j, the ant k uses a rule called the random proportional rule:

$$p_{ij}^k = \frac{[\tau_{ij}]^\alpha [\eta_{ij}]^\beta}{\sum_{l \in N_i^k} [\tau_{il}]^\alpha [\eta_{il}]^\beta}, \tag{4.4}$$

where $j \in N_i^k$. N_i^k is the feasible neighbourhood of ant k from city i, where α and β are two parameters which determine the relative influence of the pheromone trails and heuristic information, respectively. The probability of choosing a city outside N_i^k is 0. This implies that the probability of choosing $arc(i, j)$ is influenced by the pheromone trail τ_{ij}, and the heuristic information η_{ij}.
If $\alpha = 0$, then the ants rely only on the heuristic information and the closest cities are likely to be chosen. However, when the ants use pheromone trails without β, i.e., when $\beta = 0$, then the ants are more likely to construct bad solutions. Dorigo and Stutzle [28] made a significant contribution in this field by analyzing different values for α and β and how they perform in various ACO algorithms (without inclusion of a local search in the ACO). They also checked for parameters which resulted in good performance. Here, every ant k in position i stores all cities it has visited, in chronological order, to memory M^k. This memory allows the ant to compute the length it generated. We take a detailed look at this process.

4.2.2.2 Updating pheromone trail. Once all the ants have constructed their tours, pheromone evaporation is implemented on all the arcs in the graph. This is to avoid unlimited accumulation of pheromone laid by ants at every iteration which might render the ants trapped in suboptimal paths. It also helps the ants forget any prior bad decisions made. The formula for pheromone evaporation is given below:

$$\tau_{ij} \leftarrow (1 - \rho)\tau_{ij}, \quad (i, j) \in L, \tag{4.5}$$

where $0 \leq \rho \leq 1$ is the evaporation rate. Pheromone deposits are only made on arcs which the ants have traversed. This is done at each iteration once all ants have built their tours, but is not the case in later advances. Dorigo [28], proposed that pheromone deposit be done before evaporation, We will, however, not consider this in the discussion

which follows nor in our implementation of ACO. Pheromone updates are given by:

$$\tau_{ij} \leftarrow \tau_{ij} + \sum_{k=1}^{m} \Delta\tau_{ij}^{k} \quad \forall (i,j) \in L, \tag{4.6}$$

where $\Delta\tau_{ij}^{k}$ is the amount of pheromone ant k deposits on arcs which it has traversed. Here $\Delta\tau_{ij}^{k}$ is given as:

$$\Delta\tau_{ij}^{k} = \begin{cases} \frac{1}{C^k} & \text{if arc}(i,j) \in T^k, \\ 0 & \text{otherwise,} \end{cases} \tag{4.7}$$

where C^k is the length of the tour T^k built by the k^{th} ant. This means that arcs which are shorter in length are more likely to receive more pheromone, which increases the probability of more ants choosing to traverse these arcs.

So far we have discussed the mathematical formulation of AS. In the next subsection we will provide a more detailed implementation of this algorithm.

4.2.3 The ACO Metaheuristic

Two types of implementations exist for AS, and ACO algorithms in general. These are sequential implementation and parallel implementation. The difference between these two implementations is that in the sequential implementation, an individual ant constructs a full solution before the next ant builds its solution. Conversely, in the parallel implementation all ants construct solutions simultaneously. In this paper, we present the parallel implementation.

4.2.3.1 Matlab Implementations

When implementing the simple AS in Matlab, the following have to be initialized:

- Parameters,
- Distance matrix,
- Pheromone and heuristic information matrix,
- Nearest neighbour list (for large problems).

An $(n \times n)$ distance matrix contains the intercity distances. One can also use two individual arrays containing the x and y coordinates respectively (of the cities). For computational efficiency, a matrix to store the product of τ and η can be created. In

this paper, we will call it the 'ChoiceMatrix'. Additionally, pheromone and heuristic information matrices should be created to store all the cities and arc information in advance. When the ant is created it should be noted that an ant is a computational agent. It constructs a solution by depositing pheromone to the arcs it has traversed. At any step in the algorithm, when the code calls for an ant, it should be able to display the following:

- The partial solution it has constructed so far,
- The ant should able to compute and store the objective function of its solution,
- The ant should determine the feasible neighbourhood.

More information can be added to this list depending on the implementation. These include storing the accumulated demand per ant when solving the CVRP. An ant can, therefore, comprise of an $(n+1)$-dimensional array to store the tour, a variable to store its cost, and an n-dimensional array to store the visited cities.
Algorithm 6. presents a Matlab pseudo code of the AS applied to ATSP.

Algorithm 6 Generating solution step

```
1: Initialization (a)
2: for  k = 1 to m  do
3:    ant(k).visited(all rows) = false
4: end for
5: step ← 1
6: for  k = 1 to m do
7:    ant(k).tour(step) = random{1,...,n}
8:    rval = ant(k).tour(step)
9:    ant(k).visited(rval) = true
10: end for
11: while step < n do
12:    step = step +1
13:    for  k = 1 to m do
14:       Ant = ASDecisionRule(Ant,ChoiceMatrix,k,step,n)
15:    end for
16: end while
17: for k = 1 to m do
18:    ant(k).tour(n+1) = ant(k).tour(1)
19:    ant(k).cost = ComputeCostFunction(ant(k).tour)
20:    if  ant(k).cost < bestSol.cost then
21:       BestSol = ant(k)
22:    end if
23: end for
```

The Initialization step of **Algorithm** 6, refers to all the initial information required as

mentioned in steps 1 to step 3 in the beginning of section 4.2.3.1 above. Step 3 to step 6 initializes the memory of ants to false (not visited), where m is the number of ants in the system. Step 6 to step 10 uniformly allocates all ants to random cities. In step 11 to step 16, the code calls for **Algorithm** 7, which describes the ants decision process when choosing which cities to visit next. Step 19 computes the cost of the new selected tour, and the remaining steps compare the newly selected tour with the best known solution. We present algorithm 7.

Algorithm 7 Ant system Decision Rule

```
1:  function Ant = ASDecisionRule(Ant,ChoiceMatrix,k,step,n)
2: c = ants(k).tour(step - 1)
3: SumProbabilities ← 0
4: SelectionProbability ← zeros(1,n)
5: for j = 1 to n do
6:    if ants(k).visited(j) = true then
7:       SelectionProbability(j) ← 0
8:    else
9:       SelectionProbability(j) ← ChoiceMatrix(c,j)
10:      SumProbabilities ← SumProbabilities + SelectionProbability(j)
11:   end if
12: end for
13: Roulette Wheel
14: r ← (0 + SumProbabilities) × rand(1)
15: j ← 1
16: p ← SelectionProbability(j)
17: while (p < r) do
18:    j ←j + 1
19:    p ← p + SelectionProbability(j)
20: end while
21: ants(k).tour(step) = j
22: ants(k).visited(r) = true
23: return
```

A number of approaches can be used to describe the decision rule process for ants described in **Algorithm 7.** In this implementation, the ants are synchronized when building a solution. As stated before, we have followed a parallel implementation method. Step 1 of **Algorithm 7** collects necessary information from **Algorithm 6**. New variables like the SumProbabilies, SelectionProbability, and SelectionProbabilities are also created. The purpose for these variables is straight forward from the algorithm. Step 2 to step 12 explains that an ant located at city i, probabilistically chooses to move to an unvisited city j, based on the pheromone trails τ_{ij}^{α} and heuristic information η_{ij}^{β}. In Step 13, we use the roulette wheel procedure, explained in chapter 3, to select the next city which the ant will visit, while in city c. Analogously, one can use equation (4.4) directly to represent step 14 to step 20 of the code.

We now present **Algorithm 8**, which shows how evaporation and pheromone updates are implemented. **Algorithm** 8 describes the pheromone evaporation and pheromone deposit rule as explained in section 4.2.2.2. Step 2 implements equation (4.5). In this step, the value of pheromone trail is decreased on all arcs (i, j) with a constant factor ρ. The ACO procedure requires pheromone to be added again to tours constructed by ants in each iteration. This is shown by step 4 to step 10 of **Algorithm** 8, where m

Algorithm 8 Evaporation and Update pheromone

1: `Evaporation`
2: tau = $\leftarrow (1-\rho) \times \tau$
3: `Pheromone Update`
4: **for** 1 to m **do**
5: tour $\leftarrow$ ant(k).tour
6: **for** l to n **do**
7: i = tour(l)
8: j = tour(l+1)
9: $\tau(i,j) = \tau(i,j) + \frac{1}{ant(k).cost}$
10: **end for**
11: **end for**
12: `Update Choice Matrix`
13: ChoiceMatrix $\leftarrow (\tau\ ^{\alpha}) \times (\eta^{\beta})$

represents the number of ants in the system. Step 13 is optional, as explained above in section 4.2.3, but this is done for computational efficiency.

In **Algorithm** 7, we enforce the decision rule that ants must transition from the current city to the next. We now explain why we use the roulette wheel selection procedure, used in GA, instead of using equation (4.4). The probabilistic choice of the next city works analogously to roulette wheel selection, explained in the previous chapter, i.e., all visible neighbouring cities (step 9, of **Algorithm** 7) of ant k at position j, form a slice on the circular roulette wheel. The other approach is still valid and can be used. In the following subsections, we shall introduce a few ACO implementations and explain how they work.

4.2.4 ACO Implementation

ACO has a number of modifications which have been implemented over the years. We present these improvements below.

(i) Elitist ant. The idea of an elitist approach to an algorithm is not something new. In the context of GA, from Chapter 3, an elitist approach keeps the best performing genome parent in every generation until it is defeated. Dorigo et al. [25] proposed this idea to ACO as well. An elitist ant, in the context of ACO, keeps a record of the best route constructed so far and allocates to it an additional pheromone. Let T^{bs} denote the best tour so far since the start of the algorithm. When updating the pheromone, we add an additional e/C^{bs}, where e is a parameter that defines the weight given to the best tour T^{bs} thus far, and C^{bs} is the best length. Therefore, equation (4.6) becomes:

$$\tau_{ij} \leftarrow \tau_{ij} + \sum_{k=1}^{m} \Delta\tau_{ij}^{k} + e\Delta\tau_{ij}^{bs} \quad \forall (i,j) \in L, \tag{4.8}$$

where, $\Delta\tau_{ij}^{k}$ is given as in AS and $\Delta\tau_{ij}^{bs}$ is added only to the best tour:

$$\Delta\tau_{ij}^{bs} = \begin{cases} \frac{1}{C_{bs}} & \text{if arc}(i,j) \in T^{bs}, \\ 0 & \text{otherwise.} \end{cases} \tag{4.9}$$

(ii) Rank-based ACO

When we have a very good chromosome in GA, and the rest are very poor, implementing the roulette wheel would be an unfair advantage during selection. This is because most trails would favour the one high valued chromosome, ultimately resulting in premature convergence. In rank-based ACO, pheromone depositing is similar to the selection step in the GA, in the sense that the probability of depositing pheromone depends on the fitness of an individual (path). In GA, the ranking method was introduced to remedy this problem. Bullnheimer et al. [11] proposed the AS rank method for ACO and showed in their paper that the ranking system performed better than the elitist AS.

In the rank based method, as with the elitist ant method, the current best ant always deposits slightly more pheromone than the others. For the rest of the ants, pheromone deposit decreases with the ants rank r. The ants are sorted by decreasing tour length, and for a given iteration, only $(w-1)$ best ranked ants are the ones which are allowed to deposit pheromone. Therefore, equation (4.6) becomes:

$$\tau_{ij} \leftarrow \tau_{ij} + \Delta\tau_{ij} + \Delta\tau_{ij}^{bs}. \quad \forall (i,j) \in L, \tag{4.10}$$

where,

$$\Delta\tau_{ij} = \sum_{r=1}^{w-1} \Delta\tau_{ij}^{r}, \tag{4.11}$$

$$\Delta\tau_{ij} = \begin{cases} (w-r)\frac{1}{C^{r}} & \text{if the r-th best ant travels on arc}(i,j), \\ 0 & \text{otherwise,} \end{cases} \tag{4.12}$$

$$\Delta\tau_{ij}^{bs} = \begin{cases} w\frac{1}{C^{bs}} & \text{if arc}(i,j) \quad \text{is part of the best solution found,} \\ 0 & \text{otherwise.} \end{cases} \tag{4.13}$$

(iii) Ant Colony System

Gamberdella and Dorigo [24, 32], introduced another improvement to ACO algorithms called the Ant Colony System (ACS). The ACS outperformed evolutionary

computation (EC) [2] and simulated annealing (SA) [49, 59], on a number of experiments for symmetric and asymmetric TSPs [24]. The ACS has three main improvements:

- It introduces a more aggressive action choice rule which balances between exploration of new edges and exploitation of older ones,
- It follows the elitist ant procedure in that pheromone evaporation, and deposits are only applied to the arcs belonging to the best ant so far,
- It introduces a local pheromone update rule such that each time ant k at position i moves to position j, it removes some pheromone from the arc to increase exploration on other edges.

In ACS, as with AS, all ants (m) are allocated to random cities in the beginning of the algorithm. While constructing its tour, an ant modifies the pheromone on the arcs it traverses by the local pheromone update rule. The pheromone update rule is designed so as to favour edges not yet visited by ants, i.e., an ant k at city i chooses city j based on the following:

$$j = \begin{cases} \mathbf{argmax}_{l \in N_i^k}[\tau_{il}][n_{il}]^\beta & \text{if } q \leq q_0 \quad (exploration), \\ J & \text{otherwise} \quad (exploitation). \end{cases} \tag{4.14}$$

This state transition rule is referred to as the pseudo transition rule (J representing equation (4.4), with $\alpha = 1$). q, which can be any value taken at random with uniform distribution between $[0, 1]$, q_0 $(0 \leq q_0 \leq 1)$, is a parameter, and α, β still play the same roles as in equation (4.4). While building tours, ants apply the local updating rule :

$$\tau_{ij} \leftarrow (1 - \xi)\tau_{ij} + \xi \Delta\tau_{ij}, \tag{4.15}$$

where $\xi \in (0, 1)$ is a parameter. Dorigo et al. [28], experimented on three $\Delta\tau_{ij}$ values:

1. $\Delta\tau_{ij} = \gamma \cdot max_{l \in N_i^k} \tau_{jk}$,
2. $\Delta\tau_{ij} = \tau_0$,
3. $\Delta\tau_{ij} = 0$,

and found that 1 and 2 performed best, while 3 yielded poor results. In ACS, when the local update is given as in 1, the algorithm is known as *AntQ* [24]. Experimentally it was found that a good value for ξ is 0.01 and a good value for τ_0 is $1/nC^{nn}$, where n is the number of cities in the problem, and C^{nn} is the length of a nearest - neighbour [28].

While constructing tours, ants follow the elitist ant procedure by applying the reinforcement on shorter tours by laying more pheromone. The formula describing pheromone is given by:

$$\tau_{ij} \leftarrow (1-\rho)\tau_{ij} + \rho.\Delta\tau_{ij}^{bs}; \tag{4.16}$$

where $\Delta\tau_{ij} = 1/C^{bs}$ if $(i,j) \in$ global best tour and $\Delta\tau_{ij} = 0$ otherwise. C^{bs} is the length of the global best ant so far. Gamberdella and Dorigo [24, 32] showed that it is also possible to substitute it with iterations best length (C^{ib}). Their experimental results show that there is minimal difference between the two schemes, with global best ant dominating.

(iv) Hyper-Cube framework for ACO

The hypercube framework (HCF) by Blum et al. [7], is a different approach from the implementations of the ACO algorithms discussed thus far, the HCF makes the ACO algorithm more robust. Blum et al. [7], and Blum and Dorigo [8], observed that the pheromone values and the performance of the ACO *strongly* depend on the scale of the problem. This is undesirable because two problems which are considered to be isomorphic, would output different results and differ in performance.

Pheromone values are viewed as $|V|$-dimensional vectors $\bar{\tau}$, in a $|V|$-dimensional hyperspace defined by the limits of $\bar{\tau}$. This is shown in equation (4.17):

$$\tau_j = \quad (1-\rho)\tau_j + \sum_{i=1}^{k} \Delta\tau_j^i. \tag{4.17}$$

Equation(4.17) can be rearranged as:

$$\tau_j = \quad \frac{1}{\rho} \cdot \sum_{i=1}^{k} \Delta\tau_j^i,$$

where,

$$\lim_{t\to\inf} \tau_i(t) \leq \quad \frac{1}{\rho} \cdot \frac{k}{f(s^*)}$$

From equation (4.17), it can be observed that the limits of τ depend on $\Delta\tau_j = 1/f(s)$ and on the problem. Therefore in HCF, the hyperspace formed is independent of the quality of the function. The new pheromone rule becomes:

$$\tau_j \leftarrow \rho \cdot \tau_j + (1-\rho) \cdot \sum_{i=1}^{k} \Delta\tau_j^i, \tag{4.18}$$

where

$$\Delta\tau_j^i = \begin{cases} \frac{\frac{1}{f(s^i)}}{\sum_{l=1}^{k} \frac{1}{s^l}} & \text{if} \quad o_j \in s^i, \\ 0 & \text{otherwise.} \end{cases} \tag{4.19}$$

The VRP, as it appears in real life, has many constraints. There are several commercial packages available which model real world problems with more complicated constraints. We now discuss how the ACO is applied to the CVRP.

4.2.5 ACO for Capacitated vehicle routing problem

We begin by looking at approaches to solve CVRPs using ACO.

4.2.5.1 Approaches to solve CVRP with ACO.

If the depot is thought of as some food source, then this setting is similar to the Ant colony system and motivates why the ACO can be used to solve VRPs. The vehicle routing problem can be visualized as multiple TSP paths. Therefore, by grouping different cities according to some criteria, we can solve for the minimum set of TSPs which minimizes the distances. Bullnheimer et al. [12], used this method and showed that this approach is competitive with existing CVRP metaheuristics. In this approach, each ant simulates a vehicle during the construction stage. Following this approach, the artificial ant now has to consider both capacity Q and path feasibility, before constructing the next path. Whenever the choice of another city leads to an infeasible solution (i.e violation of the capacity constraint), the depot is chosen (if infeasible routes are not desired). A new city j, which has not yet been chosen, in the neighborhood of the same ant is selected at random as the starting city of the next circuit. Equations (4.4) to (4.6) are still applicable. The total distance traversed by an individual ant, across all circuits, is the objective function value.

In this method, a *candidate list* can be used to speed up the process. Bullnheimer et al.[12], was the first to propose a candidate list approach for ACOs, which is used to reduce the available options to be explored by an ant during the construction step. This becomes very helpful when looking at large VRPs or TSPs, as it speeds up the solution process. In an instance of a TSP, the candidate list can contain a list of cities closest to the current city i. When an ant k is at this step i, it first chooses which city to visit next, from this candidate list. Once all cities from the candidate list are visited, an ant can explore elsewhere. The improved ant colony optimization algorithm for VRPs describes the candidate list procedure.

4.2.4.2 Improved Ant colony Optimization (IACO) for VRP.

The improved ant colony optimization (IACO) algorithm was suggested by Chen and Ting [14]. This is a direct improvement of the model proposed by Bullnheimer et al [12]. IACO can be applied to very large scaled CVRPs. In our algorithm, we will implement a few ideas from IACO. The IACO consists of the following 3 steps.

- Every ant builds its own solution, and applies the local pheromone on its path,
- Local heuristics are considered,
- Global pheromone information is updated.

During the construction stage, the state transition rule given in equation (4.14) is used. Here, the heuristic information is defined as:

$$\eta_{ij} = d_{i0} + d_{0j} - d_{ij}, \tag{4.20}$$

where η_{ij}, is the saving of combining city i and city j in one tour instead of different tours, d_{ij} is the distance between city i and city j and 0 refers to the depot. The ant still constructs the route by considering its capacity. Once the accumulated demand of visited customers reach the vehicle capacity Q, the depot (v_0) is chosen and the next available city is chosen randomly to start a new circuit. The elitist approach can be applied to improve the quality of the solution found.

Different types of local search methods exist and in the ACS this step is usually done before pheromone levels are updated. IACO exploits this step, and only the best solutions found per iteration are considered in the local search step. By doing this, computation time is saved whilst searching for an improved solution in a good region. Local pheromone in IACO is applied to edges once an ant has completed a tour, this is done using:

$$\tau_{ij} \quad = \quad \tau_{ij} + \rho\tau_0. \tag{4.21}$$

The global updating rule for IACO is carried out a bit differently from ACS. Here, both the best global and local ants are allowed to deposit pheromone per iteration as opposed to choosing either of the two. The idea here is to balance exploration and exploitation of pheromone laying. The Global updating rule is given in (4.16), with $0 \leq \rho \leq 1$, and $\Delta_{ij} = \quad (L_3 - L_g) + (L_3 - L_l)/L_3$, where L_g and L_l denote the tour length of global best solutions and iterations, respectively. L_3 is the best solution corresponding to the current iteration.

4.2.4.3 Two-Stage ACO.

Two-Stage ACO (TACO), developed by Chen and Ting [15], is a two step approach to

solving large VRPs. Customers are mapped to a vehicle, after which the independent TSPs are solved using IACO. In addition, each stage has a pheromone matrix and a different updating rule. IACO is applied in phase two (solving independent TSPs) to speed up the process. In the customer assignment stage, one vehicle is allocated to a single customer at first, and then the rest of the ants are assigned according to some state transitional rule. To determine the number of vehicles which are required in the customer assignment stage, the following equation is used:

$$m = \frac{\sum_{i \in V} q_i}{Q}. \tag{4.22}$$

During the construction step, infeasible solutions, $y \in Y$are allowed, where Y is the set of solutions which satisfy the VRP. The set of roads y is evaluated using a penalty function given by:

$$f(y) = c(y) + w_q q(y), \tag{4.23}$$

where $c(y)$ is the total distance traveled per route, $q(y)$ is the capacity violation and w_q is the positive penalty parameter. i.e. for a feasible solution $f(y) = c(y)$. The violation capacity is calculated as follows:

$$q(y) = \sum_{k \in W} \left[\left(\sum_{i \in V} \sum_{j \in V} q_i x_{ij}^k \right) - Q \right]^+, \tag{4.24}$$

where $[b]^+ = max\{0, b\}$. If a vehicle goes directly to customer j from i, then $x_{ij}^k = 1$; otherwise $x_{ij}^k = 0$.

4.2.6 Proposed Solution Model

To solve the CVRP, the following numerical experiments for constructing vehicle routes were carried out:

(i) First experiment.The ACO algorithm was implemented such that any two cities which were close to each another formed part of the final solution. This method did not work since customers have varying demands.

(ii) Second experiment.The ACO algorithm was implemented with clustering. Here, customers are clustered first, then TSP approaches are used to find the shortest Hamiltonian. Buhrmann [10] presents an interesting study on the effects of clustering on CVRPs. The method did not produce good results, but we did not experiment on a large number of clustering techniques.

(iii) Third experiment. The ACO algorthm was implemented using a model where all ants constructed subtours such that, at a particular cycle, all subtours shared a

common customer. Therefore, the length of a subtour, the accumulated demand, and the cost of each subtour, respectively, differed per ant. These three factors were compared to find the best subtour. Once the first subtour was built, a random available city was chosen and all ants built subtours about this city. This process was done repeatedly. Unfortunately, this approach also did not work well, but we made the following findings:

- The subtour approach is good if we are only minimizing the total cost and not distances as well. Therefore, even if a subtour has the least cost with the maximum number of customers served and full vehicle capacity, this does not guarantee that it is the best subtour.
- The pheromone value and the heuristic information value favour the connection of two cities close to each other. However, without the knowledge of the costs of the ant's total trip, creating subtours alone might not work.
- This approach could form infeasible solutions[2] (Figure 4.2).

After continuous attempts to fix this model, we implemented a different approach.

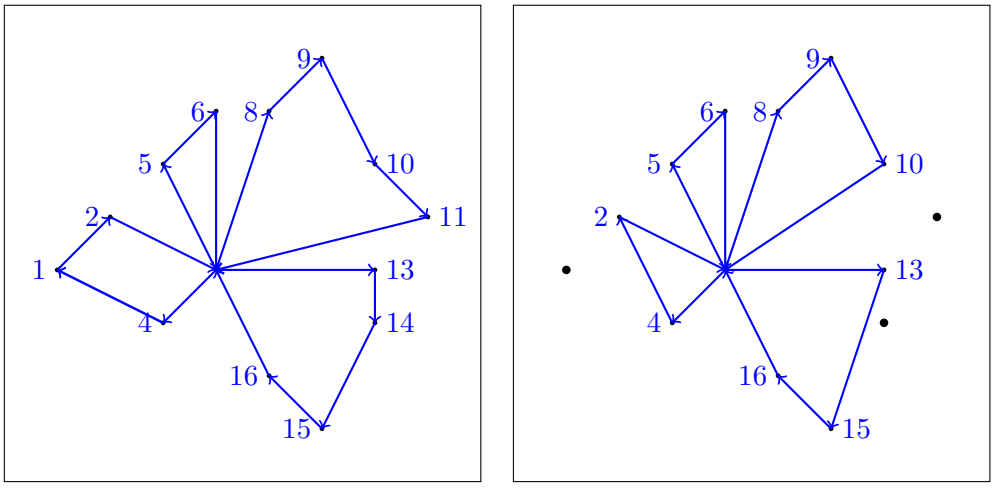

FIGURE 4.2: (*a*) Image on the left has a feasible solution, all customers are visited. (*b*) The image on the right has an feasible solution, all cars are used up, but some customers are not visited

(iv) Fourth experiment. In this model, artificial ants are placed randomly at different customer locations. The artificial ants construct routes by moving from one customer to the next, until all customers have been visited (with this approach no infeasible solution is found). The ant decision rule follows the state transition rule explained in (4.14). Therefore, the pheromone for the traversed path is updated every time an ant

[2]Performing searches may yield good results, depending on the implementation.

crosses said path, and global pheromone is credited only to the best performing ant since initialization of the algorithm (Here the best tour per iteration can still work).

If the accumulated demand of an ant reaches the max capacity (Q) or close to that such that, the next city of interest exceeds this restriction, then the artificial ant's next visit is set to be the depot (v_0). A random city which is not yet visited is selected, and the process continues until the ant has visited all customers. By doing this, different subtours are formed by one single ant; each subtour represents a single path for a vehicle. In our implementation, we also included step 5 (ApplyLocalSearch) of **Algorithm** 5. We used 2-Opt local heuristic search. The 2-Opt algorithm is a basic local search heuristic for TSPs. It improves the tour length by making successive improvements on intersecting edges, exchanging those two with other two edges in the same tour. Although we realized that 2-Opt alone with ACO is not sufficient to yield the optimal path for the CVRP, we proceeded with it. **Algorithm** 9 gives a demonstration of the 2-Opt local heuristic search.

Local heuristics slow down the performance of algorithms, so we have only applied the 2-Opt to ants with a minimum number of subtours. 2-Opt is performed on subtours found by an individual artificial ant, and not to the whole tour. Ants with a minimum number of subtours were explored further, and the best ant in this set is credited with global pheromone.

Algorithm 9 Basic 2-Opt heuristics

1: Create an initial tour $S = (x_1, x_2, \cdots, x_n, x_1)$
2: Set $i = 1$, C be the length of the tour S
3: Set $j = i + 2$
4: Break the link (x_i, x_{i+1}) and (x_j, x_{j+1}), and create a new tour $\bar{S} = (x_1, x_2, \cdots, x_i, x_j, \cdots, x_{i+1}, x_{j+1}, x_{j+2}, \cdots, x_1)$.
5: **if** cost $(\bar{S}) \leq C$ **then**
6: $S := \bar{S}$ and go to step 2, else go to step 8
7: **end if**
8: Set $j = j + 1$.
9: **if** $j < n$ **then**
10: Go to step 4. in the opposite case increase i by 1, if $i \leq (n - 2)$ then go to step 3. Otherwise finish.
11: **end if**

Results for these experiments are shown in Chapter 5.

Chapter 5

Numerical Results and Analysis

5.1 Comparison and Analysis of Results

In this chapter, we present numerical results for the algorithms discussed in Chapters 3 and Chapter 4; the GA and ACO, respectively. To make comparisons of these techniques, we check for the quality of the solution, the efficiency of the algorithm, and the flexibility in implementing the algorithms. We believe that these are the best criteria for comparison of the two approaches.
We test the algorithms on available benchmark instances taken from the CVRP library (CVRP LIB) [1]. These problems are geometric, i.e., they are defined on the Cartesian coordinate system (x-y plane), and distances between them can be computed.
In this section, all tests are performed on a PC with Intel (R) Core $i7-7700$ CPU at 3.60 GHz with 16.0 GB RAM (15.9 GB is usable), running on a 64 bit Operating System, x64 - based processor. All algorithms are coded in MATLAB $R2016b$.
Optimal solutions to most CVRPs are unknown due to their complexity. For this comparison, however, we have selected 19 different benchmarking problems. These problems are taken from two problem sets. The first of these is the test set by Christofides et al. 1979 [1]. The second test set is the set by Uchoa et al. 2014 [1]. A comparison is done on each set independently. The first set contains five problems, viz., CMT1-CMT5 with problem size ranging from 50 to 199 customers, for which solutions are readily available. The second set contains 14 problems with problem sizes ranging from 110 to 700. Of the 14, 4 are problems with unknown optimal solutions.

5.1.1 Results obtained using the Genetic Algorithm

5.1.1.1 Problem set by Christofides et al. [1]

The performance of the GA largely depends on the convergence rate, the population size, and the total number of generations evaluated. Therefore, no criteria is used to determine the number of evaluations and the population size for the GA results presented below. Instead, the initial population was seeded randomly with an initial population ($n = 1000$) for all the instances. Since our algorithm is randomized, each problem is executed 30 times per 1000 generations. The average and the best solutions of the 30 executions per problem, as well as the crossover operator, are presented in Table 5.1. The values of all the parameters for the problem instances are also included. In Tables 5.1 - 5.2, 'W' represents the maximum capacity per vehicle, and 'n' is the size of the problem which represents the number of customers. In columns 5 - 8 'PMX', 'AEX', 'OX' and 'HGreedyX' represent the different crossover operators. 'Best sol' is the best known solutions of the problem and finally '% dev' denotes the percentage deviation from the 'Best sol' and the best performing operator. The best performing operator is represented in bold in each row.

Table 5.1: Solutions obtained for the GA without mutation

Problem	Vehicles	W	n	Crossover operator					
				PMX	AEX	OX	HGreedyX	Best Sol.	% dev.
CMT1	5	160	50	**526.19**	634.34	526.32	543.02	524.61	0.30%
CMT2	10	140	75	943.81	1254.30	975.33	**899.60**	835.36	7.69%
CMT3	8	200	100	916.24	1232.20	936.21	**898.71**	826.14	8.78%
CMT4	12	200	150	1278.10	1903.30	1484.80	**1182.00**	1028.45	14.93%
CMT5	17	200	199	2249.40	2615.20	2205.00	**1562.20**	1291.45	20.94%

In Table 5.2 below, we perform one execution for 5000 generations.

Table 5.2: Solutions obtained for the GA without mutation

Problem	Vehicles	W	n	Crossover operator					
				PMX	AEX	OX	HGreedyX	Best Sol.	% dev.
CMT1	5	160	50	548.39	634.34	**524.62**	527.12	524.61	0.00%
CMT2	10	140	75	896.22	1204.70	**843.82**	899.79	835.36	1.01%
CMT3	8	200	100	892.94	1212.80	**879.70**	885.80	826.14	6.48%
CMT4	12	200	150	1278.10	1909.50	1315.90	**1182.00**	1028.45	14.93%
CMT5	17	200	199	1822.80	2615.20	1940.50	**1579.50**	1291.45	22.30%

From Tables 5.1 - 5.2, it can be concluded that the heuristic greedy crossover (HGreedyX) operator performs best on average, especially when $n > 200$. The order

crossover (OX) seems to outperform all the three operators for only small problem instances, while the PMX seems to outperform OX for larger n.

It is evident that AEX exhibits inferior performance compared to the other three operators. When n increases this deviation increases. Figure 5.1 and Figure 5.2, correspond to Table 5.1 and Table 5.2, respectively. These figures show the percentage deviation changes per problem as n increases for all four selection operators considered.

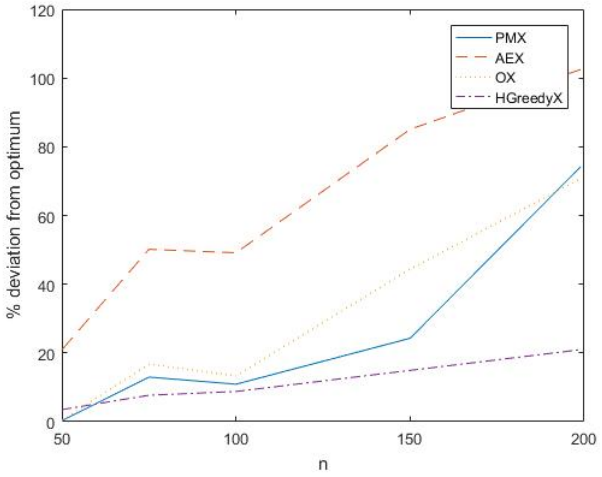

FIGURE 5.1: Percentage deviation from the optimum for different genetic operators in the GA, as per Table 5.1

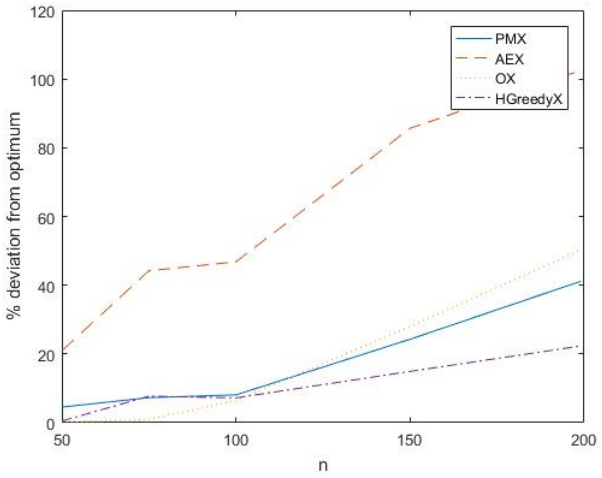

FIGURE 5.2: Percentage deviation from the optimum for different genetic operators in the GA, as per Table 5.2

5.1.1.2 Problem set by Uchoa et al. 2014 [1]

For this problem set, we continue to perform tests for the GA for larger n; $109 \leq n \leq 700$. Similar steps mentioned in Section 5.1.1.1 are followed to set the initial population. Each problem is executed once and is evaluated for 3000 iterations. The best solution per problem as well as the best known solution are presented in Table 5.3. Table 5.3 is structured similarly to Tables 5.1 - 5.2, with the exception of the AEX column. Tests for AEX where not done due to inferior performance exhibited, as shown in Tables 5.1 - 5.2. The results marked with an asterisk are not optimal solutions but are the best known solutions.

TABLE 5.3: Solutions obtained without mutation

Problem	Vehicles	W	n	Crossover operator				
				PMX	OX	HGreedyX	Best Known Sol.	% dev
X-n110-k13	13	66	109	17905	16376	**15957**	14971	6.58%
X-n120-k6	18	21	119	14544	15092	**14598**	13332	9.49%
X-n129-k18	18	39	128	36510	37090	**32362**	28940	11.82%
X-n139-k10	10	106	138	16061	15235	**14761**	13590	8.61%
X-n153-k22	22	144	152	31586	27352	**24588**	21220	15.87%
X-n162-k11	11	1174	161	19484	16275	**16051**	14138	13.53%
X-n172-k5	51	161	171	54928	53782	**50148**	45607	9.95%
X-n181-k23	23	8	180	27670	29884	**27843**	25569	8.89%
X-n190-k8	8	138	189	21025	21196	**20173**	16980	18.80%
X-n200-k36	36	402	199	70338	69232	**63295**	58578	8.05%
X-n280-k17	17	192	279	65228	52021	**46269**	33503*	38.10%
X-n401-k29	29	745	400	101088	95143	**75401**	66187*	13.92%
X-n502-k39	39	13	501	111248	84155	**76067**	69230*	9.87%
X-n701-k44	44	87	700	160560	132210	**103117**	81934*	25.85%

In this problem set, HGreedyX performs best over the entire test set, followed by OX and PMX. The performance of PMX is slightly inferior to that of OX. Figure 5.3 shows the results obtained for Table 5.3. From Figure 5.3, it is observed that even for large n, some problems were easily solved.

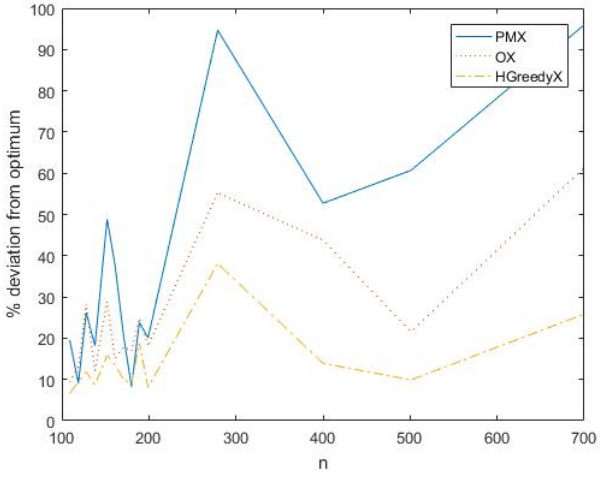

FIGURE 5.3: Percentage deviation from the optimum for the different genetic operator implementations in Table 5.3.

5.1.2 Results obtained using the Ant colony Optimization

5.1.2.1 Problem set by Christofides et al. [1]

In this subsection, we present the ACO metaheuristic results for experiment 4 of section 4.2.6. To investigate how well the proposed implementation performs, the following parameters were used: $q_0 = 0.8$, $\beta = 3$, $\alpha = 1$, $\xi = 0.1$, $\rho = 0.1$. These are parameters discussed in Section 4.2.4, where q_0 is a parameter in the state transition rule, β is heuristic exponential weight, α is the pheromone exponential weight, ξ is a parameter for local update, and ρ is the evaporation rate. The number of ants (m), used in the system ranged from 10 to 25. We increased m by a small fraction and limited it to 25 as the problem size increased. The reason for this increase, is to have more edges explored in an iteration. This, however, affects the computational time of the algorithm. Each problem was solved 50 times, and ran for 1000 iterations. The results obtained are shown in the Table 5.4, together with results of the percentage deviation (% dev) from the optimal solution, the average CPU time, and the average costs per 50 runs. In Table 5.4, the headings in columns 1 through 3 are the same as in Tables 5.1 - 5.2 and the headings in columns 4 through 5 are self explanatory. In columns 7 through 8, 'mean time(s)' denotes the CPU time, and mean cost denotes the average cost per problem on 50 runs. Figure 5.4 shows the percentage deviation of our results from the optimum.

In Table 5.5, we compare the deviation in results recorded obtained by Bullnheimer et al. [12], with the deviation obtained in our implementation. We present Table 5.5 as a basis to compare the performance of our implementation of ACO with that of [12].

TABLE 5.4: Computational results of the CVRP instance

Problem	n	W	Best known Sol.	Our Results	% dev.	mean time(s)	mean cost
CMT1	50	160	524.61	534.31	1.84 %	24.24	571.83
CMT2	75	140	835.36	896.65	7.33%	248.94	935.67
CMT3	100	200	826.14	903.17	9.32%	213.02	943.13
CMT4	150	200	1028.42	1172.50	14.00%	464.97	1201.90
CMT5	199	200	1291.45	1473.90	14.12%	567.09	1515.30

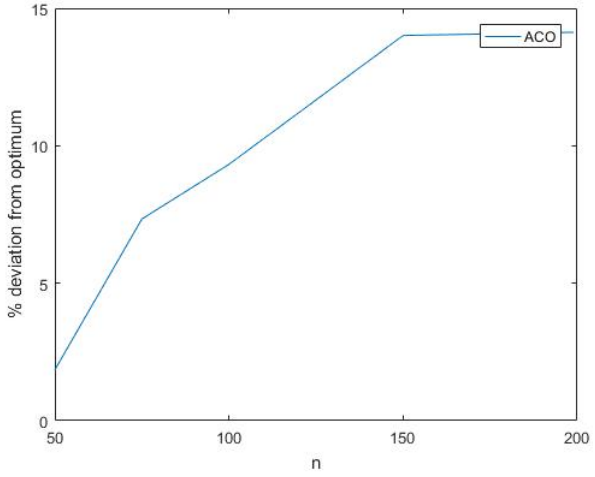

FIGURE 5.4: Percentage deviation from the optimum for ACO, as per Table 5.4

TABLE 5.5: % dev. comparison

Problem	% dev. by [12]	Our % dev.
CMT1	0.00	1.84
CMT2	4.23	7.33
CMT3	6.45	9.32
CMT4	11.57	14.00
CMT5	14.09	14.12

Based on the computational results in Table 5.5, we can deduce that our implementation of the ACO metaheuristic performs relatively well. The relative error of the results obtained and the optimal solution are within the same rage as Bullnheimer et al. [12] results. Bullnheimer et al. [12] presented their paper at the 2nd international conference on metaheuristics. In their paper they compared the ACO algorithm to Tabu Search, Simulated Annealing, and Neural Networks.

5.1.2.2 Problem set by Uchoa et al. 2014 [1]

For this problem set, we continue to perform tests for the ACO but for larger n; $109 \leq n \leq 700$. The same values for the parameters were used as in Table 5.4. The number of ants (m), used is the same as Table 5.4. Column 9 shows the number of vehicles

used to obtain our final results in column 6. We ran each problem for 3000 iterations. The results obtained are shown in the Table 5.6, the columns of this table are similar to those of Table 5.4.

TABLE 5.6: Computational results of the CVRP instance using ACO

Problem	n	W	Vehicles	Best known Sol.	Our Results	% dev.	mean time(s)	Vehicles used
X-n110-k13	110	66	13	14971	16039	7.13%	439	13
X-n120-k6	119	21	6	13332	14392	7.95%	512	6
X-n129-k18	128	39	18	28940	30966	7.00%	312	18
X-n139-k10	138	106	10	13590	14981	10.24%	496	10
X-n153-k22	152	144	22	21220	24802	16.88%	506	23
X-n162-k11	161	1174	11	14138	15899	12.46%	1982	11
X-n172-k51	171	161	51	45607	49519	8.58%	6429	58
X-n181-k23	180	8	23	25569	26840	4.97%	5957	23
X-n190-k8	189	138	8	16980	18143	6.85%	7920	8
X-n200-k36	199	402	36	58578	62760	7.14%	1591	38
X-n280-k17	279	192	17	33503	39427	17.68%	2037	18
X-n401-k29	400	745	29	66187	71238	7.63%	41136	30
X-n502-k39	501	13	39	69230	72500	4.72%	61000	39
X-n701-k44	700	87	44	81934	90606	10.58%	71231	45

In Figure 5.5 we show the percentage deviation our results from the optimum

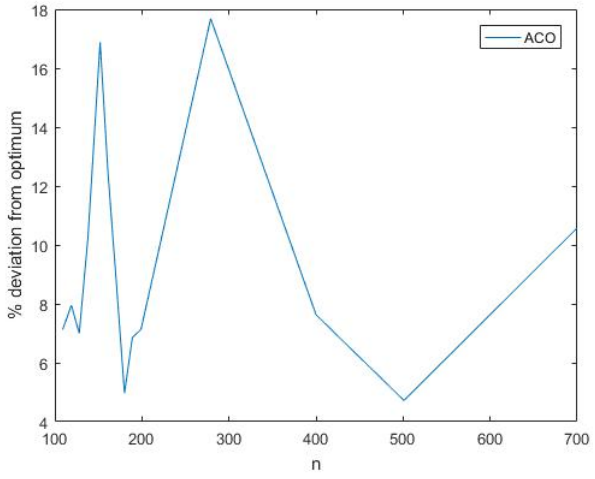

FIGURE 5.5: Percentage deviation from the optimum for ACO, as per Table 5.6

5.1.3 Analysis

In this subsection, analysis and comparison of the results from section 5.1.1 and 5.1.2 is carried out.

5.1.3.1 Efficiency of the Algorithms

The efficiency of an algorithm is often difficult to measure, and among others, we need

to consider the following factors: CPU time, memory capacity, operating system, programming language, parameters of the code, and how the code is implemented. For CVRPs, the complexity of measuring the efficiency of the algorithms increases with the size of the problem. Figure 5.6 below, represents the run time graph of results from Table 5.3 and Table 5.6.

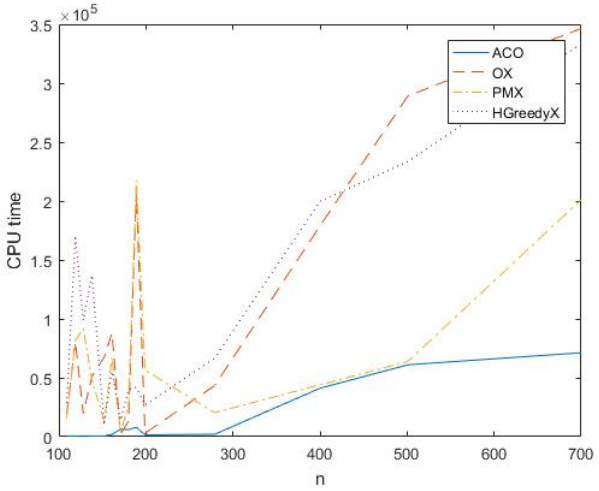

FIGURE 5.6: CPU time required for convergence of the GA and ACO implementations

From our results, the ACO had the smallest run time as compared to all the GA operators, meaning it converged faster than the GA.

5.1.3.2 Analysis

Both the GA, as well as the ACO metaheuristic, solved CVRPs within reasonable time to obtain good quality solutions. We now present two Figures 5.7 - 5.8, followed by a brief discussion of these graphs. In Figure 5.7, we plotted best results of Table 5.2 and Table 5.4 to compare which of our implementations yielded better results.

In Figure 5.8, we plotted best results of Table 5.3 and 5.6 to compare which of our implementations yielded better results.

From Figure 5.7, the quality of solutions of the GA and ACO algorithms is roughly the same when $n < 150$, but this changes as n increases with ACO outperforming the GA. Similar performance is exhibited in Figure 5.8.

We observed that some factors influence the quality of a solution, and the overall performance of the methods considered.

1. For ACO, some of these influencing factors are:

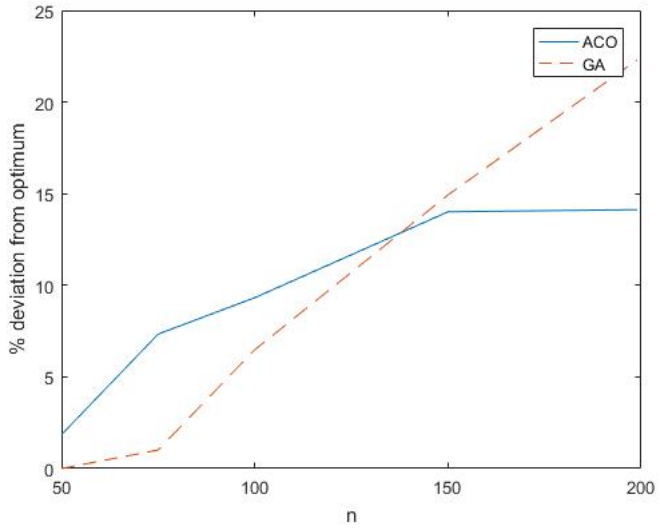

FIGURE 5.7: Percentage deviations from the optimum for the GA and ACO implementations.

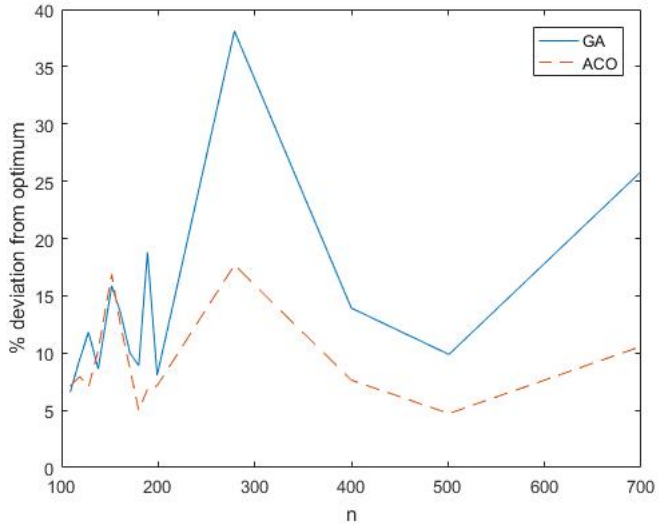

FIGURE 5.8: Percentage deviations from the optimum for the GA and ACO implementations.

- The initial placement of the ants and the number of ants relative to the size of the problem.
- The elitist ant on the system.
- The effect of different parameters on the ACO, such as α and β.
- Initial pheromone impact on the system. (Further studies on the effects of clustering on CVRPs can be carried out)

- The impact of hybridization with ACO, also know as 'the post-optimization' step. Hybridization involves combining more than one algorithm to improve the performance of the main code in terms of the quality of solution.

2. For GA, some of these influencing factors are:

- The size of the initial population, how it is seeded and the chromosome representation.
- The type of selection procedure used.
- The method of crossover operators used.
- Whether mutation is included in the system or not.
- The impact of hybridization with GA (post optimization steps.)

Through continuous testing and experimenting, we discovered that VRPs are very hard and complex to solve, especially when n is large. The ACO metaheuristics have to be modified when applied to CVRP, in order to yield good results.

Even though existing implementations of the methods presented here yield better results than our implementations [3, 14], the results presented in this dissertation are encouraging, and provide a good basis for further exploration.

Chapter 6

Conclusion

The objective of this dissertation was devoted to study the theoretical approach of cutting planes method on CVRP and perform a comparison of techniques on two metaheuristics, namely the GA and ACO by performing computational experiments. In order to achieve the objective, we introduced COPs and discussed the linear assignment problem, the TSP, as well as the CRVP. Throughout the dissertation we discuss complexity issues and different variations of the CVRP in a number of ways.

Chapter 2 contains a study on cutting planes. In particular, studies on valid defining inequalities to the CVRP and the CVRP polytope structure. This structure is complex and its inequalities are not well understood. The motivation for this review is to understand the structure of CVRPs, which is helpful during the investigation of exact methods for NP-complete problems. We have investigated the concept of cutting planes, starting with the basic Gomory cuts for integer programming, followed by an in depth study of valid inequalities for STSP polytope structures. We then investigated cutting planes for CVRPs, by considering different valid inequalities and their effectiveness.

A comparative study of two metaheuristics for solving CVRPs, viz., GA and ACO, was also discussed. Extensive numerical experiments have been presented for both methods.

In Chapter 3, we discussed the GA. We first discussed the background of the metaheuristic, as well as the purpose and motivation for investigating the performance of this method. The GA can be implemented in various ways, and no single implementation is superior to the other for every problem instance. We, therefore, decided to test different crossover operators in the implementation of the GA, to determine which operators worked best when applied to CVRPs. This was advantageous because most operators were built to solve TSPs and their performance

is not recorded for CVRPs. The four operators used for the implementation of the GA were: the partial mapped crossover, the order crossover, the alternative edge crossover, and the heuristic greedy crossover. These operators were built for CVRP and tested on 19 benchmark instances. Results were presented in chapter 5.

In Chapter 4, the ACO algorithm is discussed. We introduce this metaheuristic by describing its origin (real ants) and the transition from there onto artificial ants. The metaheuristic is first explained through the TSP, and is then extended for CVRPs. Detailed ACO implementation steps were presented for CVRP. Results of the ACO implementation presented in this section are provided in Chapter 5, where they are compared to the GA.

In Chapter 5, algorithms discussed in Chapter 4 and Chapter 5 are tested on 19 Benchmark problems. Results show that ACO outperforms the GA marginally for small scale problems, and exhibits superior performance for larger problems.

6.1 Future work

The main area of improvement in this dissertation includes extending the theory of existing cutting planes, and conducting numerical experiments on the performance of cutting planes for CVRPs.

Bibliography

[1] Capacitated Vehicle Routing Problems Library, retrieved 24-03-2018, from http://vrp.atd-lab.inf.puc-rio.br/index.php/en/.

[2] Thomas Bäck, David B Fogel, and Zbigniew Michalewicz. *Handbook of evolutionary computation*. CRC Press, 1997.

[3] Barrie M Baker and MA Ayechew. A genetic algorithm for the vehicle routing problem. *Computers & Operations Research*, 30(5):787–800, 2003.

[4] James Edward Baker. Adaptive selection methods for genetic algorithms. In *Proceedings of an International Conference on Genetic Algorithms and their applications*, pages 101–111. Hillsdale, New Jersey, 1985.

[5] Egon Balas, Sebastián Ceria, and Gérard Cornuéjols. A lift-and-project cutting plane algorithm for mixed 0–1 programs. *Mathematical programming*, 58(1-3):295–324, 1993.

[6] Norman Biggs, E Keith Lloyd, and Robin J Wilson. *Graph Theory, 1736-1936*. Oxford University Press, 1976.

[7] Christian Blum and Marco Dorigo. The hyper-cube framework for ant colony optimization. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 34(2):1161–1172, 2004.

[8] Christian Blum, Andrea Roli, and Marco Dorigo. Hc–aco: The hyper-cube framework for Ant Colony Optimization. In *Proceedings of MIC*, volume 2, pages 399–403, 2001.

[9] Sylvia C. Boyd and William H. Cunningham. Small Travelling Salesman Polytopes. *Math. Oper. Res.*, 16:259–271, 1991.

[10] Jacoba Hendrina Buhrmann. *The effects of clustering on the medium and large-scale capacitated location-routing problem*. PhD thesis, 2016.

[11] Bernd Bullnheimer, Richard F Hartl, and Christine Strauss. A new rank based version of the Ant System. A computational study. 1997.

[12] Bernd Bullnheimer, Richard F Hartl, and Christine Strauss. Applying the ant system to the vehicle routing problem. In *Meta-heuristics*, pages 285–296. Springer, 1999.

[13] Robert D Carr. Separating clique tree and bipartition inequalities in polynomial time. In *International Conference on Integer Programming and Combinatorial Optimization*, pages 40–49. Springer, 1995.

[14] Chia-Ho Chen and Ching-Jung Ting. An improved ant colony system algorithm for the vehicle routing problem. *Journal of the Chinese institute of industrial engineers*, 23(2):115–126, 2006.

[15] Chen Chia-Ho and Ting Ching-Jung. Applying two-stage ant colony optimization to solve the large scale vehicle routing problem. *Journal of the Eastern Asia Society for Transportation Studies*, 8:761–776, 2010.

[16] Václav Chvátal. Edmonds polytopes and weakly Hamiltonian graphs. *Mathematical programming*, 5(1):29–40, 1973.

[17] Edward G Coffman Jr, Michael R Garey, and David S Johnson. Approximation algorithms for bin-packing—an updated survey. In *Algorithm design for computer system design*, pages 49–106. Springer, 1984.

[18] Gérard Cornuéjols, Jean Fonlupt, and Denis Naddef. The traveling salesman problem on a graph and some related integer polyhedra. *Mathematical programming*, 33(1):1–27, 1985.

[19] George Dantzig, Ray Fulkerson, and Selmer Johnson. Solution of a large-scale traveling-salesman problem. *Journal of the operations research society of America*, 2(4):393–410, 1954.

[20] George B Dantzig and John H Ramser. The truck dispatching problem. *Management science*, 6(1):80–91, 1959.

[21] Charles Darwin and William F Bynum. *The origin of species by means of natural selection: or, the preservation of favored races in the struggle for life*. AL Burt, 2009.

[22] J-L Deneubourg, Serge Aron, Simon Goss, and Jacques M Pasteels. The self-organizing exploratory pattern of the argentine ant. *Journal of insect behavior*, 3(2):159–168, 1990.

[23] Marco Dorigo and Luca Maria Gambardella. A study of some properties of Ant-Q. In *International Conference on Parallel Problem Solving from Nature*, pages 656–665. Springer, 1996.

[24] Marco Dorigo and Luca Maria Gambardella. Ant colony system: a cooperative learning approach to the traveling salesman problem. *IEEE Transactions on evolutionary computation*, 1(1):53–66, 1997.

[25] Marco Dorigo, Vittorio Maniezzo, and Alberto Colorni. The ant system: An autocatalytic optimizing process. 1991.

[26] Marco Dorigo, Vittorio Maniezzo, and Alberto Colorni. Ant system: optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 26(1):29–41, 1996.

[27] Marco Dorigo, Vittorio Maniezzo, Alberto Colorni, and Vittorio Maniezzo. Positive feedback as a search strategy. 1991.

[28] Marco Dorigo and Thomas Stutzle. From Real to artificial ants. *MIT Press Book*, 2004.

[29] Jack Edmonds and Ellis L Johnson. Matching, Euler tours and the Chinese postman. *Mathematical programming*, 5(1):88–124, 1973.

[30] Marshall L Fisher. A polynomial algorithm for the degree-constrained minimum k-tree problem. *Operations Research*, 42(4):775–779, 1994.

[31] Bernhard Fleischmann. A cutting plane procedure for the travelling salesman problem on road networks. *European Journal of Operational Research*, 21(3):307–317, 1985.

[32] Luca Maria Gambardella and Marco Dorigo. Solving symmetric and asymmetric TSPs by ant colonies. In *Evolutionary Computation, 1996., Proceedings of IEEE International Conference on*, pages 622–627. IEEE, 1996.

[33] K Ganesh, Sanjay Mohapatra, et al. Models for practical routing problems in logistics. 2014.

[34] David E Goldberg, Robert Lingle, et al. Alleles, loci, and the traveling salesman problem. In *Proceedings of an international conference on genetic algorithms and their applications*, volume 154, pages 154–159. Lawrence Erlbaum, Hillsdale, NJ, 1985.

[35] Ralph E Gomory. An algorithm for integer solutions to linear programs. Princeton IBM Mathematics Research Project. *Techn. Report*, (1), 1958.

[36] Ralph E Gomory and Tien Chung Hu. Multi-terminal network flows. *Journal of the Society for Industrial and Applied Mathematics*, 9(4):551–570, 1961.

[37] Simon Goss, Serge Aron, Jean-Louis Deneubourg, and Jacques Marie Pasteels. Self-organized shortcuts in the Argentine ant. *Naturwissenschaften*, 76(12):579–581, 1989.

[38] John Grefenstette. Proportional selection and sampling algorithms. *Evolutionary computation*, 1:172–180, 2000.

[39] John Grefenstette, Rajeev Gopal, Brian Rosmaita, and Dirk Van Gucht. Genetic algorithms for the traveling salesman problem. In *Proceedings of the first International Conference on Genetic Algorithms and their Applications*, pages 160–168. Lawrence Erlbaum, New Jersey (160-168), 1985.

[40] Martin Grötschel. On the symmetric travelling salesman problem: solution of a 120-city problem. *Combinatorial Optimization*, pages 61–77, 1980.

[41] Martin Grötschel and Manfred Padberg. On the symmetric travelling salesman problem i. *Mathematical Programming*, 16:265 – 280, 1979.

[42] Martin Grötschel and Manfred W Padberg. On the symmetric travelling salesman problem I: inequalities. *Mathematical Programming*, 16(1):265–280, 1979.

[43] Martin Grötschel and William R Pulleyblank. Clique tree inequalities and the symmetric travelling salesman problem. *Mathematics of operations research*, 11(4):537–569, 1986.

[44] Jianxiu Hao and James B Orlin. A faster algorithm for finding the minimum cut in a graph. In *Proceedings of the third annual ACM-SIAM symposium on Discrete algorithms*, pages 165–174. Society for Industrial and Applied Mathematics, 1992.

[45] Randy L Haupt and Sue Ellen Haupt. *Practical genetic algorithms*. John Wiley & Sons, 2004.

[46] John H Holland. Genetic algorithms and the optimal allocation of trials. *SIAM Journal on Computing*, 2(2):88–105, 1973.

[47] Toshimde Ibaraki. Integer programming formulation of combinatorial optimization problems. *Discrete Mathematics*, 16(1):39–52, 1976.

[48] Michael Jünger, Gerhard Reinelt, and Giovanni Rinaldi. The traveling salesman problem. *Handbooks in operations research and management science*, 7:225–330, 1995.

[49] Scott Kirkpatrick, C Daniel Gelatt, and Mario P Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[50] Alan Kirman. Ants, rationality, and recruitment. *The Quarterly Journal of Economics*, 108(1):137–156, 1993.

[51] Ailsa H Land and Alison G Doig. An automatic method of solving discrete programming problems. *Econometrica: Journal of the Econometric Society*, pages 497–520, 1960.

[52] Gilbert Laporte, Yves Nobert, and Serge Taillefer. Solving a family of multi-depot vehicle routing and location-routing problems. *Transportation science*, 22(3):161–172, 1988.

[53] Eugene L Lawler, Jan Karel Lenstra, AH-G Rinnooy-Kan, and David B Shmoys. The traveling salesman problem. 1985.

[54] Jon Lee. *A first course in combinatorial optimization*, volume 36. Cambridge University Press, 2004.

[55] Adam Nicholas Letchford. *Polyhedral results for some constrained arc-routing problems*. PhD thesis, University of Lancaster, 1996.

[56] Sean Luke. Essentials of metaheuristics. lulu, 2009. *Available for free at http://cs. gmu. edu/sean/book/metaheuristics/. There is no corresponding record for this reference*, 2011.

[57] Vittorio Maniezzo and Antonella Carbonaro. An ANTS heuristic for the frequency assignment problem. *Future Generation Computer Systems*, 16(8):927–935, 2000.

[58] Silvano Martello and Paolo Toth. Lower bounds and reduction procedures for the bin packing problem. *Discrete applied mathematics*, 28(1):59–70, 1990.

[59] Nicholas Metropolis, Arianna W Rosenbluth, Marshall N Rosenbluth, Augusta H Teller, and Edward Teller. Equation of state calculations by fast computing machines. *The journal of chemical physics*, 21(6):1087–1092, 1953.

[60] Clair E Miller, Albert W Tucker, and Richard A Zemlin. Integer programming formulation of traveling salesman problems. *Journal of the ACM (JACM)*, 7(4):326–329, 1960.

[61] Denis Naddef and Yves Pochet. The symmetric traveling salesman polytope revisited. *Mathematics of Operations Research*, 26(4):700–722, 2001.

[62] Hiroshi Nagamochi and Toshihide Ibaraki. Computing edge-connectivity in multigraphs and capacitated graphs. *SIAM Journal on Discrete Mathematics*, 5(1):54–66, 1992.

[63] George L Nemhauser and Laurence A Wolsey. Integer programming and combinatorial optimization. *Wiley, Chichester. GL Nemhauser, MWP Savelsbergh, GS Sigismondi (1992). Constraint Classification for Mixed Integer Programming Formulations. COAL Bulletin*, 20:8–12, 1988.

[64] Manfred Padberg and Giovanni Rinaldi. An efficient algorithm for the minimum capacity cut problem. *Mathematical Programming*, 47(1):19–36, 1990.

[65] Manfred Padberg and Giovanni Rinaldi. A branch-and-cut algorithm for the resolution of large-scale symmetric traveling salesman problems. *SIAM review*, 33(1):60–100, 1991.

[66] Manfred W Padberg and M Ram Rao. Odd minimum cut-sets and b-matchings. *Mathematics of Operations Research*, 7(1):67–80, 1982.

[67] Hari Mohan Pandey, Ankit Chaudhary, and Deepti Mehrotra. A comparative review of approaches to prevent premature convergence in GA. *Applied Soft Computing*, 24:1047–1077, 2014.

[68] Christos H Papadimitriou and Kenneth Steiglitz. *Combinatorial optimization: algorithms and complexity*. Courier Corporation, 1982.

[69] Gábor Pataki. Teaching integer programming formulations using the traveling salesman problem. *SIAM review*, 45(1):116–123, 2003.

[70] Christian Prins, Philippe Lacomme, and Caroline Prodhon. Order-first split-second methods for vehicle routing problems: A review. *Transportation Research Part C: Emerging Technologies*, 40:179–200, 2014.

[71] Giovanni Righini. Approximation algorithms for the vehicle routing problem with pick-up and delivery. *Note del Polo-Ricerca*, 33, 2000.

[72] Nitasha Soni and Tapas Kumar. Study of various mutation operators in genetic algorithms. *International Journal of Computer Science and Information Technologies*, 5(3):4519–4521, 2014.

[73] Thomas Stützle and Holger H Hoos. 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*, 1996.

[74] Paolo Toth and Daniele Vigo. *Vehicle routing: problems, methods, and applications*, volume 18. Siam, 2014.

[75] Sewall Wright. Evolution in Mendelian populations. *Genetics*, 16(2):97–159, 1931.