Investigating Heuristic and Meta-Heuristic Algorithms for Solving Pickup and Delivery Problems

A thesis submitted in partial fulfilment
of the requirement for the degree of Doctor of Philosophy

Manar Ibrahim Hosny

March 2010

Cardiff University
School of Computer Science & Informatics

Declaration

This work has not previously been accepted in substance for any degree and is not concurrently submitted in candidature for any degree.

Signed ..... (candidate)

Date .....

Statement 1

This thesis is being submitted in partial fulfillment of the requirements for the degree of PhD.

Signed ..... (candidate)

Date .....

Statement 2

This thesis is the result of my own independent work/investigation, except where otherwise stated. Other sources are acknowledged by explicit references.

Signed ..... (candidate)

Date .....

Statement 3

I hereby give consent for my thesis, if accepted, to be available for photocopying and for inter-library loan, and for the title and summary to be made available to outside organisations.

Signed ..... (candidate)

Date .....

Abstract

The development of effective decision support tools that can be adopted in the transportation industry is vital in the world we live in today, since it can lead to substantial cost reduction and efficient resource consumption. Solving the Vehicle Routing Problem (VRP) and its related variants is at the heart of scientific research for optimizing logistic planning. One important variant of the VRP is the Pickup and Delivery Problem (PDP). In the PDP, it is generally required to find one or more minimum cost routes to serve a number of customers, where two types of services may be performed at a customer location, a pickup or a delivery. Applications of the PDP are frequently encountered in every day transportation and logistic services, and the problem is likely to assume even greater prominence in the future, due to the increase in e-commerce and Internet shopping.

In this research we considered two particular variants of the PDP, the Pickup and Delivery Problem with Time Windows (PDPTW), and the One-commodity Pickup and Delivery Problem (1-PDP). In both problems, the total transportation cost should be minimized, without violating a number of pre-specified problem constraints.

In our research, we investigate heuristic and meta-heuristic approaches for solving the selected PDP variants. Unlike previous research in this area, though, we try to focus on handling the difficult problem constraints in a simple and effective way, without complicating the overall solution methodology. Two main aspects of the solution algorithm are directed to achieve this goal, the *solution representation* and the *neighbourhood moves*.

Based on this perception, we tailored a number of heuristic and meta-heuristic algorithms for solving our problems. Among these algorithms are: Genetic Algorithms, Simulated Annealing, Hill Climbing and Variable Neighbourhood Search. In general, the findings of the research indicate the success of our approach in handling the difficult problem constraints and devising simple and robust solution mechanisms that can be integrated with vehicle routing optimization tools and used in a variety of real world applications.

Acknowledgements

This thesis would not have been possible without the help and support of the kind people around me. I would like to extend my thanks to all individuals who stood by me throughout the years of my study, but I wish to particulary mention a number of people who have had a profound impact on my work.

Above all, I am heartily thankful to my supervisor, Dr. Christine Mumford, whose encouragement, guidance and support from start to end enabled me to develop my research skills and understanding of the subject. Her wealth of knowledge, sincere advice and friendship with her students made this research fruitful as well as enjoyable.

It is an honour for me to extend my thanks to my employer King Saud University (KSU), Riyadh, Saudi Arabia, for offering my postgraduate scholarship, and providing all financial support needed to complete my degree. I am also grateful to members of the IT Department of the College of Computers and Information Sciences at KSU, for their support and encouragement during the years of my PhD program.

I would like to acknowledge the academic and technical support of the School of Computer Science & Informatics at Cardiff University, UK and all its staff members. I am also grateful for the kindness, help and support of all my colleagues at Cardiff University.

Last but by no means least, I owe my deepest gratitude to my parents, my husband, and my children for their unconditional support and patience from the beginning to the end of my study.

Contents

List of Figures

List of Tables

List of Algorithms

Acronyms

1M	One Level Exchange Mutation
1-PDP	One Commodity Pickup and Delivery Problem
ACO	Ant Colony Optimization
ALNS	Adaptive Large neighbourhood Search
AVNS-SA	Adaptive Variable neighbourhood Search - Simulated Annealing
BF	Best Fit
CDARP	Capacitated Dial-A-Ride Problem
CEL	centre TW - Early TW - Late TW
CKKL	Evolutionary Approach to the MV-PDPTW in [33]
CL	Number of Closest neighbours
CLE	centre TW - Late TW - Early TW
CPP	Chinese Postman Problem
CTSPPD	Capacitated Traveling Salesman Problem with Pickup and Delivery
CV	Capacity Violations
CVRP	Capacitated Vehicle Routing Problem
DARP	Dial-A-Ride Problem
DLS	Descent Local Search
DP	Dynamic Programming
ECL	Early TW - centre TW - Late TW
ELC	Early TW - Late TW - centre TW
EPA	Exact Permutation Algorithm
EPS	Evolutionary Perturbation Scheme
FCGA	Family Competition Genetic Algorithm
GA	Genetic Algorithm
GALIB	Genetic Algorithm Library
GDP	Gross Domestic Product
GGA	Grouping Genetic Algorithm
GIS	Geographic Information Systems
GLS	Guided Local Search

GPS	Geographic Positioning Systems
GRASP	Greedy Randomized Adaptive Search Procedure
GRGA	Grouping Routing Genetic Algorithm
GRS	Greedy Random Sequence
GUI	Graphical User Interface
HC	Hill Climbing
HPT	Handicapped Person Transportation
IBX	Insertion-Based Crossover
ILS	Iterated Local Search
IMSA	Iterated Modified Simulated Annealing
IRNX	Insert-within-Route Crossover
LC1	Li & Lim benchmark instances [100]- clustered customers- short TW
LC2	Li & Lim benchmark instances [100]- clustered customers- long TW
LCE	Late TW - centre TW - Early TW
LEC	Late TW - Early TW -centre TW
LNS	Large neighbourhood Search
LR1	Li & Lim benchmark instances [100]- random customers- short TW
LR2	Li & Lim benchmark instances [100]- random customers- long TW
LRC1	Li & Lim benchmark instances [100]- random & clustered customers - short TW
LRC2	Li & Lim benchmark instances [100] - random & clustered customers - long TW
LS	Local Search
LSM	Local Search Mutation
MSA	Modified Simulated Annealing
MTSP	Multiple Vehicle Traveling Salesman Problem
MV	Multiple Vehicle
MV-PDPTW	Multiple Vehicle Pickup and Delivery Problem with Time Windows
MX1	Merge Crossover
NN	Nearest neighbour
$\mathcal{NP}$	Non Deterministic Polynomial Time Problems
OR	Operations Research
OR/MS	Operations Research/ Management Science
$\mathcal{P}$	Polynomial Time Problems
PBQ	Parallel Construction Best Request
PBR	Parallel Construction Best Route
PD	Pickup and Delivery

PDP	Pickup and Delivery Problem
PDPTW	Pickup And Delivery Problem with Time Windows
PDPX	Pickup And Delivery Crossover
PDTSP	Pickup And Delivery Traveling Salesman Problem
PDVRP	Pickup And Delivery Vehicle Routing Problem
PF	Push Forward
PFR	Parallel Construction First Route
PMX	Partially Matched Crossover
PS	Predecessor-Successor
PTS	Probabilistic Tabu Search
PVRPTW	Periodic Vehicle Routing Problem with Time Widows
QAP	Quadratic Assignment Problem
RBX	Route Based Crossover
RCL	Restricted Candidate List
SA	Simulated Annealing
SAT	Boolean Satisfiability Problem
SBR	Swapping Pairs Between Routes
SBX	Sequence Based Crossover
SDARP	Single Vehicle Dial-A-Ride Problem
SEQ	Sequential Construction
SPDP	Single Vehicle Pickup and Delivery Problem
SPI	Single Paired Insertion
SV	Single Vehicle
SV-PDPTW	Single Vehicle Pickup and Delivery Problem with Time Windows
TA	Threshold Accepting
TS	Tabu Search
TSP	Traveling Salesman Problem
TSPB	Traveling Salesman Problem with Backhauls
TSPCB	Traveling Salesman Problem with Clustered Backhauls
TSPDDP	Traveling Salesman Problem with Divisible Delivery and Pickup
TSPMB	Traveling Salesman Problem with Mixed Linehauls and Backhauls
TSPPD	Traveling Salesman Problem with Pickup and Delivery
TSPSDP	Traveling Salesman Problem with Simultaneous Delivery and Pickup
TW	Time Window
TWV	Time Window Violations
UOX	Uniform Order Based Crossover

VCX	Vehicle Copy Crossover
VMM	Vehicle Merge Mutation
VMX	Vehicle Merge Crossover
VND	Variable neighbourhood Descent
VNS	Variable neighbourhood Search
VRP	Vehicle Routing Problem
VRPB	Vehicle Routing Problem with Backhauls
VRPCB	Vehicle Routing Problem with Clustered Backhauls
VRPDDP	Vehicle Routing Problem with Divisible Delivery and Pickup
VRPMB	Vehicle Routing Problem with Mixed Linehauls and Backhauls
VRPPD	Vehicle Routing Problem with Pickups and Deliveries
VRPSDP	Vehicle Routing Problem with Simultaneous Delivery and Pickup
VRPTW	Vehicle Routing Problem with Time Widows
WRI	Within Route Insertion

Chapter 1

Introduction

Efficient transportation and logistics plays a role in the economic wellbeing of society. Almost everything that we use in our daily lives involves logistic planning, and practically all sections of industry and society, from manufacturers to home shopping customers, require the effective and predictable movement of goods. In today's dynamic business environment, transportation cost constitutes a significant percentage of the total cost of a product. In fact, it is not unusual for a company to spend more than 20% of the product's value on transportation and logistics [76]. In addition, the transportation sector itself is a significant industry, and its volume and impact on society as a whole continues to increase every day. In a 2008 report, the UK Department of Transport estimates that freight and logistics sector is worth £74.5 billion to the economy and employs 2.3 million people across 190,000 companies [2].

In the last few decades, advancement in transportation and logistics has greatly improved people's lives and influenced the performance of almost all economic sectors. Nevertheless, it also produced negative impacts. Approximately two thirds of our goods are transported through road transport [2]. However, vehicles moving on our roads contribute to congestion, noise, pollution, and accidents. Reducing these impacts requires cooperation between the various stakeholders in manufacturing and distribution supply chains for more efficient planning and resource utilization. Efficient vehicle routing is essential, and automated route planning and transport management, using optimization tools, can help reduce transport costs by cutting mileage and improving driver and vehicle usage. In addition, it can improve customer service, cut carbon emissions, improve strategic decision making and reduce administration costs.

Research studying effective planning and optimization in the vehicle routing and scheduling field has increased tremendously in the last few decades [48] . Advances in technology and computational power has encouraged researchers to consider various problem types and real-life constraints, and to experiment with new algorithmic techniques that can be applied for the automation of vehicle planning activities. A number of these techniques have been implemented in commercial optimization software and successfully used in

every day transportation and logistics applications. However, due to the large increase in demand and the growing complexity of this sector, innovations in solution techniques that can be used to optimize vehicle routing and scheduling are still urgently needed.

Our research is concerned with developing efficient algorithmic techniques that can be applied in vehicle routing and planning activities. In Section 1.1 we introduce the research problems that we are trying to tackle and emphasize the motivation behind our selection of these problems. Section 1.2 highlights the hypothesis and contribution of the thesis. Section 1.3 gives an overview of the contents of this thesis and indicates the publications related to each part of the research. Finally, some brief remarks about the computational experimentation performed in this research are presented in Section 1.4.

1.1 Research Problem and Motivation

The Vehicle Routing Problem (VRP) is at the heart of scientific research involving the distribution and transportation of people and goods. The problem can be generally described as finding a set of minimum cost routes for a fleet of vehicles, located at a central depot, to serve a number of customers. Each vehicle should depart from and return to the same depot, while each customer should be visited exactly once. Since the introduction of this problem by Dantzig and Fulkerson in 1954 [35] and Dantzig and Ramser in 1959 [35], several extensions and variations have been added to the basic VRP model, in order to meet the demand of realistic applications that are complex and dynamic in nature. For example, restricting the capacity of the vehicle and specifying certain service times for visiting clients are among the popular constraints that have been considered while addressing the VRP.

One important variant of the VRP is the Pickup and Delivery Problem (PDP). Unlike the classical VRP, where all customers require the same service type, a central assumption in the PDP is that there are two different types of services that can be performed at a customer location, a pickup or a delivery. Based on this assumption, other variants of the PDP have also been introduced depending on, for example, whether the origin/destination of the commodity is the depot or some customer location, whether one or multiple commodities are transferred, whether origins and destinations are paired, and whether people or goods are transported. In addition, some important real-life constraints that have been applied to the basic VRP have also been applied to the PDP, such as the vehicle capacity and service time constraints.

Applications of the PDP are frequently seen in transportation and logistics. In addition,

the problem is likely to become even more important in the future, due to the rapid growth in parcel transportation as a result of e-commerce. Some applications of this problem include bus routing, food and beverage distribution, currency collection and delivery between banks and ATM machines, Internet-based pickup and delivery, collection and distribution of charitable donations between homes and different organizations, and the transport of medical samples from medical offices to laboratories, just to name a few. Also, an important PDP variant, known as dial-a-ride, can provide an effective means of transport for delivering people from door to door. This model is frequently adopted for the disabled, but could provide a greener alternative than the car and taxi to the wider community. Besides road-based transportation, applications of the PDP can also be seen in maritime and airlift planning.

As previously mentioned, considerable attention has been paid to the classical VRP and its related variants by the scientific community. Literally thousands of papers have been published in this domain (see for example survey papers [142], [96], [48], [147] and the survey paper in the 50th anniversary of the VRP [97]). Despite this, research on Pickup and Delivery (PD) problems is relatively scarce [137]. A possible reason is the complexity of these problems and the difficulty in handling the underlying problem constraints. Innovations in solution methods that handle different types of PD problems are certainly in great demand.

We have selected two important variants of PD problems as the focus of this research. These are: **the Pickup and Delivery Problem with Time Windows (PDPTW)**, and **the One-commodity pickup and delivery problem (1-PDP)**, with more emphasis on the first of these two variants and a thorough investigation of both its single and multiple vehicle cases. The main difference between the two problems is that the PDPTW assumes that origins and destinations are paired, while the 1-PDP assumes that a single commodity may be picked up from any pickup location and delivered to any delivery location. In fact, the problems dealt with in this research are regularly encountered in every day transportation and logistics applications, as will be demonstrated during the course of this thesis.

1.2 Thesis Hypothesis and Contribution

Similar to the VRP, the PDP belongs to the class of Combinatorial Optimization (CO) problems, for which there is usually a huge number of possible solution alternatives. For example, the simplest form of the VRP is the Traveling Salesman Problem (TSP), where a salesman must visit n cities starting and ending at the same city, and each city should be visited exactly once. It is required to find a certain route for the salesman such that the

total traveling cost is minimized. Although this problem is simple for small values of n, the complexity of solving the problem increases quickly as n becomes large. For a TSP problem of size n, the number of possible solutions is $n!/2$. So, if $n = 60$ for example, the number of possible solutions that must be searched is approximately 4.2×10^{81}.

If one considers the VRP with added problem constraints, the solution process becomes even more complex. Despite the advancement in algorithmic techniques, solving the VRP and similar highly constrained CO problems to optimality may be prohibitive for very large problem sizes. Exact algorithms that may be used to provide optimum problem solutions cannot solve VRP instances with more than 50-100 customers, given the current resources [72]. Approximation algorithms that make use of heuristic and meta-heuristic approaches are often used to solve problems of practical magnitudes. Such approaches provide good solutions to CO problems in a reasonable amount of time, compared to exact methods, with no guarantee to solve the problem to optimality (see Chapter 2 for more details about combinatorial optimization, complexity theory and exact and approximation techniques).

Many heuristic and meta-heuristic algorithms have been applied to solving the different variants of PD problems. However, most of these approaches are adaptations of algorithms that have been previously developed for the classical VRP. The solution process for such problems is usually divided into two phases: *solution construction* and *solution improvement*. In the solution construction phase one or more initial problem solutions is generated, while in the solution improvement phase the initial solution(s) is gradually improved, in a step-by-step fashion, using a heuristic or a meta-heuristic approach. Applying classical VRP techniques to the PDP, though, requires careful consideration, since the PDP is in fact a highly constrained problem that is much harder than other variants of the VRP.

We noticed during our literature survey that research tackling PD problems tends to exhibit complexity and sophistication in the solution methodology. Probably the main reason behind this is that handling the difficult, and sometimes conflicting, problem constraints is often a hard challenge for researchers. Straightforward heuristic and meta-heuristic algorithms cannot normally be applied to PD problems, without augmenting the approach with problem-specific techniques. In many cases, researchers resort to sophisticated methods in their search for good quality solutions. For example, they may hybridize several heuristics and meta-heuristics, or utilize heuristics within exact methods. Some researchers use adaptive and probabilistic search algorithms, which undergo behavioural changes dynamically throughout the search, according to current features of the search space or to recent performance of the algorithm. Dividing the search into several stages, adding a

post-optimization phase, and employing a large number of neighbourhood operators are also popular approaches to solving the PDP. Unfortunately, when complex multi-facet techniques are used, it can be very difficult to assess which algorithm component has contributed most to the success of the overall approach. Indeed, it is possible that some components may not be needed at all. In many papers only the final algorithm is presented, and experimental evidence justifying all its various components and phases is not included. Notwithstanding the run time issues, the more complex the approach, the harder it is for other researchers to replicate.

Another important issue to consider with PD problems is *solution infeasibility*. Due to the many problem constraints that often apply, obtaining a feasible solution (i.e., a solution that does not violate problem constraints) may be a challenge in itself. Since the generation of infeasible solutions cannot be easily avoided during the search, solution techniques from the literature often add a repair method to fix the infeasibility of solutions. This will inevitably make the solution algorithm less elegant and slow down the optimization process.

In our research, we investigate the potential of using heuristic and meta-heuristic approaches in solving selected important variants of PD problems. Unlike previous research in this field, we aspire to develop solution techniques that can handle this complex problem in a simple and effective way, sometimes even at the expense of a slight sacrifice in the quality of the final solution obtained. The simpler the solution technique, the more it can be integrated with other approaches, and the easier it can be incorporated in real-world optimization tools. In our opinion, to achieve this goal the solution technique must be directed towards handling the underlying constraints efficiently, without complicating the whole solution method. In addition, we aim to provide robust methodologies, to avoid the need for extensive parameter tuning. While we support, and appreciate, the need to compare the performance of our algorithms with the state-of-the-art, we try not to engage in *hyper-tuning* simply to produce marginally improved results on benchmark instances.

The heuristic and meta-heuristic approaches applied in this research focus on two main aspects that we believe can help us solve hard PD problems and achieve good quality solutions, while keeping the overall technique simple and elegant. These aspects are: **the solution representation** and the **neighbourhood moves**. The solution representation should reflect the problem components, while being simple to code and interpret. In addition, it should facilitate dealing with the problem constraints by being flexible and easily manageable, when neighbourhood moves are applied to create new solutions during the search. An appropriate solution representation is thus the initial step towards an overall successful solution technique. In addition, 'intelligent' neighbourhood moves enable the

algorithm to progress smoothly exploring promising areas of the search space, and at the same time, avoiding generating and evaluating a large number of infeasible solutions.

Specifically, our research mainly concentrates on solution representations and neighbourhood moves as portable and robust components that can be used within different heuristics and meta-heuristics to solve some hard pickup and delivery problems, and may be adapted for solving other variants of vehicle routing problems as well. Based on this perception, we were able to tailor some well-known heuristic and meta-heuristic approaches to solving the selected variants of pickup and delivery problems, i.e., the PDPTW and the 1-PDP. In this research, we employed our key ideas within some famous heuristic and meta-heuristic algorithms, such as Genetic Algorithms (GAs), Simulated Annealing (SA), Hill Climbing (HC) and Variable Neighbourhood Search (VNS), and tried to demonstrate how the proposed constraint handling mechanisms helped guide the different solution methods towards good quality solutions and manage infeasibility throughout the search, while keeping the overall algorithm as simple as possible. The proposed 'techniques', though, have the potential of being applicable within other heuristics and meta-heuristics, with no or just minor modifications, based on the particular problem to which they are being applied.

The contribution of this thesis is six fold:

1. We have developed a unique and simple solution representation for the PDPTW that enabled us to simplify the solution method and reduce the number of problem constraints that must be dealt with during the search.

2. We have devised intelligent neighbourhood moves and genetic operators that are guided by problem specific information. These operators enabled our solution algorithm for the PDPTW to create feasible solutions throughout the search. Thus, the solution approach avoids the need for a repair method to fix the infeasibility of newly generated solutions.

3. We have developed new simple routing heuristics to create individual vehicle routes for the PDPTW. These heuristics can be easily embedded in different solution construction and improvement techniques for this problem.

4. We have developed several new solution construction methods for the PDPTW, that can be easily used within other heuristic and meta-heuristic approaches to create initial problem solutions.

5. We have demonstrated how traditional neighbourhood moves from the VRP literature can be employed in a new way within a meta-heuristic for the 1-PDP, which

helped the algorithm to escape the trap of local optima and achieve high quality solutions.

6. We have applied simple adaptation of neighbourhood moves and search parameters, in both the PDPTW and the 1-PDP, for more efficient searching.

Although the techniques adopted in our research were only tried on some pickup and delivery problems, we believe that they can be easily applied to other vehicle routing and scheduling problems with only minor modifications. As will be seen through the course of this thesis, the techniques mentioned above were in most cases quite successful in achieving the objectives that we had in mind when we started our investigation of the problems in hand. Details about these techniques and how we applied them within the heuristic and meta-heuristic framework will be explained in the main part of this thesis, i.e., Chapters 5 to 10.

1.3 Thesis Overview

Chapter 1 documents the motivation behind the research carried out in this thesis. The hypothesis and contribution of the thesis are highlighted, and an overview of the structure of the thesis is presented.

Chapter 2 provides some background information about complexity theory and algorithm analysis. A quick look at some exact solution methods that can be used to solve combinatorial optimization problems to optimality is taken, before a more detailed explanation of some popular heuristic and meta-heuristic approaches is provided. The techniques introduced in this chapter include: Hill Climbing (HC), Simulated Annealing (SA), Genetic Algorithms (GAs), Ant Colony Optimization (ACO), Tabu Search (TS) and Variable Neighbourhood Search (VNS).

Chapter 3 is a survey of vehicle routing problems, with more emphasis on one particular variant that is of interest to our research, the Vehicle Routing Problem with Time Windows (VRPTW). For this problem, some solution construction and solution improvement techniques are described. In addition, a quick summary of some published meta-heuristic approaches that have been applied to this problem is provided.

Chapter 4 is another literature survey chapter dedicated to one variant of vehicle routing problems that is the focus of this research, namely pickup and delivery problems. A classification of the different problem types is given, and a brief summary of some published heuristic and meta-heuristic approaches that tackle these problems is provided.

The survey presented in this chapter is intended to be a concise overview. More details about state-of-the-art techniques are presented in later chapters, when related work to each individual problem that we handled is summarized.

Chapter 5 is the first of six chapters that detail the research carried out in this thesis. This chapter describes the first problem that we tackled, the Single Vehicle Pickup and Delivery Problem with Time Windows (SV-PDPTW). A general overview of the problem, and a summary of related work from the literature is given. In addition, a Genetic Algorithm (GA) approach to solving this problem is described in detail. A novel solution representation and intelligent neighbourhood moves that may help the algorithm overcome difficult problem constraints are tried within the GA approach. The experimental findings of this part of the research were published, as a late breaking paper, in *GECCO2007* conference [79].

Chapter 6 continues with the research started in Chapter 5 for the SV-PDPTW. Nevertheless, a more extensive investigation of the problem is given here. Several heuristic and meta-heuristic techniques are tested for solving the problem, employing the same tools introduced in Chapter 5. Three approaches are applied to solving the problem: genetic algorithms, simulated annealing and hill climbing. A comparison between the different approaches and a thorough analysis of the results is provided. The algorithms and the findings of this part of the research were published in the *Journal of Heuristics* [84].

Chapter 7 starts the investigation of the more general Multiple-Vehicle case of the Pickup and Delivery Problem with Time Windows (MV-PDPTW). After summarizing some related work in this area, new solution construction methods are developed and compared, using the single vehicle case as a sub-problem. The algorithms introduced here act as a first step towards a complete solution methodology to the problem, which will be presented in the next chapter. The different construction heuristics are compared, and conclusions are drawn about the construction algorithm that seems most appropriate for this problem. This part of the research was published in the *MIC2009* [81].

Chapter 8 continues the investigation of the MV-PDPTW, by augmenting *solution construction*, introduced in the previous chapter, with *solution improvement*. In this chapter, we tried both a GA and an SA for the improvement phase. Several techniques from the first part of our research (explained in Chapters 5 to Chapter 7) are embedded in both the GA and the SA. The GA approach is compared with two related GA techniques from the literature, and the results of both the GA and the SA are thoroughly analyzed. Part of the research carried out in this chapter was published in the *MIC2009* [80].

Chapter 9 starts the investigation of another important PD problem which is the One-commodity Pickup and Delivery Problem (1-PDP). In this chapter, we explain the pro-

blem, provide a literature review, and highlight some related problems. After that, we try to solve the 1-PDP using an Evolutionary Perturbation Scheme (EPS) that has been used for solving other variants of vehicle routing problems. Experimental results on published benchmark instances are reported and analyzed.

Chapter 10 continues the investigation of the 1-PDP, using another meta-heuristic approach, which is a Variable Neighbourhood Search (VNS) hybridized with Simulated Annealing (SA). Also, adaptation of search parameters is applied within this heuristic for more efficient searching. The approach is thoroughly explained and the experimental results on published test cases are reported. Two conference papers that discuss this approach have now been accepted for publication in *GECCO2010* (late breaking abstract) [82], and *PPSN2010* [85]. In addition, a third journal paper has been submitted to the *Journal of Heuristics* and is currently under review [83].

Chapter 11 looks beyond the current research phase and discusses how a theoretical scientific research, such as that carried out in this thesis, can be used in real-life commercial applications of vehicle routing problems. We give examples of some commercial software tools, and we also highlight important industrial aspects of vehicle routing that the research community should be aware of. An overview of future trends in scientific research tackling this issue is also provided.

Chapter 12 summarizes the research undertaken in this thesis and elaborates on the thesis contribution. Some critical analysis of parts of the current research and suggestions for future work are also presented.

1.4 A Note on Implementation and Computational Experimentation

All algorithmic implementations presented in this thesis are programmed in C++, using Microsoft Visual Studio 2005. Basic Genetic Algorithms (GAs) were implemented with the help of an MIT GA library GALIB [155]. Most of the computational experimentations were carried out using Intel Pentium (R) CPU, 3.40 GHz and 2 GB RAM, under a Windows XP operating system, unless otherwise indicated. The algorithms developed in this thesis were tested on published benchmark data for the selected problems, or on problem instances created in this research. We indicate in the relevant sections of this thesis the exact source of the data, and provide links where applicable.

Chapter 2

Combinatorial Optimization, Heuristics, and Meta-heuristics

In this chapter, we introduce an important class of problems that are of interest to researchers in Computer Science and Operations Research (OR), which is the class of Combinatorial Optimization (CO) problems. The importance of CO problems stems from the fact that many practical decision making issues concerning, for example, resources, machines and people, can be formulated under the combinatorial optimization framework. As such, popular optimization techniques that fit this framework can be applied to achieve optimal or best possible solutions to these problems, which should minimize cost, increase profit and enable a better usage of resources.

Our research handles one class of CO problems, which is concerned with vehicle routing and scheduling. Details about this specific class will be presented in Chapters 3 and 4. In the current chapter we provide a brief description of CO problems, and the related algorithm and complexity analysis theory in Section 2.1. We will then proceed to review important techniques that are usually applied to solving CO problems. Section 2.2 briefly highlights some *exact algorithms* that can be used to find optimal solutions to these problems. On the other hand, *heuristic methods* that may be applied to find a good solution, which is not necessarily the optimum, are introduced in Section 2.3. We emphasize in this section: Hill Climbing (HC), Simulated Annealing (SA), Genetic Algorithms (GAs), Ant Colony Optimization (ACO), Tabu Search (TS), and Variable Neighbourhood Search (VNS). Finally Section 2.4 concludes this chapter with a brief summary.

2.1 Combinatorial Optimization, Algorithm and Complexity Analysis

Combinatorial optimization problems can generally be defined as problems that require searching for the best solution among a large number of finite discrete candidate solutions.

More specifically, it aspires to find the best allocation of limited resources that can be used to achieve an underlying objective. Certain constraints are usually imposed on the basic resources, which limit the number of feasible alternatives that can be considered as problem solutions. Yet, there is still a large number of possible alternatives that must be searched to find the best solution.

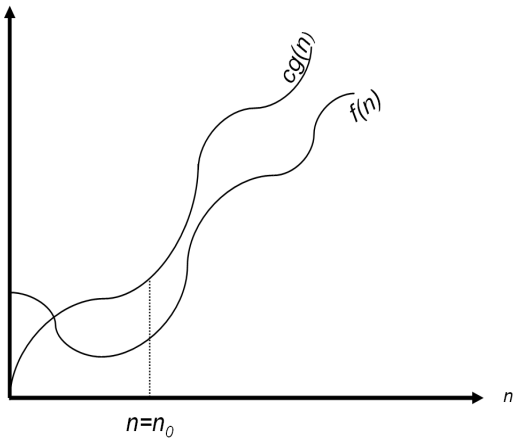
There are numerous applications of CO problems in areas such as crew scheduling, jobs to machines allocation, vehicle routing and scheduling, circuit design and wiring, solid-waste management, energy resource planning, just to name a few. Optimizing solutions to such applications is vital for the effective operation and perhaps the survival of the institution that manages these resources. To a large extent, finding the best or optimum solution is an integral factor in reducing costs, and at the same time maximizing profit and clients' satisfaction.

Many practical CO problems can be described using well-known mathematical models, for example the Knapsack problem, job-shop scheduling, graph coloring, the TSP, the boolean satisfiability problem (SAT), set covering, maximum clique, timetabling... etc. Many algorithms and solution methods exist for solving these problems. Some of them are exact methods that are guaranteed to find optimum solutions given sufficient time, and others are approximation techniques, usually called heuristics or meta-heuristics, which will give a good problem solution in a reasonable amount of time, with no guarantee to achieve optimality.

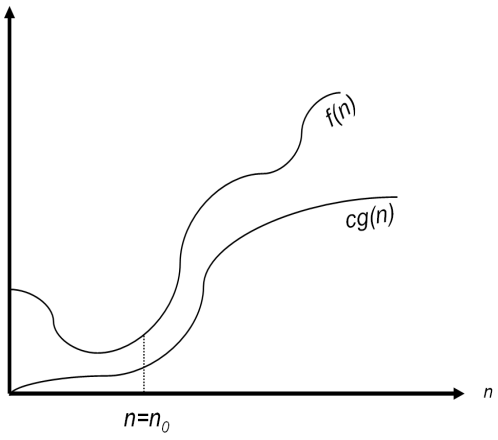
Within this domain, researchers are often faced with the requirement to compare algorithms in terms of their efficiency, speed, and resource consumption. The field of **algorithm analysis** helps scientists to perform this task by providing an estimate of the number of operations performed by the algorithm, irrespective of the particular implementation or input used. Algorithm analysis is usually preferred to comparing actual running times of algorithms, since it provides a standard measure of algorithm complexity irrespective of the underlying computational platform and the different types of problem instances solved by the algorithm. Within this context, we usually study the *asymptotic efficiency* of an algorithm, i.e., how the running time of an algorithm increases as the size of the input approaches infinity.

In algorithm analysis, the O notation is often used to provide an asymptotic *upper bound* of the complexity of an algorithm. We say that an algorithm is of $O(n)$ (Order n), where n is the size of the problem, if the total number of steps carried out by the algorithm is at most a constant times n. More specifically, $f(n) = O(g(n))$, if there exist positive constants c and n_0 such that $0 \leq f(n) \leq cg(n)$, for all $n \geq n_0$. The O notation is usually used to describe the complexity of the algorithm in a worst-case scenario. Other less

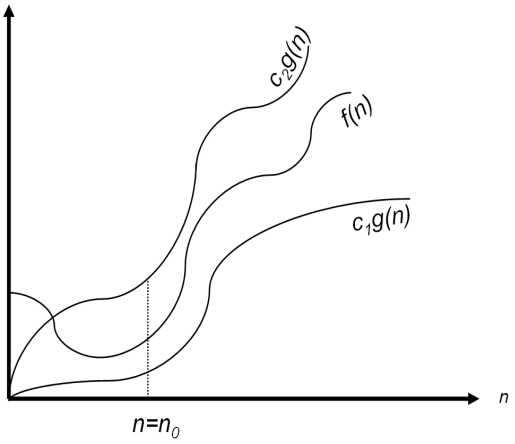
frequently used notations for algorithm analysis are the Ω notation and the Θ notation. The Ω notation provides an asymptotic *lower bound*, i.e., $f(n)$ is of $\Omega(g(n))$ if there exist positive constants c and n_0 such that $0 \leq cg(n) \leq f(n)$, for all $n \geq n_0$. On the other hand, the Θ notation provides an asymptotically *tight bound* for $f(n)$, i.e., $f(n)$ is of $\Theta(g(n))$ if there exist positive constants c_1, c_2 and n_0 such that $0 \leq c_1 g(n) \leq f(n) \leq c_2 g(n)$, for all $n \geq n_0$ [32]. Figures 2.1(a), 2.1(b) and 2.1(c) demonstrate a graphic example of the *O* notation, the Ω notation, and the Θ notation respectively.[1]



(a) The *O* notation



(b) The Ω notation



(c) The Θ notation

Figure 2.1: Asymptotic Notations.

In addition to analyzing the efficiency of an algorithm, we sometimes need to know what types of algorithms exist for solving a particular problem. The field of **complexity analysis** analyzes problems rather than algorithms. Two important classes of problems are usually identified in this context. The first class is called $\mathcal{P}$ **(polynomial time problems)**.

[1] In this thesis we use the O notation for the analysis of algorithms when needed, since it is the most commonly used type of asymptotic notations among the three notations introduced in this section.

It contains problems that can be solved using algorithms with running times such as $O(n)$, $O(log(n))$ and $O(n^k)$ [2]. They are relatively easy problems, sometimes called ***tractable*** problems. Another important class is called $\mathcal{NP}$ **(non-deterministic polynomial time problems)**. This class includes problems for which there exists an algorithm that can *guess* a solution and *verify* whether the guessed solution is correct or not in polynomial time. If we have an unbounded number of processors that each can be used to guess and verify a solution to this problem in parallel, the problem can be solved in polynomial time. One of the big open questions in computer science is whether the class $\mathcal{P}$ is equivalent to the class $\mathcal{NP}$. Most scientists believe that they are not equivalent. This, however, has never been proven.

Researchers also distinguish a sub class of $\mathcal{NP}$, called the $\mathcal{NP}$*-complete* class. In a sense, this class includes the hardest problems in computer science, and is characterized by the fact that either all problems that are $\mathcal{NP}$*-complete* are in $\mathcal{P}$, or not in $\mathcal{P}$. Many $\mathcal{NP}$*-complete* problems may require finding a subset within a base set (e.g. the SAT problem), or an arrangement (or permutation) of discrete objects (e.g. the TSP), or partitioning an underlying base set (e.g. graph coloring). As mentioned above, these problems belong to the combinatorial optimization class of problems, and sometimes called ***intractable*** problems.

An optimization problem for which the associated decision problem is $\mathcal{NP}$*-complete* is called an $\mathcal{NP}$*-hard* problem. A **decision problem** is a problem that has only two possible answers: *yes* or *no*. For example, if the problem is a cost minimization problem, such that it is required to find a solution with the minimum possible cost, the associated decision problem would be formulated as: " is there a solution to the problem whose cost is B, where B is a positive real number?". Figure 2.2 demonstrates how the different complexity classes may relate to each other. For more details about algorithm analysis and complexity theory, the reader is referred to the book by Garey and Johnson [54] and Cormen *et al.* [32].

Solving CO problems has been a challenge for many researchers in computer science and operations research. Exact methods used to solve regular problems cannot be used to solve CO problems given current resources, since searching among all possible solutions of a certain intractable problem is usually prohibitive for large problem sizes. The natural alternative would be to use approximation methods that give good, rather than optimal, solutions to the problem in a reasonable amount of time.

In the next section we briefly consider some exact methods that can be used to solve

[2] Running times of $O(n)$ are usually called linear running times. Also, running time of $O(n^k)$ are often referred to as polynomial running times, while running times of $O(2^n)$ are called exponential running times.

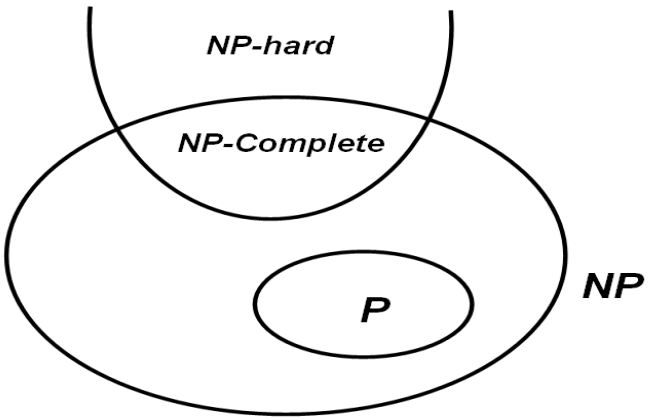


Figure 2.2: Complexity Classes.

CO problems. However, since exact algorithms are not practical solution methods for the applications considered in this thesis, our discussion of these techniques will be very brief. On the other hand, a more detailed investigation of approximation algorithms, i.e., heuristics and meta-heuristics, will be presented in Section 2.3.

2.2 Exact Algorithms

As previously mentioned, exact algorithms are guaranteed to find an optimum solution for a CO problem if one exists, given a sufficient amount of time. In general, the most successful of these algorithms try to reduce the solution space and the number of different alternatives that need to be examined in order to reach the optimum solution. Some exact algorithms that have been applied to solving CO problems are the following:

1. **Dynamic Programming (DP):** which refers to the process of simplifying a complicated problem by dividing it into smaller subproblems in a recursive manner. Problems that are solved using dynamic programming usually have the potential of being divided into stages with a decision required at each stage. Each stage also has a number of states associated with it. The decision at one stage transforms the current state into a state in the next stage. The decision to move to the next state is only dependent on the current state, not the previous states or decisions. *Top-down* dynamic programming is based on storing the results of certain calculations, in order to be used later. *Bottom-up* dynamic programming recursively transforms a

complex calculation into a series of simpler calculations. Only certain CO problems are amenable to DP.

2. **Branch-and-Bound (B&B):** is an optimization technique which is based on a systematic search of all possible solutions, while discarding a large number of non-promising candidate solutions, using a depth-first strategy. The decision to discard a certain solution is based on estimating upper and lower bounds of the quantity to be optimized, such that nodes whose objective function are lower/higher than the current best are not explored. The algorithm requires a 'branching' tool that can split a given set of candidates into two smaller sets, and a 'bounding' tool that computes upper and lower bounds for the function to be optimized within a given subset. The process of discarding fruitless solutions is usually called 'pruning'. The algorithm stops when all nodes of the search tree are either pruned or solved.

3. **Branch-and-Cut (B&C):** is a B&B technique with an additional cutting step. The idea is to try to reduce the search space of feasible candidates by adding new constraints (cuts). Adding the cutting step may improve the value returned in the bounding step, and could allow solving subproblems without branching.

2.3 Heuristic Algorithms

Given the limitation of exact methods in solving large CO problems, approximation techniques are often preferred in many practical situations. Approximation algorithms, like heuristics and meta-heuristics[3], are techniques that solve 'hard' CO problems in a reasonable amount of computation time, compared to exact algorithms. However, there is no guarantee that the solution obtained is an optimal solution, or that the same solution quality will be obtained every time the algorithm is run.

Heuristic algorithms are usually experience-based, with no specific pre-defined rules to apply. Simply, they are a 'common-sense' approach to problem solving. On the other hand, an important subclass of heuristics are meta-heuristic algorithms, which are general-purpose frameworks that can be applied to a wide range of CO problems, with just minor changes to the basic algorithm definition. Many meta-heuristic techniques try to mimic biological, physical or natural phenomena drawn from the real-world.

[3]In this discussion we use the term approximation/heuristic algorithm to refer to any 'non-exact' solution method. More accurately, however, the term *approximation algorithm* is often used to refer to an optimization algorithm which provides a solution that is guaranteed to be within a certain distance from the optimum solution every time it is run, with provable runtime bounds [153]. This may not be necessarily the case for *heuristic* algorithms, though.

In solving a CO problem using a heuristic or a meta-heuristic algorithm, the search for a good problem solution is usually divided into two phases: *solution construction* and *solution improvement*. **Solution construction** refers to the process of creating one or more initial feasible solutions that will act as a starting point from which the search progresses. In this phase, a construction heuristic usually starts from an empty solution and gradually adds solution components to the partially constructed solution until a feasible solution is generated. On the other hand, **solution improvement** tries to gradually modify the starting solution(s), based on some predefined metric, until a certain solution quality is obtained or a given amount of time has passed.

Within the context of searching for a good quality solution, we usually use the term **search space** to refer to the *state space* of all feasible solutions/states that are reachable from the current solution/state. To find a good quality solution, a heuristic or a meta-heuristic algorithm moves from one state to another, i.e., from one candidate solution to another, through a process that is often called **Local Search (LS)**. During LS, the new solution is usually generated within the *neighbourhood* of the previous solution. A **neighbourhood** $N(x)$ of a solution x is a subset of the search space that contains one or more **local optima**, the best solutions in this neighbourhood. The transition process from one candidate solution to another within its neighbourhood requires a **neighbourhood move** that changes some attributes of the current solution to transform it to a new solution x'. A cost function $f(x)$ is used to evaluate each candidate solution x and determine its cost compared to other solutions in its neighbourhood. The best solution within the overall search space is called the **globally optimal** solution, or simply the **optimum solution**. Figure 2.3 shows a search space for a minimization function, i.e., our goal is to obtain the global minimum for this function. In this figure three points (solutions) are local optima within their neighbourhoods, ***A***, ***B*** and ***C***. The global optimum among the three points is point ***C***.

Several modifications to the basic local search algorithm have been suggested to solve CO problems. For example, Baum [10] suggested an Iterated Local Search (ILS) procedure for the TSP, in which a local search is applied to the neighbouring solution x' to obtain another solution x''. An acceptance criterion is then applied to possibly replace x' by x''. Voudouris and Tsang [154] suggest a Guided Local Search (GLS) procedure, in which penalties are added to the objective function based on the search experience. More specifically, the search is driven away from previously visited local optima, by penalizing certain solution features that it considers should not occur in a near-optimal solution.

In addition, some local search methods have been formulated under the meta-heuristic framework, in which a set of predefined rules or algorithmic techniques help guide a heuristic

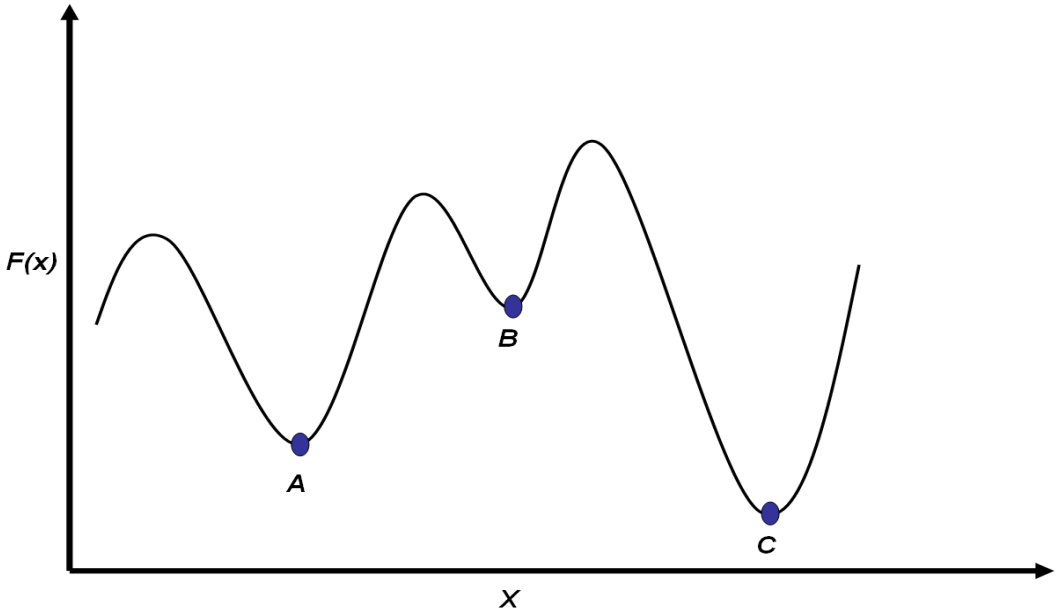


Figure 2.3: Search space for a minimization problem: local and global optima.

search towards reaching high quality solutions to complex optimization problems. Among the most famous meta-heuristic techniques are: Hill Climbing (HC), Simulated Annealing (SA), Genetic Algorithms (GAs), Ant Colony Optimization (ACO), Tabu Search and Variable Neighbourhood Search (VNS). For more details about local search techniques and its different variations and applications to CO problems, the reader is referred to [3] and [125]. In what follows we describe some important meta-heuristic techniques that have been widely used for solving CO problems.

2.3.1 Hill Climbing (HC)

Hill Climbing is the simplest form of local search, where the new neighbouring solution always replaces the current solution if it is better in quality. The process can thus be visualized as a step-by-step movement towards a locally optimum solution. Figure 2.4 shows how an HC algorithm transitions 'downhill' from an initial solution to a local minimum. On the other hand, Algorithm 2.1 shows the steps of an HC procedure applied to a minimization problem.

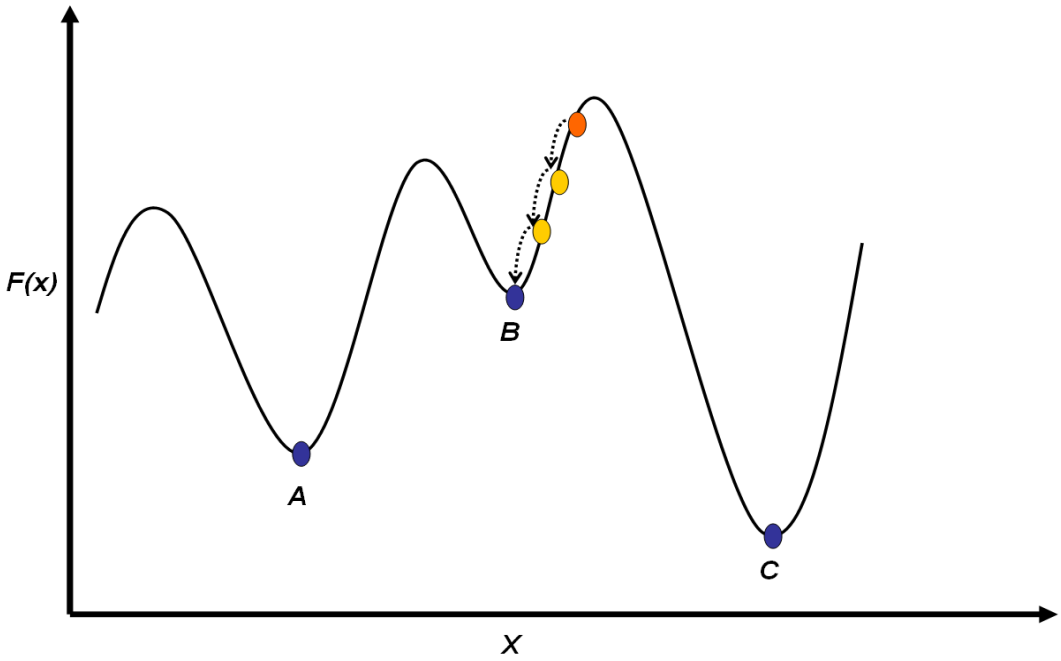


Figure 2.4: Hill Climbing: downhill transition from an initial solution to a local optimum.

Algorithm 2.1: Hill Climbing (HC).

1: Generate an initial solution x
2: **repeat**
3: Generate a new solution x' within the neighbourhood of x ($x' \in N(x)$)
4: $\Delta \leftarrow f(x') - f(x)$
5: **if** ($\Delta < 0$) **then**
6: $x \leftarrow x'$
7: **until** (Done){stopping condition is reached}
8: Return solution x

HC is frequently applied within other meta-heuristic techniques to improve solution quality at various stages of the search. In our research we tried HC among the selected approaches for solving the Single Vehicle Pickup and Delivery Problem with Time Windows (SV-PDPTW), as will be explained in Chapter 6. HC was also an important part of the solution construction heuristics that we developed for the Multiple Vehicle Pickup and Delivery Problem with Time Windows (MV-PDPTW), as will be explained in Chapter 7. We have also applied it in parts of the algorithms developed for solving the One-Commodity Pickup and Delivery Problem (1-PDP), as will be shown in Chapters 9 and 10.

2.3.2 Simulated Annealing (SA)

Simulated annealing is a well-known meta-heuristic search method that has been used successfully in solving many combinatorial optimization problems. It is a hill climbing algorithm with the added ability to escape from local optima in the search space. However, although it yields excellent solutions, it is very slow compared to a simple hill climbing procedure.

The term simulated annealing is adopted from the annealing of solids, where we try to minimize the energy of the system using slow cooling until the atoms reach a stable state. The slow cooling technique allows atoms of the metal to line themselves up and to form a regular crystalline structure that has high density and low energy. The initial temperature and the rate at which the temperature is reduced is called the **annealing schedule**.

The theoretical foundation of SA was led by Kirkpatrick *et al.* in 1983 [93], where they applied the *Metropolis algorithm* [107] from statistical mechanics to CO problems. The Metropolis algorithm in statistical mechanics provides a generalization of iterative improvement, where controlled uphill moves (moves that do not lower the energy of the system) are probabilistically accepted in the search for obtaining a better organization and escaping local optima. In each step of the Metropolis algorithm, an atom is given a small random displacement. If the displacement results in a decrease in the system energy, the displacement is accepted and used as a starting point for the next step. If on the other hand the energy of the system is not lowered, the new displacement is accepted with a certain probability $\exp^{(-E/k_bT)}$ where E is the change in energy resulting from the displacement, T is the current temperature, and k_b is a constant called a *Boltzmann constant* . Depending on the value returned by this probability either the new displacement is accepted or the old state is retained. For any given T, a sufficient number of iterations always leads to thermal equilibrium. The SA algorithm has also been shown to possess a formal proof of convergence using the theory of *Markov Chains* [47].

In solving a CO problem using SA, we start with a certain feasible solution to the problem. We then try to optimize this solution using a method analogous to the annealing of solids. A neighbour of this solution is generated using an appropriate method, and the cost (or the fitness) of the new solution is calculated. If the new solution is better than the current solution in terms of reducing cost (or increasing fitness), the new solution is accepted. If the new solution is not better than the current solution, though, the new solution is accepted with a certain probability. The probability of acceptance is usually set to $\exp^{(-\Delta/T)}$, where Δ is the change in cost between the old and the new solution and T is the current temperature. The probability thus decreases exponentially with the badness of

the move.

The SA procedure is less likely to get stuck in a local optimum, compared to a simple HC, since bad moves still have a chance of being accepted. The annealing temperature is first chosen to be high so that the probability of acceptance will also be high, and almost all new solutions are accepted. The temperature is then gradually reduced so that the probability of acceptance of low quality solutions will be very small and the algorithm works more or less like hill climbing, i.e., high temperatures allow a better exploration of the search space, while lower temperatures allow a fine tuning of a good solution. The process is repeated until the temperature approaches zero or no further improvement can be achieved. This is analogous to the atoms of the solid reaching a crystallized state. Figure 2.5 shows how occasional uphill moves may allow the SA algorithm to escape local optima in a minimization problem, compared to the HC algorithm shown in Figure 2.4.

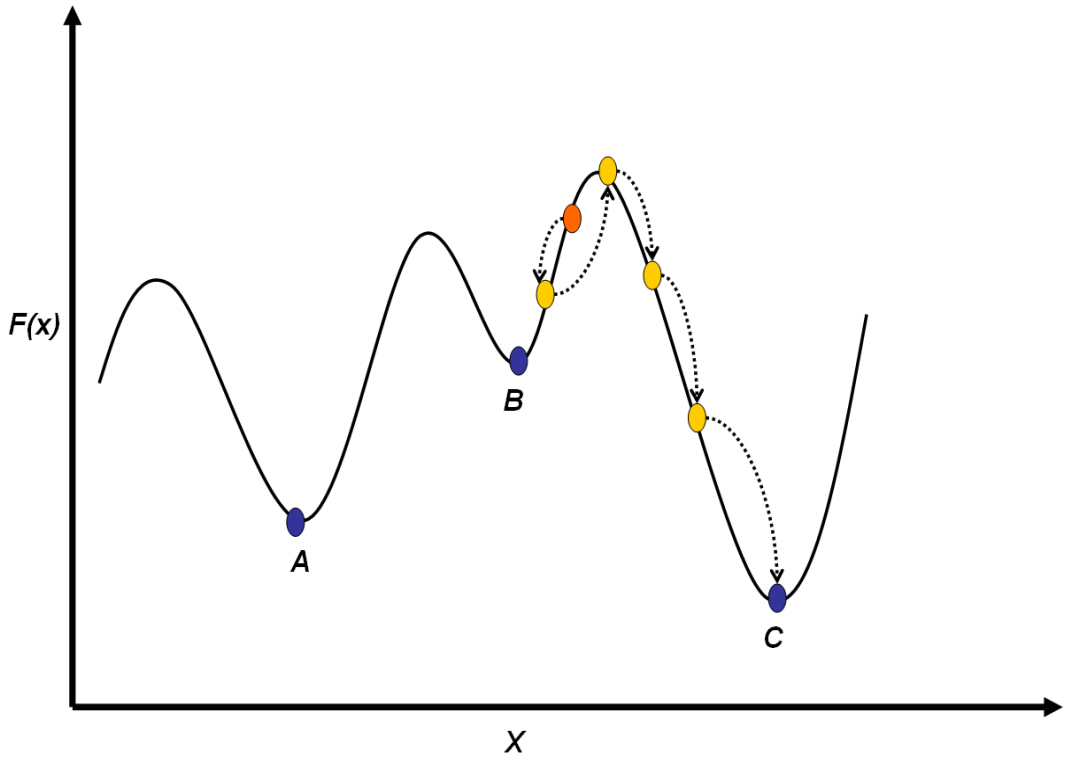


Figure 2.5: Simulated Annealing: occasional uphill moves and escaping local optima.

Applying SA to CO problems also requires choices concerning both general SA parameters and problem specific decisions. The choice of **SA parameters** is critical to the performance of the algorithm. These parameters are: the value of the initial temperature T, a temperature function that determines how the temperature will change with time, the

number of iterations $N(t)$ to be executed at each temperature, and a stopping criterion to terminate the algorithm.

As previously mentioned, the initial temperature $T(0)$ is generally chosen high enough so that almost any move is accepted regardless of its fitness. This choice is adopted from the physical analogy and corresponds to heating up a substance until all particles are randomly arranged in a liquid. The temperature update function is usually a proportional temperature function $T(t+1) = \alpha T(t)$, where t is the current iteration. Typical values of α lie between 0.8 and 0.99. Using such values provides very small decrements of the temperature, which corresponds to slow cooling of the substance until the temperature approaches zero.

The number of iterations carried out at each temperature value should be large enough to bring the system to a stable state equating to thermal equilibrium in the physical analogy. Some applications may choose $N(t)$ to be constant for each temperature. The stopping criterion of the algorithm is usually the stagnation of the system when no change in the result can be obtained for a specified number of iterations or temperature changes.

Implementing SA also requires a set of **problem-specific decisions**. These include: identifying the set of feasible solutions to the problem, defining a clear objective function, generating an initial solution, and defining a neighbourhood operator that generates moves using the current solution.

The topology of the neighbourhood structure is also critical to the performance of the SA algorithm. In general, a smooth topology with shallow local optima is favoured over a bumpy topology with many deep local minima. A neighbourhood function is also easy to implement for discrete problems, while implementing a neighbourhood function for continuous problems is more challenging. Constrained problems also raise some difficulties. A choice must be made between restricting the solution space to solutions that conform to the constraints, or allowing solutions that break the constraints at the expense of a suitably defined penalty function. The generic SA algorithm is described in Algorithm 2.2, where the underlying optimization problem is again assumed to be a minimization function.

Some modifications to the basic SA algorithm have been suggested. These modifications are intended to provide an improvement of the quality of the solution and/or processing time. One attempt is to store the best solution found so far. Since the SA algorithm accepts solutions probabilistically, it may accept solutions that are worse than the current solution. A good solution found during the run may be discarded because it was not lucky during the acceptance attempt. Storing the best solution found so far prevents the SA algorithm from returning a final solution that is worse than the best solution ever found (e.g. [63]).

Algorithm 2.2: The Simulated Annealing (SA) Algorithm.

1: Generate an initial solution x
2: $T \leftarrow T_0$ {initialize current temperature}
3: **repeat**
4: **for** $(i = 0; i < numIterations; i++)$ **do**
5: Generate a new solution x' within the neighbourhood of x ($x' \in N(x)$)
6: $\Delta \leftarrow f(x') - f(x)$
7: **if** $(\Delta < 0)$ **then**
8: $x \leftarrow x'$
9: **else**
10: $p = Random(0, 1)$ {generate a random number in the interval (0,1)}
11: **if** $(p < \exp^{(-\Delta/T)})$ **then**
12: $x \leftarrow x'$
13: $T \leftarrow \alpha \times T$ {reduce current temperature}
14: **until** (Done){stopping condition is reached}
15: Return solution x

Some researches also modify the basic SA algorithm using problem specific information. For example [148] suggest a neighbourhood operator that is able to identify promising areas in the neighbourhood, and give a greater probability to generated moves that fall in the promising areas. SA has also been hybridized with other optimization techniques. This can be done by using another technique to generate a good initial solution that SA can improve, or using SA to generate a good solution that can be used by another search technique.

To sum up, SA has several attractive features, especially in difficult optimization problems in which obtaining a good solution, with a reasonable computational effort and processing time, is preferred to an optimal solution with considerably higher cost. The basic advantages of SA are the following:

1. It is very easy to implement, since it just requires a method for generating a move in the neighbourhood of the current solution, and an appropriate annealing schedule.

2. It can be applied to a wide range of problem types. For example, any combinatorial optimization problem can be tackled using SA, if an appropriate neighbourhood structure has been devised.

3. High quality solutions can be obtained using SA, if a good neighbourhood structure and a good annealing schedule have been chosen.

These benefits, however, may be at the expense of a longer processing time, compared to simple hill climbing. In our research we applied SA to solving the SV-PDPTW, as explained in Chapter 6, and for the MV-PDPTTW, as explained in Chapter 8. We also hybridized SA with a Variable Neighbourhood Search (VNS) approach for solving 1-PDP, as will be shown in detail in Chapter 10.

2.3.3 Genetic Algorithms (GAs)

The idea of simulation of biological evolution and the natural selection of organisms dates back to the 1950's. One of the early pioneers in this area was Alex Fraser with his research published in 1957 [50] [51]. Nevertheless, the theoretical foundation of GAs were established by John Holland in 1975 [78], after which GAs became popular as an intelligent optimization technique that may be adopted for solving many difficult problems.

The theme of a GA is to simulate the processes of biological evolution, natural selection and survival of the fittest in living organisms. In nature, individuals compete for the resources of the environment, and they also compete in selecting mates for reproduction. Individuals who are better or fitter in terms of their genetic traits survive to breed and produce offspring. Their offspring carry their parents' basic genetic material, which leads to their survival and breeding. Over many generations, this favourable genetic material propagates to an increasing number of individuals. The combination of good characteristics from different ancestors can sometimes produce 'super fit' offspring who out-perform their parents. In this way, species evolve to become better suited to their environment.

GAs operate in exactly the same manner. They work on a **population** of individuals representing possible solutions to a given problem. In traditional GAs, each individual is usually represented by a string of bits analogous to **chromosomes** and **genes**, i.e., the parameters of the problem are the genes that are joined together in a solution chromosome. A **fitness** value is assigned to each individual in order to judge its ability to survive and breed. The highly fit individuals are given a chance to breed by being selected for reproduction. Thus, the selection process usually favours the more fit individuals. Good individuals may be selected several times in one iteration, while poor ones may not be selected at all. By selecting the 'most fit' individuals, favourable characteristics spread throughout the population over several generations, and the most promising areas of the search space are explored. Finally, the population should converge to an optimal or near optimal solution. Convergence means that the population evolves toward increasing uniformity, and the average fitness of the population will be very close to the highest fitness.

During the reproduction phase of a GA, two individuals breed by combining their genes in

an operation called **crossover**. Not all selected pairs undergo crossover. A random choice is applied, where the likelihood of crossover is some given probability. If crossover is not performed, offspring are produced simply by duplicating their parents. Crossover allows the basic genetic material of parents to pass to their children, who then form the next generation.

Another operation that is performed by GAs is **mutation**. Mutation is applied to each child generated from crossover. With a certain small probability, each gene may be altered. Thus, Crossover allows a rapid exploration of the search space by producing large jumps, while mutation allows a small amount of random search. The basic outline of a GA is shown in the following algorithm:

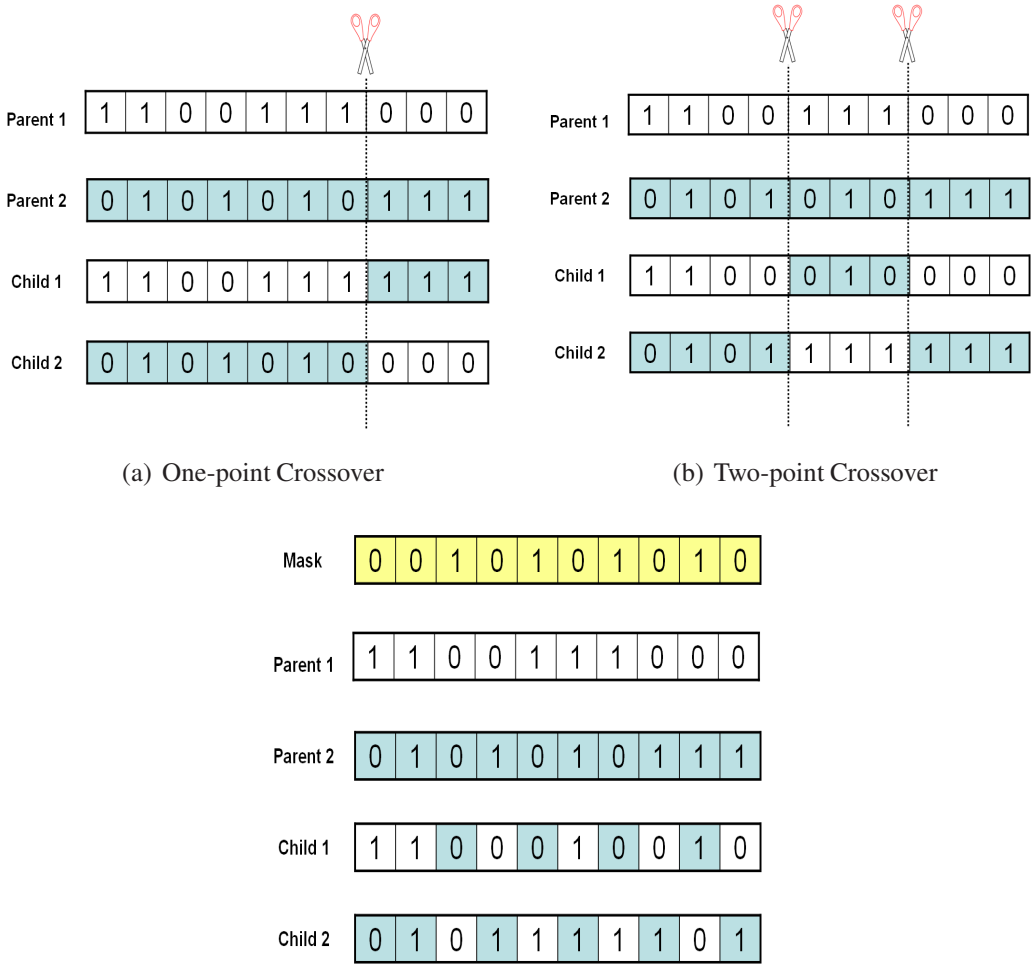
Algorithm 2.3: The Basic Genetic Algorithm (GA).

1: ***Coding:*** initialize and encode a random population of solutions called chromosomes
2: **repeat**
3: ***Fitness Assignment:*** decode and evaluate the fitness of each chromosome
4: ***Selection:*** select some chromosomes from the current population for reproduction, where the selection criterion is based on the fitness of the selected parents
5: ***Recombination:*** with some probability apply crossover between the selected parents to produce new children
6: ***Variation:*** apply mutation with a small probability to some genes of the newly produced offspring, or to selected members of the population
7: ***Replacement:*** integrate the newly generated offspring with the old population to create a new generation
8: **until** (a certain stopping condition is reached)

The traditional GA uses strings of bits to represent chromosomes. In addition, the classical crossover operator is called **one-point crossover**, where the two mating chromosomes are each cut once at corresponding points and the sections after the cuts are exchanged. However, many different crossover types have been devised, often involving more than one cut point. For example, in a **two-point crossover** two cut points are chosen randomly in the parent chromosomes. The section between the selected cut points is exchanged between the two children. Another form of crossover is called **uniform crossover**, in which a random mask of bits is generated. Each gene in the offspring is created by copying the corresponding gene from one of the parents. The parent is selected according to the value of the corresponding bit in the mask. These three crossover operators are illustrated in Figure 2.6.

On the other hand, the most popular mutation operators are **point mutation**, where one

bit in the chromosome is flipped, and **inversion mutation**, in which a substring is selected at random and the bits within this substring are inverted. Figure 2.7 demonstrates these two mutation operators.

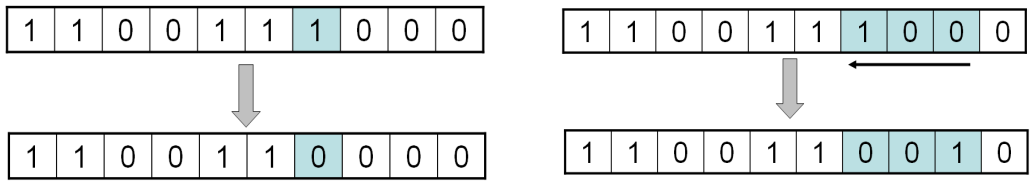


(a) One-point Crossover

(b) Two-point Crossover

(c) Uniform Crossover

Figure 2.6: Crossover.



(a) Point Mutation

(b) Inversion Mutation

Figure 2.7: Mutation.

Chromosome representation as a bit string is not suitable for many problems types, though. For example, a permutation of integers representing nodes or locations has been often used for the TSP and other vehicle routing problems. This kind of representation is called an *order-based* representation, since the fitness depends on the order in which the genes appear. Different crossover and mutation operators must be used in this case, since classical genetic operators, such as the ones described above, will result in 'infeasible' chromosomes. Goldberg in [64] described several order-based operators, such as the **Partially Matched Crossover (PMX)**. In a PMX, it is not the genes that are crossed, but the order in which they appear, i.e., offspring have genes that inherit order information from each parent. This avoids the problem of generating offspring that violate problem constraints, such as having duplicate cities in a chromosome that represents a solution to the TSP, or having a chromosome with some non-visited cities. Figure 2.8 shows an example of a PMX, where cities 5,6,7 exchange their positions with 2, 3 and 10 respectively. Other crossover operators that can be used for order-based representations are *cycle crossover* and *order crossover* (see [64] for details).

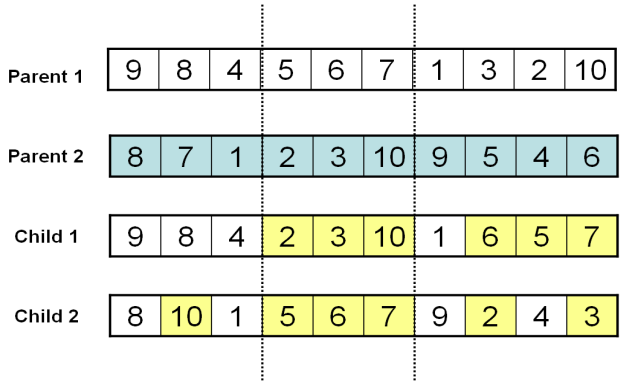


Figure 2.8: Partially Matched Crossover (PMX).

To select two individuals in the population for mating, several **selection** methods may be applied. For example, in a **rank selection**, individuals are sorted according to their objective function values, and each individual is assigned a number of offspring that is a function of its rank in the population. **Roulette wheel selection** picks an individual, with a probability, based on the magnitude of the fitness score relative to the rest of the population. The higher the score, the more likely an individual will be selected, and the probability of the individual being chosen is equal to the fitness of the individual divided by the sum of the fitness values of all individuals in the population. On the other hand, a **tournament selection** uses the roulette wheel, or another selection method, to select two

or more individuals, then picks the one with the higher score.

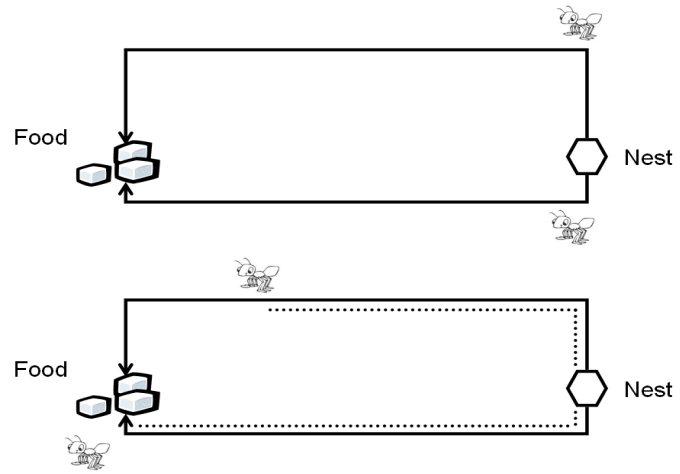
Similarly, many **replacement** schemes can be used to integrate the new offspring and produce the next generation. In a **Simple GA**, the new generation of offspring completely replace the old population, while a **Steady State GA** allows generations to overlap, with a certain percentage of replacement. On the other hand, an **Incremental GA** only allows a small degree of overlap between generations, for example by replacing one or two individuals at each generation.

In summary, GAs represent an intelligent search method, since they operate on a population of solutions and allocate trials to promising areas of the search space. GAs do not depend heavily on information available from the underlying problem. They are easy to connect to existing simulations and models, and can be easily hybridized to generate knowledge-augmented GAs. Using the operations of selection of the fittest, mutation, and crossover, GAs can quickly reach fit individuals (not always the most fit), but who are usually good enough as solutions to problems of a large magnitude. Crossover is considered as the main GA operator. Having to combine two solutions rather than one, makes designing an appropriate crossover operator often more challenging than developing a mutation operator or a simple neighbourhood move. This usually makes GAs implementation more difficult compared to SA or simple HC.

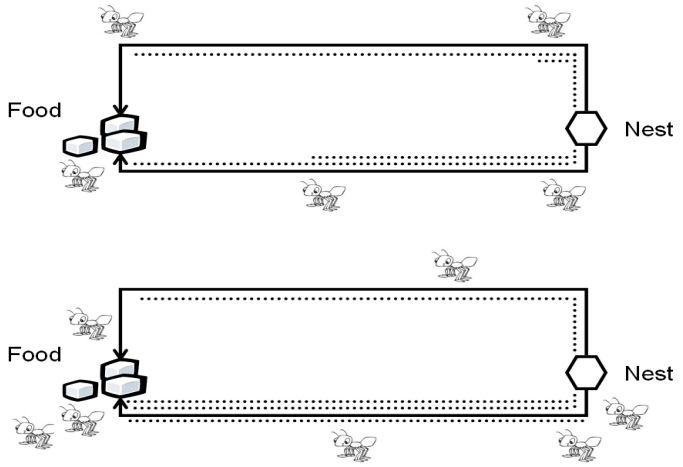
In our research we applied GAs to solving the PDPTW, for both the single and multiple vehicle variants, as will be explained in detail in Chapters 5, 6 and 8. We also applied a GA within a perturbation scheme for the 1-PDP, as will be detailed in Chapter 9.

2.3.4 Ant Colony Optimization (ACO)

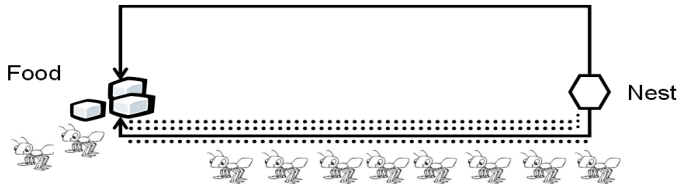
Ant Colony Optimization is a meta-heuristic technique that is inspired by the behaviour of real ants. Its principles were established by Dorigo *et al.* in 1991 [42]. Real ants cooperate to find food resources by laying a trail of a chemical substance called ‘pheromone’ along the path from the nest to the food source. Depending on the amount of pheromone available on a path, new ants are encouraged, with a high probability, to follow the same path, resulting in even more pheromone being placed on this path. Shorter routes to food sources have higher amounts of pheromone. Thus, over time, the majority of ants are directed to use the shortest path. This type of indirect communication is called ‘stigmergy’ [41], in which the concept of *positive feedback* is exploited to find the best possible path, based on the experience of previous ants. Figure 2.9 visualizes the foraging behaviour of real ants in their search for food sources.



(a) Two ants start with equal probability of going on either path. The ant on the shorter path arrives more quickly to the food source



(b) The density of the pheromone on the shorter path increases more rapidly than the pheromone on the longer path. Many ants begin using the path with higher pheromone



(c) Over time, the shorter path is almost exclusively used, and the pheromone on the longer path evaporates

Figure 2.9: Foraging behaviour of ants.

Applying ACO to hard combinatorial optimization problems is attractive, due the flexibility and robustness of the algorithm. In fact, the technique can be applied to solving a wide range of problems with only minimal changes to the basic algorithm. In addition, its population based nature allows exploring the search space, while exploiting positive feedback during the search process to find near optimal solutions.

In solving a combinatorial optimization problem using an ACO, the problem may be visualized as finding the shortest path in a weighted graph. Artificial ants (software agents) cooperate to find the best path by constructing solutions consisting of graph nodes step-by-step. Each step adds a selected node to the partial solution in a stochastic manner, but biased by the amount of pheromone available on the nodes or the edges connecting them. To this end, problem-specific information is placed in a common memory, which plays the role of the pheromone trail. The information in the memory, i.e., the pheromone trail, is updated by ants at each iteration, allowing ants to cooperate in finding good problem solutions. Pheromone values also diminish over time, similar to the evaporation of real pheromone. Thus, solutions of bad quality are eliminated from further consideration as the search progresses. Algorithm 2.4 shows the main steps of the ACO meta-heuristic.

Algorithm 2.4: The Ant Colony Optimization (ACO) Algorithm.

```
1: Set parameters, initialize pheromone trails
2: while (termination condition not met) do
3:    ConstructAntSolutions
4:    DaemonActions {optional}
5:    UpdatePheromones
```

The steps of Algorithm 2.4 are briefly explained below:

1. ***ConstructAntSolutions***: during this phase, each ant constructs a solution from the set of feasible problem solutions, by adding one solution component at each step of the construction process to the current partial solution. For example, in solving a TSP, the construction process will add one non-visited city at a time to the current solution. The choice of the new solution component depends on a *probabilistic rule* that takes into account both the pheromone trail of the component at the current iteration and other problem-specific heuristic information. In a TSP, the pheromone trail is associated with the edge connecting the last added city and the potential new city, such that edges previously used by other ants are favoured, due to an increased pheromone value. On the other hand, the heuristic information (sometimes called the *attractiveness*) will be proportional to the length of the edge connecting the two

cities. Certain parameters are also used in the probabilistic rule to determine the relative importance of the pheromone trail and the heuristic information. During the solution construction, individual ants may update the pheromone trail on the edges they visited in a process called *local pheromone update*.

2. ***DaemonActions***: in this optional step of the ACO algorithm, some problem-specific action may be required, which cannot be usually performed by a single ant. For example, local search may be applied to optimize the set of generated solutions. The new optimized solutions are then used to decide which pheromone trails to update.

3. ***UpdatePheromones***: this is a *global pheromone update* process that is performed at the end of each iteration, where updating the pheromone values depends on the quality of the generated solutions in the current iteration. This is usually done by decreasing the pheromone value for all solutions, in a process called *evaporation*, and increasing the pheromone value of good solutions. Evaporation is a tool that ACO uses to explore new areas of the search space and avoid being trapped in local optima.

ACO has become very popular in solving CO problems. For example, its versatility and robustness have been demonstrated in [43] by tailoring the approach to the TSP, the asymmetric TSP, the Quadratic Assignment Problem (QAP), and the job-shop scheduling. It has also been applied to variants of vehicle routing problems (e.g. [39] and [90]). The idea of 'attractiveness' and 'pheromone trails' were also exploited within other meta-heuristic techniques. For example, in the crossover operator used by [158] for solving the 1-PDP. For further information about the various types of Ant systems and their applications, the reader is referred to the book by Dorigo and Stützle [44].

2.3.5 Tabu Search (TS)

Tabu search is another popular search technique proposed by Glover in 1977 [59]. Since then, it has been widely used for solving CO problems. Its name is derived from the word 'taboo' meaning forbidden or restricted. TS, like SA, allows for exploring the search space 'intelligently' in an attempt to escape the trap of local optima. Nevertheless, there are three main differences between TS and SA. Firstly, unlike SA, TS only accepts moves within the vicinity of the current solution that improve the objective function. Secondly, TS always searches for the best solution in the current neighbourhood before applying the replacement criterion. Thirdly, the most distinguishing feature of TS is the use of

a short term memory called a *tabu list*, in which moves that have been recently visited during the search are recorded. Moves in the tabu list are considered prohibited by the search and cannot be visited again for a certain number of iterations. The idea is to avoid the problem of *cycling*, meaning that the search may be trapped within the boundaries of a certain neighbourhood region, oscillating among solutions that have been previously visited, as illustrated in Figure 2.10. By prohibiting recently visited moves, the algorithm is forced to explore new areas of the search space in an attempt to escape local optima.

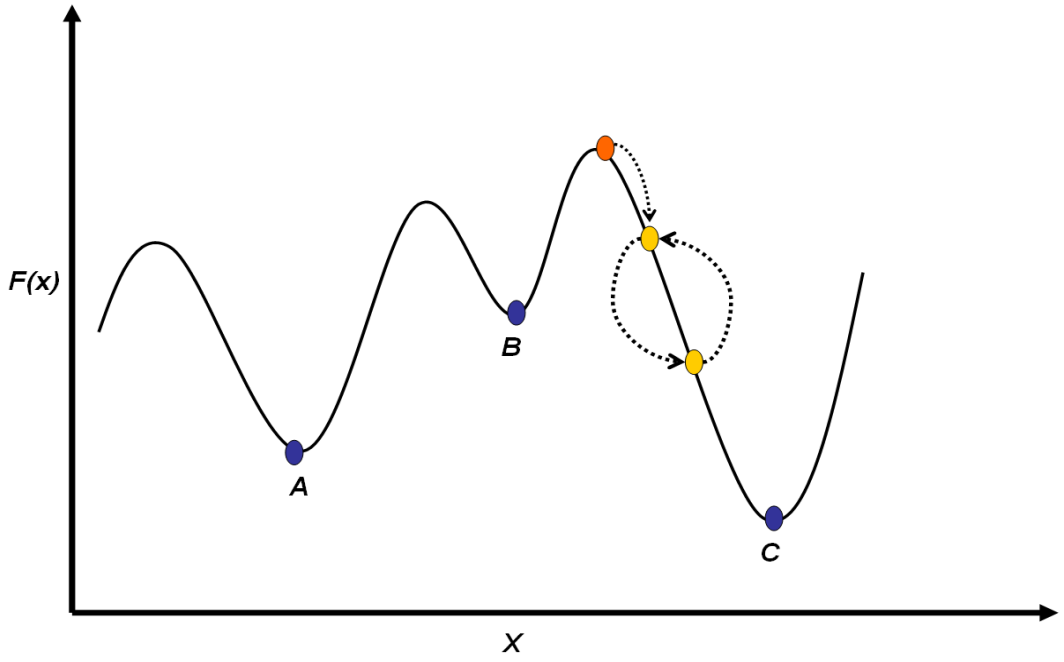


Figure 2.10: The problem of cycling.

The size of the tabu list is usually fixed, such that some old tabu moves are removed to allow for recording new moves recently visited, and the duration that a move is declared tabu is called its *tabu tenure*. Hence, the structure of the neighbourhood being searched varies dynamically from one iteration to another. However, always restricting the search to non-tabu moves may prevent some promising search areas from being explored. To avoid this problem, TS often makes use of an *aspiration criteria*, which allow overriding the tabu status of some moves that look attractive from a search perspective. For example, an aspiration criterion may override the tabu status of a newly generated solution, if its objective value is better than the best solution found so far.

To determine the tabu status of certain solutions, it is common in tabu search to identify particular solution features or neighbourhood moves as 'undesirable'. Accordingly, a newly generated solution holding such features will be considered tabu. Similarly, some

previously identified tabu moves may be prohibited during certain stages of the search. Algorithm 2.5 shows the main steps of TS for a cost minimization problem.

Algorithm 2.5: The Tabu Search (TS) Algorithm.

1: Initialize an empty Tabu List T
2: Generate an initial solution x
3: Let $x^* \leftarrow x$ {x^* is the best so far solution}
4: **repeat**
5: Generate a subset S of solutions in $N(x)$ {$N(x)$ is the current neighbourhood of x}
6: Select the best neighbourhood move $x' \in S$, where $f(x') < f(x)$
7: **if** $(f(x') < f(x^*))$ **then**
8: $x^* \leftarrow x'$ {aspiration condition: if current solution improves best so far, accept it even if it is in the tabu list}
9: $x \leftarrow x'$ {update current solution}
10: $T \leftarrow T + x'$ {update tabu list}
11: **else**
12: **if** $(x' \in N(x) \setminus T)$ **then**
13: $x \leftarrow x'$ {update current solution if the new solution is not tabu}
14: **if** $(f(x') < f(x^*))$ **then**
15: $x^* \leftarrow x'$ {update the best so far solution if the new solution is better in quality}
16: $T \leftarrow T + x'$ {update tabu list}
17: **until** (Done){stopping condition is reached}
18: Return solution x^*

It is also sometimes fruitful in TS to make use of an *intensification* and/or a *diversification* mechanism. Intensification tries to enhance the search around good solutions, while diversification tries to force the algorithm to explore new search areas, in order to escape local optima. For example, intensification can be performed by encouraging solutions that have some common features with the current solution. On the other hand, diversification may be enforced by applying a penalty in the objective function, at some stage of the search, to solutions that are close to the present one [156].

Some variations of TS also exist in the literature, for example *Probabilistic TS* assigns a probability to neighbourhood moves, such that some attractive moves that lower the solution cost are given a higher probability, while moves that result in a repetition of some previous state are given a lower probability [63]. Also, a *Reactive TS* was proposed by Battiti and Tecchiolli [8] in which the size of the tabu list is adapted dynamically during the search, according to the frequency of repeated moves.

TS has been extensively applied to many CO problems. For example, it has been applied

to scheduling in manufacturing systems in [113], to uniform graph partitioning in [36], to the Quadratic Assignment Problem (QAP) in [140]. In addition, it was applied to many variants of vehicle routing problems, for example in [129], [100], [112], [143] and [99]. For more details about some variations and applications of tabu search, the reader is referred to [62] and [55].

2.3.6 Variable Neighbourhood Search (VNS)

Variable Neighbourhood Search (VNS) is a relatively new meta-heuristic that has been suggested by Hansen and Mladenović in [69] and [70]. The main idea is based on exploring the search space by gradually increasing the neighbourhood size, within which a new solution is generated, until a certain stopping condition is reached. In addition, whenever a new solution is generated in the current neighbourhood, a local search is applied to this solution to optimize it before the replacement decision is undertaken. The basic VNS algorithm, as described in [70], is shown in Algorithm 2.6.

Algorithm 2.6: The Variable Neighbourhood Search (VNS) Algorithm [70] .

1: ***Initialization***: Select the set of neighbourhood structures N_k, $(k = 1, ...k_{max})$, that will be used in the search
2: Generate an initial solution x
3: **repeat**
4: $k \leftarrow 1$
5: **while** $(k < k_{max})$ **do**
6: ***Shaking***: Generate a point x' at random from the k^{th} neighbourhood of x $(x' \in N_k(x))$
7: ***Local Search***: Apply some local search method with x' as initial solution; denote with x'' the so-obtained local optimum
8: ***Move or not***: if the local optimum x'' is better than the incumbent, move there $(x \leftarrow x'')$, and continue the search with N_1 $(k \leftarrow 1)$; otherwise set $k \leftarrow k + 1$
9: **until** (Done){stopping condition is reached}
10: Return solution x

In the above VNS algorithm, the neighbourhood size k increases from 1 to a certain maximum value k_{max}. The most important step is the **Shaking** step in which a new point (solution) is generated within the current neighbourhood $N_k(x)$. It is crucial to choose a shaking procedure that will allow enough perturbation of the solution, while preserving, at the same time, the most favourable solution features which should be utilized in obtaining

a near optimum solution as the search progresses. For example, in solving the TSP, shaking my be performed by displacing or inverting a sequence of k cities in each iteration, i.e., the number of cities to be displaced is the variable neighbourhood size that gradually increases from one iteration to the next. On the other hand, the **local search** step in the VNS algorithm intensifies the search to obtain a local optimum, which may replace the current solution. For example, in a TSP a 2-Opt improvement heuristic may be applied as a local improvement method (see Section 3.4 for more details about the 2-Opt heuristic).

The basic VNS procedure is a descent algorithm with a first acceptance criterion. However, the algorithm can be easily changed to a descent/ascent procedure, similar to SA, by accepting bad moves with a certain probability in Step 8 (*move or not*) of Algorithm 2.6. Also, the *first improvement* criterion adopted by the basic VNS in the same step, may be changed to a *best improvement* by selecting the neighbourhood k^*, which yields the best improvement, among all k_{max} neighbourhoods. Other variations include starting the neighbourhood size from a certain value k_{min} rather than 1, and varying the step size, such that increasing the neighbourhood size allows jumps to far away regions of the space [70].

Another important variant of the basic VNS algorithm, called Variable Neighbourhood Descent (VND), was also suggested by Hansen in [70]. The idea is to apply a change of neighbourhood size within the local search as well. The steps of the VND algorithm as described in [70] are shown in Algorithm 2.7.

Algorithm 2.7: The Variable Neighbourhood Descent (VND) Algorithm [70] .

1: ***Initialization***: Select the set of neighbourhood structures N_k, $(k = 1, ...k_{max})$, that will be used in the local search
2: Generate an initial solution x
3: **repeat**
4: Set $k \leftarrow 1$
5: **while** $(k < k_{max})$ **do**
6: ***Exploration of the neighbourhood***: Find the best neighbour x' of x $(x' \in N_k(x))$
7: ***Move or not***: if x' is better than x, set $(x \leftarrow x')$; otherwise set $k \leftarrow k + 1$.
8: **until** (Done){no further improvement is obtained}
9: Return solution x

The VNS meta-heuristic offers many attractive features that have encouraged researchers to use it for CO problems. First, the algorithm is simple, easy to understand and implement, and, to a large extent, parameter-independent. Second, well-known neighbourhood moves and local search methods can be easily integrated in several ways and applied wi-

thin the VNS framework. Third, it is a robust technique that can be used in a variety of problem types with almost no modification to the basic structure of the algorithm. Finally, it can be easily hybridized with other heuristic and meta-heuristic approaches to solve hard optimization problems. In fact, VNS has been successfully used in solving many well-known CO problems. For example, it has been applied to the TSP and the facility location problem in [70], and to the graph coloring problem in [6]. It has also been extensively applied to vehicle routing problems, for example in [19], [119], [118] and [73]. For more details about advances in VNS and its applications, the reader is referred to the recent work by Hansen *et al.* [71]. As previously mentioned, our research applied the VNS approach, hybridized with SA, in solving the 1-PDP, as will be explained in detail in Chapter 10.

2.4 Chapter Summary

Conventional OR techniques may not be sufficient for solving complex optimization problems, which model a large number of decision making strategies in real-world applications. Heuristic and meta-heuristic techniques, which usually perform well in most practical situations, have become increasingly popular among researchers in the optimization field.

In this chapter we introduced combinatorial optimization problems and the theory of algorithm and complexity analysis. We then briefly highlighted some exact algorithms that can be used to solve these problems to optimality, for limited problem sizes only. For more practical applications, though, a heuristic or a meta-heuristic approach is usually the preferred option. We reviewed in this chapter some important meta-heuristic techniques like Local Search, Simulated Annealing, Genetic Algorithms, Ant Colony Optimization, Tabu Search, and Variable Neighbourhood Search.

In the next chapter, we will introduce an important class of CO problems, which is the class of vehicle routing and scheduling. A particular variant of vehicle routing problems, **pickup and delivery problems**, is introduced in Chapter 4. Since pickup and delivery problems are the main theme of our research, our investigation of some selected problems from this category continues in Chapters 5 to 10.

Chapter 3

Vehicle Routing Problems: A Literature Review

Solving the Vehicle Routing Problem (VRP) and its related variants is an important field in the area of Operations Research (OR), which has attracted a growing interest in recent years. Effective decision support tools that can be adopted in logistic planning are currently in great demand, since they can lead to a substantial cost reduction and efficient resource consumption. In addition, they can help reduce the negative environmental impacts of transportation. An European Commission White Paper [1] states:

> A modern transport system must be sustainable from an economic and social as well as an environmental viewpoint. Plans for the future of the transport sector must take account of its economic importance. Total expenditure runs to some EUR 1000 billion, which is more than 10% of gross domestic product. The sector employs more than 10 million people. It involves infrastructure and technologies whose cost to society is such that there must be no errors of judgment.

The use of automated route planning and scheduling can lead to huge savings in transportation costs, typically ranging from 5% to 20% [147], which should contribute to boosting the overall economic system. This great potential, together with the advancement in technological and computational powers in the last three decades, has encouraged researchers to experiment with diverse algorithms and address different applications, in order to meet the increase in demand for effective vehicle routing support tools. This indeed resulted in a large increase in the literature dealing with the VRP, especially given that its theme encompasses several intertwined disciplines, such as scientific computing, operations research, and business and management [48].

The rest of this chapter is organized as follows: Section 3.1 gives an idea about the different types of vehicle routing problems existing in the literature and highlights the diversity of published research in this area. Section 3.2 briefly describes one particular VRP variant that is of interest to our research: the Vehicle Routing Problem with Time Windows

(VRPTW). Section 3.3 emphasizes heuristic approaches for the solution construction of the VRPTW, while Section 3.4 describes some popular solution improvement heuristics. Section 3.5 then briefly summarizes some meta-heuristic algorithms that were applied to the VRPTW. Finally, section 3.6 concludes this chapter with a short summary.

3.1 Vehicle Routing Problems (VRPs)

VRPs generally involve problems that deal with the transportation of goods or people between depots and customers, where the objective is to design an optimum schedule for one or more vehicles to service the clients with minimum possible operational cost and maximum customer satisfaction. For example, the routing and scheduling plan may try to minimize the number of vehicles used, the total distance traveled, and the labour force used. At the same time, it may try to maximize orders and volumes serviced per unit distance. Moreover, for practical applicability of the problem, a number of constraints are usually added to the basic model. These may include respecting the capacity of the vehicle, applying a certain visiting order, or adhering to preferred service times for clients and to the maximum working hours of drivers. Other practical considerations include service types for customers (delivery and/or collection), number of depots (one/two/multiple), types of operating vehicles (homogeneous/heterogeneous), drivers availability (fixed/variable start times), and whether the information concerning routing requests is known in advance or revealed in real-time.

As a generalization of the famous TSP, the VRP is an $\mathcal{NP}$*-hard* problem [54] [147]. This fact, together with the above mentioned constraints and practical considerations, add to the difficulty of handling the different variants of vehicle routing problems, and contribute to the existence of a wealth of models and algorithms that deal with this problem.

Given the large amount of published research tackling the various types of VRPs [48], it becomes increasingly difficult to keep track of all problem types and underlying solution methods. However, a general classification of the basic routing and scheduling problems and their interconnections is shown in Figure 3.1. Two main problem categories are distinguished in this figure, based on the observation that routing and scheduling problems are usually represented as graphical networks, in which pickup and/or delivery points are represented as nodes connected with line segments called arcs. The first category is called **node routing problems**, in which the service demand is associated with nodes (locations). The second category is called **arc routing problems**, in which the service demand is associated with the arc connecting two nodes. Arc routing problems involve applications like refuse collection and winter gritting, and are often referred to as

the Chinese Postman Problem (CPP) [66]. The CPP, though, is beyond the scope of our research and will not be addressed further. As Figure 3.1 indicates, the simplest type of node routing problems is the TSP, in which a single vehicle is required to visit a number of nodes, with no restriction on the capacity of the vehicle. It is required in the TSP that the trip starts and ends at the same node, without identifying a particular depot point. The multiple-vehicle variant of the TSP is called the Multiple Traveling Salesman Problem (MTSP). If the demands of locations are added to the problem, the capacity of the vehicle is restricted, and a depot point is identified, it is usually called the Capacitated Vehicle Routing Problem (CVRP). Requesting two different service types (pickup and delivery) in the problem instance, transforms it to the Pickup and Delivery Problem (PDP) category. In addition, a service time window (TW) may be imposed on any of the above variants to allow for more realistic applications of the problem, in which the visiting time of each node is restricted between certain pre-defined bounds. For further details about the main variants of the VRP and both exact and heuristic methods applied to solving it, the reader is referred to the book edited by Toth and Vigo [147].

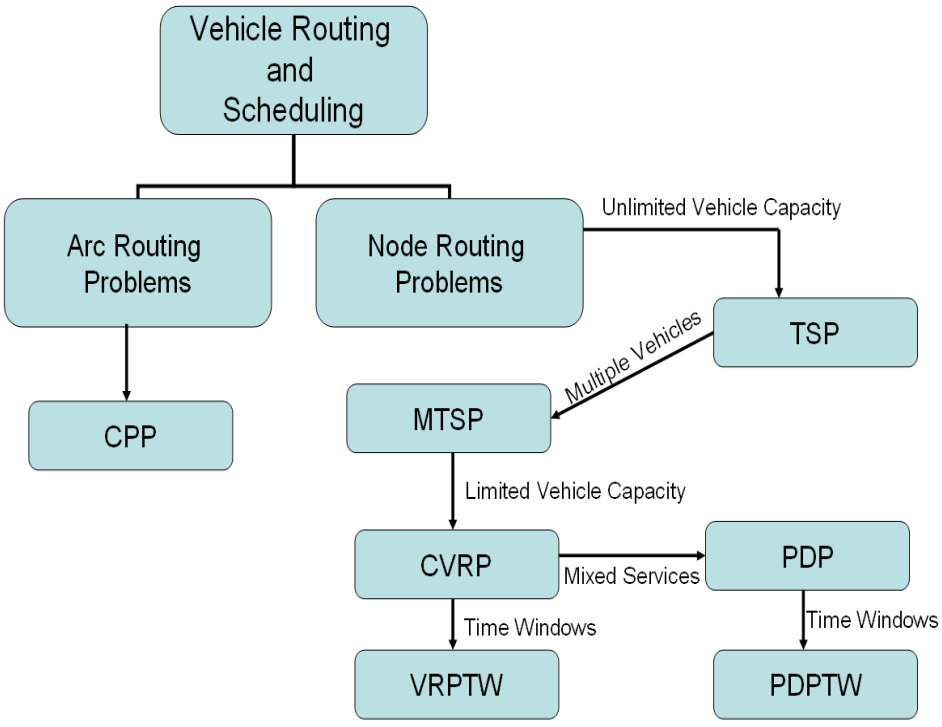


Figure 3.1: Vehicle routing and scheduling problems.

Recently, Eksioglu *et al.* in 2009 carried out a taxonomic review of the VRP [48], in which they demonstrated that the growth in the literature published from 1955 to 2005 is almost perfectly exponential, if we eliminate the first two years of data. Despite this large increase in published VRP literature, other OR/MS (Operations Research/Management Sience) disciplines have grown at a much higher rate, which indicates that research in the VRP area is probably more sophisticated and requires high skills in analysis and design. The authors classified the characteristics of different VRPs, which should help understand the extent of the available literature and also identify potential areas where more research seems to be needed. Their taxonomy of the VRP research as it appears in [48] is shown in Figure 3.2.

This taxonomy classifies the VRP literature under five major categories:

1. **Type of Study:** which identifies the nature of the study itself, for example: theoretical or applied. Some techniques listed under applied literature include exact and heuristic methods.

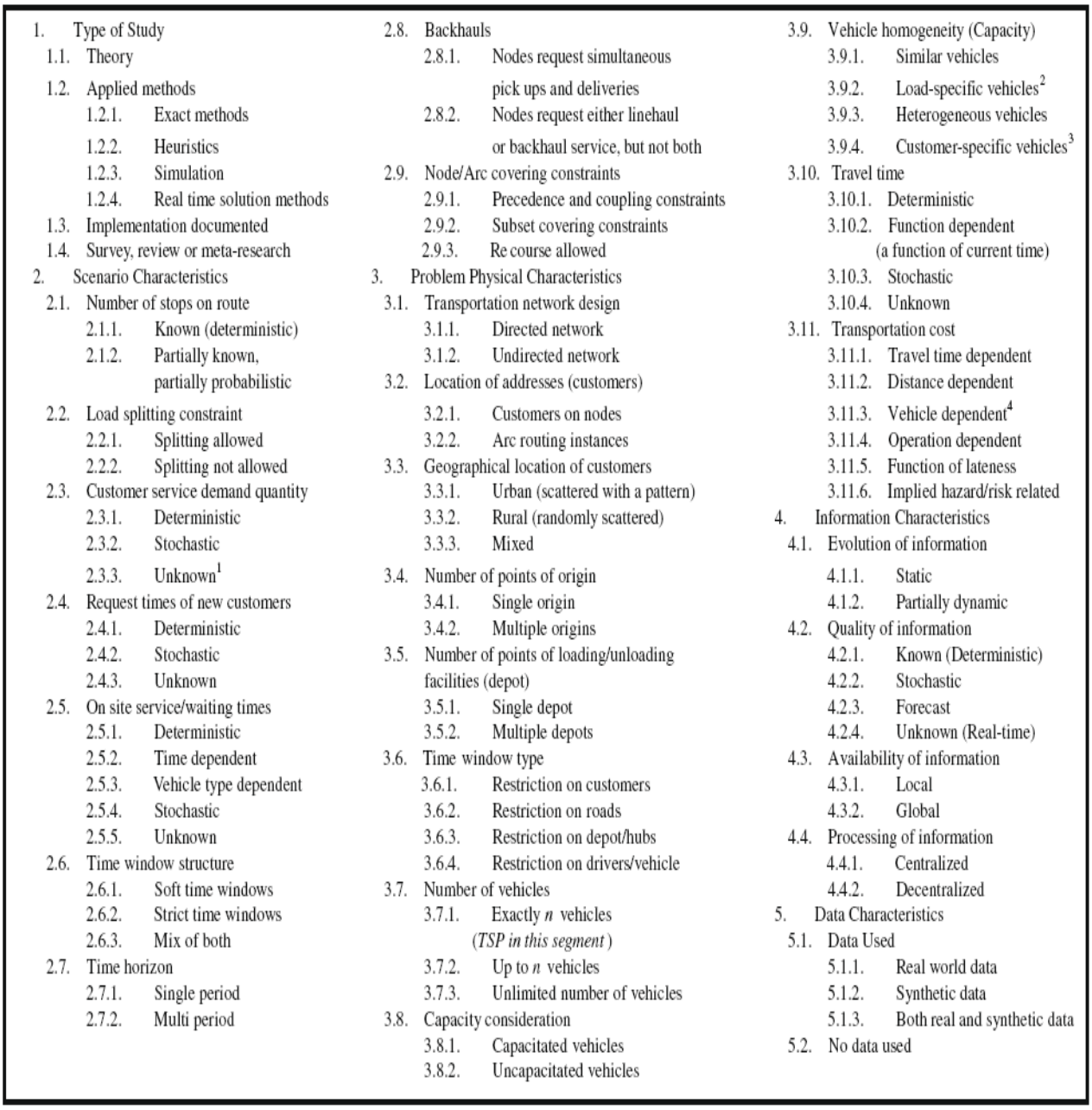
2. **Scenario Characteristic:** which indicates the problem characteristics and the operating scenario of the vehicle routing process. For example, under this category, *static* problems, in which all requests are determined in advance of the solution process, are distinguished from *dynamic* problems in which some requests may arrive later (in real-time). Dynamic problems should account for some probable changes in the current progressing routing plan. Other problem characteristics identified here include the type of service (pickup/delivery), and the precedence and coupling requirements [1].

3. **Problem Physical Characteristics:** which outlines factors that directly affect the solution, such as number of depots, number of vehicles, capacity and time windows constraints. This category also distinguishes node routing problems from arc routing problems.

4. **Information Characteristics:** which identifies the nature of information presented and accessed by the underlying solution methodology, and is mainly directed to the study of fuzzy and dynamic routing problems. For example, acting under uncertainty, when certain information (e.g. delays/vehicle breakdown) are revealed during an emergency situation, may fall under this category.

5. **Data Characteristics:** identifies the type of data used to evaluate the solution method, such as real-world or synthesized data.

[1] Precedence means that a pickup location must precede its corresponding delivery, while coupling requires that both the pickup and its delivery must be served by the same vehicle.



1. Type of Study
   - 1.1. Theory
   - 1.2. Applied methods
     - 1.2.1. Exact methods
     - 1.2.2. Heuristics
     - 1.2.3. Simulation
     - 1.2.4. Real time solution methods
   - 1.3. Implementation documented
   - 1.4. Survey, review or meta-research
2. Scenario Characteristics
   - 2.1. Number of stops on route
     - 2.1.1. Known (deterministic)
     - 2.1.2. Partially known, partially probabilistic
   - 2.2. Load splitting constraint
     - 2.2.1. Splitting allowed
     - 2.2.2. Splitting not allowed
   - 2.3. Customer service demand quantity
     - 2.3.1. Deterministic
     - 2.3.2. Stochastic
     - 2.3.3. Unknown[1]
   - 2.4. Request times of new customers
     - 2.4.1. Deterministic
     - 2.4.2. Stochastic
     - 2.4.3. Unknown
   - 2.5. On site service/waiting times
     - 2.5.1. Deterministic
     - 2.5.2. Time dependent
     - 2.5.3. Vehicle type dependent
     - 2.5.4. Stochastic
     - 2.5.5. Unknown
   - 2.6. Time window structure
     - 2.6.1. Soft time windows
     - 2.6.2. Strict time windows
     - 2.6.3. Mix of both
   - 2.7. Time horizon
     - 2.7.1. Single period
     - 2.7.2. Multi period
   - 2.8. Backhauls
     - 2.8.1. Nodes request simultaneous pick ups and deliveries
     - 2.8.2. Nodes request either linehaul or backhaul service, but not both
   - 2.9. Node/Arc covering constraints
     - 2.9.1. Precedence and coupling constraints
     - 2.9.2. Subset covering constraints
     - 2.9.3. Re course allowed
3. Problem Physical Characteristics
   - 3.1. Transportation network design
     - 3.1.1. Directed network
     - 3.1.2. Undirected network
   - 3.2. Location of addresses (customers)
     - 3.2.1. Customers on nodes
     - 3.2.2. Arc routing instances
   - 3.3. Geographical location of customers
     - 3.3.1. Urban (scattered with a pattern)
     - 3.3.2. Rural (randomly scattered)
     - 3.3.3. Mixed
   - 3.4. Number of points of origin
     - 3.4.1. Single origin
     - 3.4.2. Multiple origins
   - 3.5. Number of points of loading/unloading facilities (depot)
     - 3.5.1. Single depot
     - 3.5.2. Multiple depots
   - 3.6. Time window type
     - 3.6.1. Restriction on customers
     - 3.6.2. Restriction on roads
     - 3.6.3. Restriction on depot/hubs
     - 3.6.4. Restriction on drivers/vehicle
   - 3.7. Number of vehicles
     - 3.7.1. Exactly n vehicles (*TSP in this segment*)
     - 3.7.2. Up to n vehicles
     - 3.7.3. Unlimited number of vehicles
   - 3.8. Capacity consideration
     - 3.8.1. Capacitated vehicles
     - 3.8.2. Uncapacitated vehicles
   - 3.9. Vehicle homogeneity (Capacity)
     - 3.9.1. Similar vehicles
     - 3.9.2. Load-specific vehicles[2]
     - 3.9.3. Heterogeneous vehicles
     - 3.9.4. Customer-specific vehicles[3]
   - 3.10. Travel time
     - 3.10.1. Deterministic
     - 3.10.2. Function dependent (a function of current time)
     - 3.10.3. Stochastic
     - 3.10.4. Unknown
   - 3.11. Transportation cost
     - 3.11.1. Travel time dependent
     - 3.11.2. Distance dependent
     - 3.11.3. Vehicle dependent[4]
     - 3.11.4. Operation dependent
     - 3.11.5. Function of lateness
     - 3.11.6. Implied hazard/risk related
4. Information Characteristics
   - 4.1. Evolution of information
     - 4.1.1. Static
     - 4.1.2. Partially dynamic
   - 4.2. Quality of information
     - 4.2.1. Known (Deterministic)
     - 4.2.2. Stochastic
     - 4.2.3. Forecast
     - 4.2.4. Unknown (Real-time)
   - 4.3. Availability of information
     - 4.3.1. Local
     - 4.3.2. Global
   - 4.4. Processing of information
     - 4.4.1. Centralized
     - 4.4.2. Decentralized
5. Data Characteristics
   - 5.1. Data Used
     - 5.1.1. Real world data
     - 5.1.2. Synthetic data
     - 5.1.3. Both real and synthetic data
   - 5.2. No data used

[1] Unknown refers to the case in which information is revealed in real-time (i.e., dynamic and fuzzy studies fall under this category)

[2] Each vehicle can be used to handle specific types of loads

[3] A customer must be visited by a specific type of vehicle

[4] Cost of operating a vehicle is not negligible

Figure 3.2: Taxonomy of the VRP literature [48].

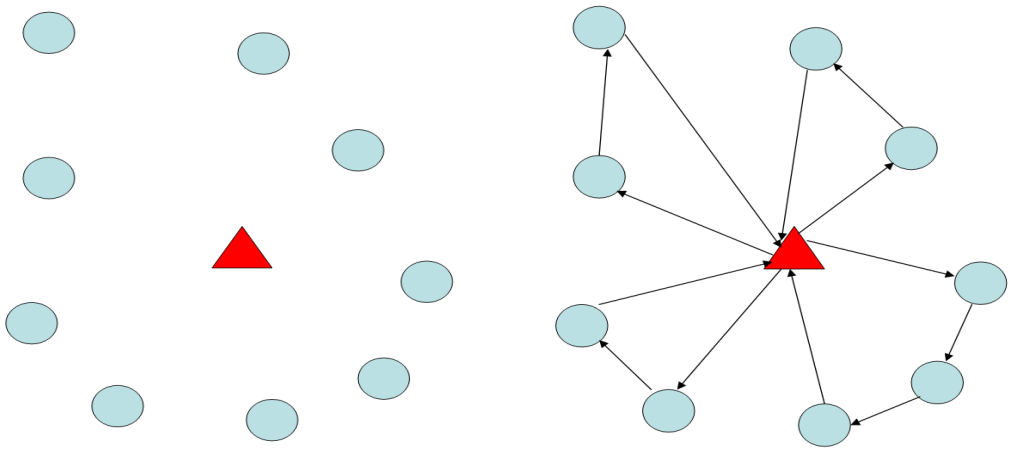
The above taxonomy gives a general idea about how sophisticated and diverse the VRP literature is. As previously mentioned in the introduction to this thesis (Chapter 1), our research mainly focuses on one particular variant of vehicle routing problems, which are problems that deal with two types of services simultaneously, *pickup and delivery* (PD). Specifically, we selected two main problems from this category, the Pickup and Delivery Problem with Time Windows (PDPTW) and the One-commodity Pickup and Delivery Problem (1-PDP), with more emphasis on the first of these two variants. In the next chapter, we will present a brief summary of some important PD problems, and give examples of published research in this field. On the other hand, we will devote the rest of this chapter to an important and closely related problem in the literature, which is the Vehicle Routing Problem with Time Windows (VRPTW). This problem is significant to our research, because it bears similarities to our selected PD problems, in terms of the constraints involved and the solution approaches used, as well as the objective function. Moreover, it is one of the most well-studied problems in the vehicle routing field.

As previously mentioned in Chapter 2, exact algorithms have their limitations in solving large scale combinatorial optimization problems. In the vehicle routing domain, Hasle and Kloster [72] indicate that today's exact methods cannot consistently solve VRP instances having more than 50-100 customers, which is generally small for most realistic applications. Moreover, given the complexity of the problem, it is very unlikely that an algorithm that solves the problem to optimality in a reasonable time will be developed in the near future. Hence, our research mainly investigates heuristics and meta-heuristics as possible solution methods to the problems of interest. This literature review will, therefore, focus on the *approximation* techniques introduced in Chapter 2. In other words, we will highlight selected published research that deals with heuristic and meta-heuristic methods only. Information about research that uses exact methods can be found in several vehicle routing and pickup and delivery problems surveys that we are going to mention throughout this discussion . In this chapter and the next chapter, a general classification and overview of the literature is presented, while more details about previous research particularly related to the individual problems we tackled will be presented in the chapters dedicated to these problems (i.e., Chapters 5 to 10).

3.2 The Vehicle Routing Problem with Time Windows (VRPTW)

The VRPTW is an important problem occurring frequently in transportation systems. The problem deals with a number of customer requests that must be dealt with by a fleet of

vehicles. All customers are assumed to require the same type of service (either pickup or delivery but not both) [2]. Each vehicle route must start and end at a central depot, and each customer must be visited exactly once. Two main constraints are strictly enforced in this problem. The capacity constraint requires that the vehicle capacity should not be exceeded at any time, while the time window (TW) constraint requires that each customer must be visited during a pre-specified time window interval, i.e., if the vehicle reaches the customer before the beginning of its TW, it should wait until the allowed service time begins. Similarly, arrival after the deadline of the TW means a violation of problem constraints. The solution objective is usually hierarchical, such that the minimization of the number of vehicles used is a primary objective, followed by minimizing the total travel distance of the operating fleet, or the total schedule duration. Figure 3.3 shows a representation of a small instance of the VRPTW, before solving it (Figure 3.3(a)) and after solution (Figure 3.3(b)).



(a) VRPTW - before solution (b) VRPTW - after solution

Figure 3.3: The vehicle routing problem with time windows.

The VRPTW is one of the most well-studied problems in the vehicle routing research. Both exact and heuristic methods have been widely applied to solving this problem. In our current literature summary, we mostly base our review on the 2-part survey of the VRPTW by Bräysy and Gendreau [20] and [21]. As previously mentioned, our investigation of the problem focuses on heuristic and meta-heuristic solution methods, since exact methods are only limited to small problem sizes. Up-to-date exact algorithms were only capable of solving some of Solomon's benchmark instances (developed in [141]) of 100 customers

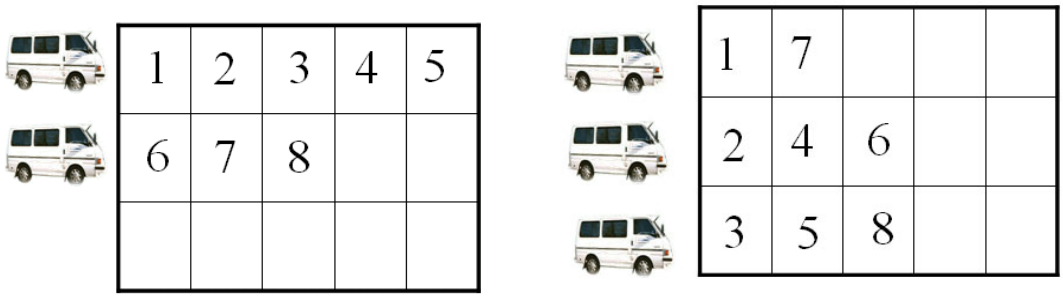
[2] In the current discussion, we use the terms ***requests*** and ***customers*** interchangeably to refer to the set of nodes to be visited. Nevertheless, a distinction between the two terms will be introduced within the context of the PDPTW in Chapters 5 to 8.

to optimality. Moreover, most of the solved instances are characterized by having a short schedule horizon (a short TW width). The remaining (unsolved) instances have a long schedule horizon, and are much harder for exact algorithms to solve. For further details about some important exact algorithms in the VRPTW literature, the reader is refereed to Cordeau *et al.* [28].

Solving the VRPTW generally requires two types of decisions: 1) the *assignment* or the *grouping* decision, which means to assign to each vehicle a subset of nodes from the customers set, and 2) the *routing and scheduling* decision which involves generating a minimum cost route for each vehicle to visit its assigned requests, such that the generated route respects the capacity and TW constraints. It is often observed that for vehicle routing problems of this sort, the assignment decision is usually of more importance than the routing and the scheduling decision, in determining the final solution quality [137]. In addition, the solution process often consists of two phases: *solution construction*, in which one or more solutions for the problem is generated, and *solution improvement*, in which the initial solution is improved using a heuristic or a meta-heuristic approach. Both phases are discussed in the following two sections.

3.3 Solution Construction for the VRPTW

Solution construction refers to the creation of a set of routes for the vehicles by selecting nodes (customers) and inserting them in one of the partial routes already created, or in a new route. The decision to select a particular node for insertion is usually based on a *cost-minimization* criterion, and requires that the insertion of the node in a selected route does not cause violations of problem constraints. Two main types of solution construction exist in the VRPTW literature: **sequential construction**, which builds routes one after the other, and **parallel construction** which builds several routes at the same time. Sequential construction does not attempt to allocate an additional vehicle unless no more requests can be 'feasibly' added to the current vehicle. A parallel construction, on the other hand, initially pre-specifies the number of vehicles that could be used, but more vehicles can be added as needed if the initial estimate of the number of vehicles does not serve all requests without violating the constraints of the problem. Figure 3.4 demonstrates these two variants.



(a) Sequential construction (b) Parallel construction

Figure 3.4: Solution Construction.

Several famous construction heuristic are described by Solomon in [141]. These are the *savings heuristics*, the *time oriented nearest neighbour heuristic*, the *insertion heuristics*, and the *time oriented sweep heuristic*. These heuristics are briefly explained below.

3.3.1 Savings Heuristics

This method is based on one of the most successful construction heuristics for the VRP, the *savings heuristic* proposed by Clarke and Wright [26]. The idea is to initially assign each customer individually to one vehicle. Thus, an initial vehicle route will only consist of a trip from the depot to the customer and then back to the depot. If we have two such routes, one serving customers i and another serving customer j, combining the two routes such that the new route will service both i and j will result in a saving of service cost $S_{ij} = c_{i0} + c_{0j} - c_{ij}$. This idea is illustrated in Figure 3.5.

The *parallel* savings construction algorithm for the VRPTW can be summarized as follows:

1. **Step 1:** compute saving S_{ij} for every two customers i and j, and sort them from largest to smallest;
2. **Step 2:** take saving S_{ij} from top of the saving list. Search for a route containing $(0, i)$ and another containing $(j, 0)$. If such routes exist, join them to form route $(0, i, j, 0)$.

Solomon in [141] adapted the Clarke and Wright savings heuristic to the VRPTW by taking the route orientation into account, as shown in Figure 3.5 and Step 2 above. In addition the algorithm must check the capacity and time window feasibility at every step

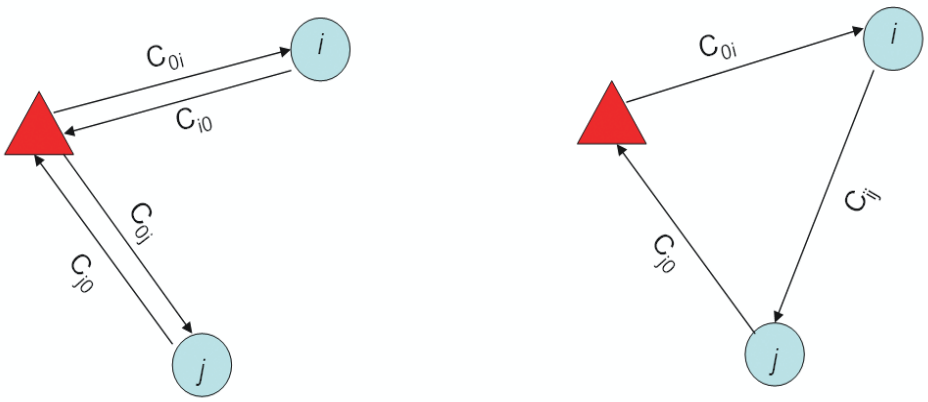


Figure 3.5: The savings heuristic.

in the heuristic process [3]. A limit on the waiting time is also imposed when routes are joined.

A *sequential* version of the savings heuristic will consider one route at a time, and then implement the best saving that can be achieved if another route is joined with the current route. The sequential savings heuristic can be briefly described as follows:

1. **Step 1:** compute saving S_{ij} for every two customers i and j, and sort them from largest to smallest;

2. **Step 2:** consider the current route $(0, i, ..., j, 0)$. Find routes containing $(k, 0)$ and $(0, l)$ that have the best savings S_{ki} and S_{jl} in the saving list, and select the best one among them for joining with the current route. If no more savings can be implemented for the current route, move to the next route.

3.3.2 Time Oriented Nearest Neighbour Heuristic

This heuristic is a *sequential construction* process that starts by initializing the current route with the depot. Then the customer closest to the depot, in terms of a certain cost measure c, is inserted next in the route. Subsequently, the customer 'closest' to the last added customer will be inserted next, if its insertion causes no constraint violation. If

[3] Solomon also proposed an important heuristic for speeding the TW feasibility checking called the *push forward (PF)* heuristic, which is based on calculating the extra time occurring in the schedule due to the insertion of the new node in the route. For more derails see [141].

all remaining un-routed customers cannot be feasibly inserted in the current route, a new route is allocated and the process is repeated.

The cost function c used by Solomon in [141] to determine the 'closeness' between two customers, is based on both *geographical* and *temporal* separation between the two nodes. This measure is described by $c_{ij} = w_1 \times d_{ij} + w_2 \times T_{ij} + w_3 \times v_{ij}$, where d_{ij} is the distance between the two customers[4], T_{ij} is the time difference between the completion of the service at i and the beginning of the service at j, and v_{ij} is the urgency of visiting customer j, which is calculated as the time remaining until the deadline of servicing customer j is reached. The weights w_1, w_2 and w_3 satisfy: $w_1 + w_2 + w_3 = 1.0$, and $w_1 \geq 0$, $w_2 \geq 0$ and $w_3 \geq 0$.

3.3.3 Insertion Heuristics

This is another class of *sequential construction* heuristics described by Solomon in [141], and is based on expanding the current route by inserting one un-routed customer at each iteration. The general idea is demonstrated in Figure 3.6, where the un-routed customer k (left) was inserted between customers i and j (right) in the progressing route.

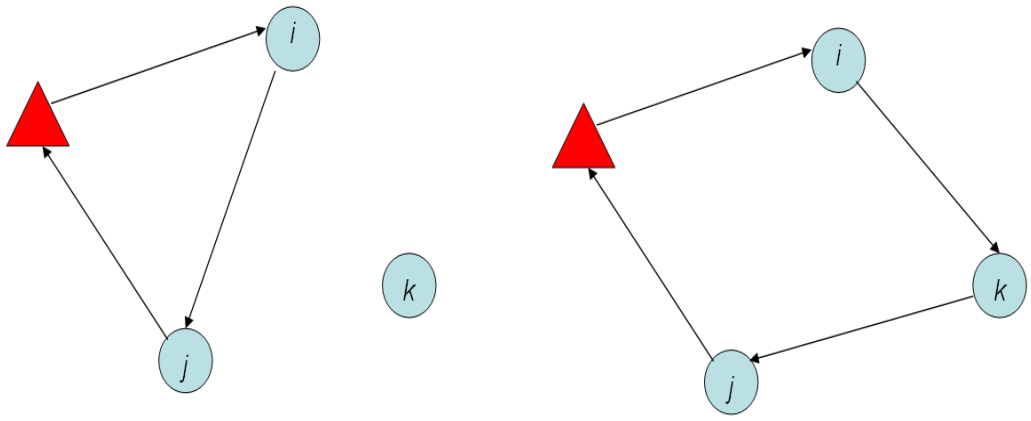


Figure 3.6: The insertion heuristic.

More specifically, the insertion heuristic starts by initializing the route under consideration with a *seed* customer. Other customers are then selected for addition to the current route,

[4]One simplifying assumption that is usually made in the VRP is that the speed of the vehicle is constant (often = 1). If this assumption is made, the distance separating i and j (d_{ij}) and the travel time between i and j (t_{ij}) may be used interchangeably.

based on two cost measures to be defined shortly. When no more un-routed customers can be 'feasibly' inserted in the current route, the process is repeated for a new route. The seed customer could be selected as the customer farthest from the depot, or as the customer having the earliest allowed starting service time.

Three insertion heuristics have been defined by Solomon, based on the criteria used for selecting an un-routed customer to be inserted next in the current route. The first and most successful insertion heuristic (called **I1** insertion) proceeds as follows, after initializing the route with a seed customer:

1. Assume the current partial route created is $(i_0, i_1, i_2, ...i_m)$, where i_0 and i_m represent the depot;
2. For each un-routed customer u, find its best feasible insertion position in the route, between two adjacent customers i and j. The best insertion position is the one that *minimizes* the first cost measure, which we will call $c_1(u)$. $c_1(u)$ is a measure that is calculated based on both the *extra* travel distance and time delay that happens in the route, due to the insertion of u;
3. Find the best customer u^* to be inserted in the route, where u^* is the customer having the *maximum* value of a second measure, which we will call $c_2(u)$. $c_2(u)$ is calculated based on both $c_1(u)$ (the first measure) and the distance from the depot.

The second insertion heuristic **(I2)** differs from I1 in the definition of the measure $c_2(u)$, since it is now a combined measure of the *total* route distance and time resulting after the insertion of u (rather than the *extra* distance and time, as in I1). On the other hand, the third sequential insertion heuristic **(I3)** differs from I1 in the definition of $c_1(u)$, since the urgency of servicing the new customer u is now also considered in this measure[5].

These insertion heuristics are in fact a generalization of the time oriented nearest neighbour heuristic, described in Section 3.3.2, since they allow for inserting a new customer between any two nodes in the route, rather than only at the end. They are sometimes referred to in the literature as the *cheapest insertion* heuristics, with I1 being the most popular among them.

A *parallel construction* heuristic for the VRPTW was proposed by Potvin and Rousseau [121], based on the I1 heuristic described above. In this heuristic several routes are first initialized with seed customers. To determine the number of initial routes, the authors

[5]Our presentation of Solomon's insertion techniques and the cost measures adopted within these heuristics is simplified to a large extent, in order to fit the general overview intended here. For further information, the reader is referred to [141] and [20].

first ran the I1 sequential construction algorithm. Further routes are later added as needed if the initial number of routes does not yield a feasible solution. After determining the best (feasible and least cost) insertion position for each un-routed customer, the customer whose insertion in the solution causes the least increase in the overall cost is selected next for insertion. Besides a weighted sum of the extra travel distance and time delay used by Solomon to determine the cost of the insertion, this parallel heuristic adds a regret measure over all routes. The regret measure is a type of 'lookahead' estimate of the cost of not inserting the customer immediately in its current best route, i.e., if its insertion was postponed to be carried out later in a different route.

Although these construction algorithms have mainly been designed for the VRPTW, researchers have also adapted them to different problem variants, such as the PDPTW. More discussion about solution construction heuristics from the perspective of the PDPTW will be presented in Chapter 7.

3.3.4 Time Oriented Sweep Heuristic

In this heuristic Solomon [141] suggests a two phase approach to construct a VRPTW solution. The first phase is a *clustering phase*, in which customers are assigned to vehicles in a way similar to the original *sweep heuristic* suggested by Gillett and Miller [58]. In this heuristic, a centre of gravity is computed based on the angle between the depot and a randomly selected customer. Then, the remaining customers are assigned to vehicles according to the polar angle separating their locations from the centre of gravity. After the assignment phase, the *scheduling phase* starts by inserting the selected customers in their respective routes, such that the insertion process follows Solomon's I1 insertion heuristic, explained in Section 3.3.3.

3.4 Solution Improvement for the VRPTW

Solution improvement within the context of the VRPTW refers to the gradual and repeated modification of the initial solution until a certain stopping condition is satisfied. The initial solution is usually a feasible solution, obtained using a construction algorithm, such as the ones described in the previous section. One must then define a neighbourhood move that can be applied to the initial/current solution to obtain a new solution within its neighbourhood, i.e., the new solution only differs in a few attributes from the current solution. For example, the new solution could be obtained by modifying some edges connecting customers in the current solution. Afterwards, the generated solution is evaluated based

on the objective function, and may replace the previous solution if it is smaller in cost. Nevertheless, two main acceptance criteria are usually adopted in this context. The *first acceptance* criterion selects the first solution found in the neighbourhood of the current solution that improves the objective function. On the other hand, the *best acceptance* examines all solutions in the neighbourhood of the current solution, and selects the best one among them.

The above process is often referred to as *local search*, and usually results in a local optimum that may be very far from the optimal solution. In addition, the quality of the local optimum depends heavily on the quality of the initial solution. Within the vehicle routing literature, *edge exchanges* are the most popular solution improvement method, first described by Lin [103] for the TSP.

Edge exchanges are applied to one route in the current solution, and depend on removing a number k of edges from the current route, and replacing them with another set of k edges. Thus, the process is often called k-exchange. A route that cannot be further improved by a k-exchange is called a k-optimal. Performing all possible k-exchanges on a route requires $O(n^k)$ time. Thus, moves beyond 2 or 3 exchanges are very time consuming. The 2-exchange move is illustrated in Figure 3.7.

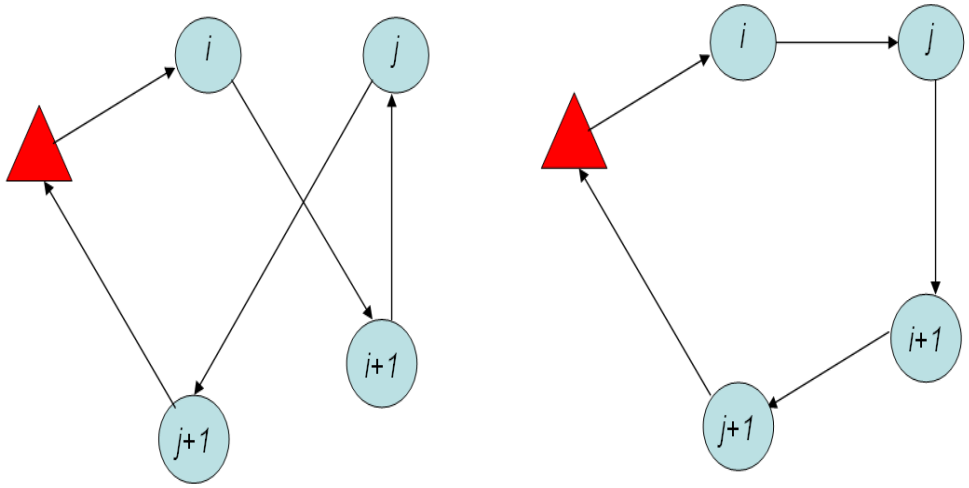


Figure 3.7: 2-exchange move.

Several other improvement methods have been described for the VRPTW, a few of them are modifications to the basic edge-exchange moves. We will briefly mention some of the most important methods, without going into details, since most of them are beyond the

scope of our research.

Potvin and Rousseau [122] introduce a modification to the 2-Opt heuristic of [103], called the ***2-Opt**** exchange heuristic. 2-Opt* works on two different routes (unlike 2-Opt which exchanges edges belonging to the *same* route). 2-Opt* tries to combine the two routes without changing the orientation of the tours, by appending the last customers of the second route after the first customers of the first route. Also, Or [114] modified the 3-Opt operator of [103] and introduced the ***Or-Opt*** operator for the TSP. In this operator, three edges in the original tour are replaced by three new ones without changing the orientation of the tour.

Inter-route operators for the VRP were introduced by Savelsbergh [136] and used by Posser and Shaw [123] for the VRPTW. These are the ***re-locate***, ***exchange*** and ***cross*** operators. The relocate operator moves a customer from one route to another. The exchange swaps two visits in different routes and the cross is similar to the 2-Opt* proposed in [122]. Other neighbourhood operators are the λ***-exchange*** of Osman [115], the ***CROSS-exchange*** of Taillard *et al.* [143], the ***GENI-exchange*** of Gendreau *et al.* [56], ***ejection chains*** of Glover [60] [61] and ***cyclic*** k***-transfers*** of Thompson and Psaraftis [146].

In addition, other improvement heuristics include the ***Large Neighbourhood Search (LNS)*** of Shaw [139]. This algorithm removes a large number of customers from their current routes, and re-schedules them at optimal cost. The removed customers could be selected at random, or based on some relatedness measure that takes into account customers demands or starting times. Schrimpf [138] used a similar approach and calls it ***ruin and recreate***. Finally, Bräysy [18] introduces a ***modified ejection chain*** approach that also considers re-ordering of the routes. For the interested reader, more details and illustrations of the different types of solution improvement heuristics are presented in [20].

Similar to solution construction approaches, solution improvement methods have also been adapted and applied to PD problems. Some examples of their application to these problems will be presented in the upcoming chapters, when related work to our selected problems is investigated.

3.5 Meta-heuristics for the VRPTW

Since our research is particularly focused on meta-heuristic techniques, we highlight in this section some papers that use meta-heuristic approaches for the VRPTW, and explain briefly some important common features among them. Table 3.1 summarizes research

that uses Tabu Search (TS), Simulated Annealing (SA), Ant Colony Optimization (ACO), Variable Neighbourhood Search (VNS), and Guided Local Search (GLS).

As Table 3.1 indicates, most meta-heuristic techniques apply some cheapest insertion method for the construction of the initial solution, with Solomon's insertion being the most popular among them. On the other hand, solution improvement is usually performed using well-known neighbourhood moves like 2-Opt, 2-Opt*, 3-Opt, Or-Opt, λ-interchange... etc. To accelerate the optimization, the search is sometimes limited to neighbourhoods having particular features. For example, in [53] only moves involving arcs close in distance are allowed, while the approach in [22] only considers selected routes and selected customers for merging during the solution construction process.

Some approaches allow infeasibility of solutions (e.g. [99]) and penalize the violation in the objective function. In addition, many algorithms apply a specific strategy for reducing the number of vehicles. For example, the algorithm in [53] removes customers from routes having a small number of nodes and inserts them into other routes, using Or-Opt exchanges. Also, in [11] a 2-phase approach is applied, where the first phase is an SA procedure intended to minimize the number of routes, while the second phase focuses on minimizing total distance. On the other hand, the ACO of [52] uses two colonies, one for the minimization of the number of vehicles and the other to minimize total travel distance.

Different diversification mechanisms have also been applied within the various meta-heuristics to improve solution quality, so that the algorithm is directed towards exploring new areas of the search space. For example, in [129] the search is diversified using an adaptive memory, where routes taken from best solutions found during the search are stored. Some of these routes are combined to form solutions that will act as new starting solutions for the TS. Also, in [143] diversification is applied by penalizing frequently performed exchanges. On the other hand, the approach in [22] adopts a Guided Local Search (GLS) diversification mechanism that penalizes certain solution features, if they are going to bring the search close to previously visited solutions.

Intensification of the search around best solutions discovered during the search have also often been considered. For instance, in [101] the SA algorithm is forced to start from the current best solution several times, while the ACO in [52] uses global pheromone updating, to further explore search areas around the best solutions obtained. Also, in [143] intensifying the search is done by selecting the best routes and re-ordering customers within these routes, using Solomon's I1 insertion.

A post-optimization phase is sometimes added to the basic meta-heuristic to further enhance best solutions. For example, the approach in [129] performs a post-optimization

using an exact set-partitioning algorithm on routes obtained from the adaptive memory. Also, in [143] the post-optimization technique of [56] is applied to each route of the final solution obtained.

In many cases, hybridization of more than one meta-heuristic technique is adopted. For example, in [101] a tabu-embedded SA is used. Also, in [25] the annealing process is enhanced using a varying size tabu-list, and the authors in [11] use both SA and LNS.

To complete our review of meta-heuristic techniques for the VRPTW, Table 3.2 summarizes some Genetic Algorithm (GA) approaches that have been applied to this problem. Almost all of these techniques hybridize GAs with some other heuristic, local search, or another meta-heuristic approach for solution construction and/or solution improvement.

Solution representation is usually based on *integer representation*, rather than the classical bit string encoding. Thus, genetic operators are directly applied to problem solutions. The initial population is often created randomly, or by adapting some well known construction methods. For example, in [144], customers are grouped randomly, and then the cheapest insertion method of [65] is used for routing each group of customers.

The fitness function is usually based on solution cost, i.e., number of routes, total distance and total duration. However, since some approaches allow violations of problem constraints, the fitness function should take this into account. For example, in [16], the fitness considers the number of un-serviced customers in an infeasible solution. Also, in [15] a penalty is added in the objective function to violated constraints. On the other hand, the selection scheme often applied is roulette-wheel selection. However, other selection methods have also been used. For instance, a ranking scheme is used in [120].

Regarding recombination, most of the GA techniques apply specialized crossover operators, since traditional genetic operators do not usually fit the VRPTW. For example, the authors in [120] propose a Sequence-Based Crossover (SBX) and a Route-Based Crossover (RBX), while the GA in [16] applies special crossover operators that depend on a global precedence relationship among genes, such as the distance or time windows ordering. On the other hand, mutation is often based on traditional local search operators like 2-Opt, 2-Opt*, Or-Opt and LNS. In addition, mutation is sometimes applied to reduce the number of routes in the solution, as in [120] and [15].

To sum up, meta-heuristic approaches to the VRPTW often embed solution construction and improvement methods as described previously. However, they usually obtain better quality solutions than algorithms that apply straightforward solution construction and improvement techniques on their own. This often comes at the expense of greater complexity in implementation and increased computation time, though. Bräysy and Gendreau

Table 3.1: Meta-heuristics for the VRPTW.

Paper	Algorithm	Solution Construction	Solution Improvement	Remarks
Garcia *et al.* [53]	TS	Solomon's insertion	2-Opt*, Or-Opt	Parallel implementation - Only allows moves involving arcs close in distance
Rochat and Taillard [129]	TS	Solomon's insertion & 2-Opt	2-Opt, relocate	Uses adaptive memory containing routes obtained from best solutions visited during the search, with the purpose of providing new starting solutions
Taillard *et al.* [143]	TS	Solomon's insertion	CROSS	Decomposes solutions into subsets of routes, based on a polar angle associated with the centre of gravity of each route - TS applied to each subset separately - Diversification by penalizing frequently performed exchanges - Intensification by reordering customers within best routes using I1 insertion
Lau *et al.* [99]	TS	Relocate from a holding list where all customers are initially stored	Exchange - relocate	Allows violation of constraints for a penalty in the objective function - Penalty parameters adjusted dynamically
Chiang and Russell [25]	SA	The parallel construction of Russell [133]	λ-interchange and k-node interchange	Enhancing of the annealing process via a varying size tabu-list
Li *et al.* [101]	Tabu-embedded SA	Insertion and extended sweep heuristics of Solomon	Shifting and exchange of customer segments between and within routes	SA restarts from current best solution several times - Reduce routes by reordering customers and inserting them into other routes - Diversification by random shifts and exchange of customer segments
Bent and Van Hentenryck [11]	SA and LNS	Not specified	2-Opt, Or-Opt, relocate, exchange and 2-Opt*	2-phase approach: SA to minimize number of routes and LNS to minimize total distance
Gambardella *et al.* [52]	ACO	Nearest-neighbour heuristic with probabilistic rules	CROSS	2 colonies: first one minimizes number of vehicles, and the second minimizes total distance - Both cooperate in updating best solution
Bräysy [19]	VNS	Solomon's Insertion and the parallel construction of Russell [133]	Or-Opt exchanges and CROSS-exchanges	Reducing number of routes using an ejection-chain phase - Second VNS phase for improving total distance
Bräysy *et al.* [22]	Threshold accepting (TA) & Guided Local Search (GLS)	Savings heuristic	SPLIT (of routes) - limited CROSS - limited Or-Opt	Construction algorithm only considers selected routes and selected customers for merging - Initial phase to minimize number of routes, based on simple customer re-insertion - Further improvement using TA & GLS - TA allows local search moves that worsen the objective function, provided it is within a certain threshold limit - GLS penalizes certain solution based on some solution features (e.g. long edges), not considered part of a near-optimal solution

Table 3.2: Genetic Algorithms for the VRPTW.

Paper	Initial Population	Crossover	Mutation	Remarks
Thangiah [144]	A set of K seed angles are planted, where K is the number of vehicle. Sector rays are drawn originating from the depot to each seed angle - Routing done using cheapest insertion of Golden and Stewart [65]	Exchange a randomly selected portion of the bit string between chromosomes	Randomly change bit value	Cluster first route second approach called GIDEON - Each chromosome is a set of possible clustering scheme and the GA used to improve the clustering - Routes improved by λ-interchange
Potvin and Bengio [120]	Chromosome is a problem solution created using Solomon's cheapest insertion	Sequence-Based Crossover (SBX): two sub-tours from two parents are linked together - Route-Based Crossover (RBX) replaces a route of one parent by another route from the second parent - Repair operator applied for infeasible offspring	Reduce number of routes by inserting customers from a randomly selected short route into another - Local search mutation using Or-Opt exchanges	The approach is named GENEROUS
Blanton and Wainwright [16]	Solution coded as a permutation of customers - Insertion heuristic used to construct solution	Specialized crossover that uses global precedence relationship among customers (TW/distance)	Randomly exchange 2-edges	The GA searches for a good ordering of customers. while construction of feasible solutions is handled by a greedy heuristic
Berger *et al.* [15]	Random insertion heuristic - Chromosomes are problem solutions - Two populations: POP1 used to minimize distance and POP2 minimizes constraint violations	Insertion-based crossover (IBX): combining k routes from parent1 (one at a time) with subsets of customers from parent2 - Insert-within-route crossover (IRNX): simultaneously combines k routes from parent1 with subsets of customers from parent2	Large Neighbourhood Search (LNS) mutation - Each customer considered for reinsertion in an alternate route. eliminate shortest route - Reorder customers within a route	Simultaneous evolution of two populations (minimize distance & constraint violations) - POP2 has smaller number of routes than POP1 - When a new better solution is obtained from POP2, POP1 is replaced by POP2
Repoussis *et al.* [128]	Discrete arc-based representation of individuals combined with a binary vector of strategy parameters - Individuals created using greedy insertion from a Restricted Candidate List (RCL), with a probabilistic element	Multiple-parent recombination applied only to strategy parameters not to problem solutions	Mutation rate determined by strategy parameters evolved during the search - Mutation done using "ruin and recreate"	$(\lambda + \mu)$ Evolution Strategy (ES). Starting from μ individuals, at each iteration a new intermediate population of λ individuals is produced by mutation - Evolutionary search is based only on mutation - Each offspring is further optimized using route elimination and a local search (based on TS) to improve total distance

in [21] identified several techniques that seem to give good results when applied within the different meta-heuristic approaches. Some of these techniques are: 1) saving best solutions found during the search, 2) varying the neighbourhood structure, with the size of the neighbourhood being typically small, 3) using a memory structure to facilitate the search, 4) applying a specific strategy for reducing the number of routes in the solution and 5) hybridization of different heuristics and meta-heuristics.

3.6 Chapter Summary

Vehicle routing problems have gained considerable attention in the last few decades. This is mainly due to the increase in complexity in transportation and logistics demands, and the urgent need for optimizing client services, reducing operational costs, and limiting the negative environmental impacts that may result from the non-optimal planning of vehicles and their routes. This chapter provided a general classification of vehicle routing problems existing in the literature and highlighted the diversity of problem types, solution methods, and the huge amount of published research that deals with this problem.

We emphasized in this chapter one particular vehicle routing problem that is of interest to our study, the Vehicle Routing Problem with Time Windows (VRPTW). This problem is closely related to pickup and delivery problems that we investigated in our research, in terms of: problem definition, solution methods, objective function and constraint handling approaches. We presented in this chapter a summary of some important heuristic and meta-heuristic algorithms that have been applied to solving the VRPTW.

The next chapter is devoted to an overview of pickup and delivery problems, which are the main focus of our research. We will present a general classification and highlight some published research in the field, as an introduction to a more in-depth analysis of our selected pickup and delivery problems, which will follow during the course of this thesis.

Chapter 4

Pickup and Delivery Problems: A Literature Review

We dedicate this chapter to a literature summary of Pickup and Delivery (PD) problems, since they are the main theme of our research. In addition, there are various types of pickup and delivery problems and different classifications of these problems that distinguish them from other variants of vehicle routing problems.

PD problems are widely applicable in areas such as: the transportation of raw materials from suppliers to factories, Internet-based pickup from sellers and delivery to buyers, food and beverage collection and delivery, post and parcel delivery, newspaper distribution, and airline and bus scheduling. In addition, an important related variant is the dial-a-ride problem and the handicapped persons' transportation, where people instead of goods are transported, giving rise to customer inconvenience issues that should be taken into consideration while constructing a solution.

Similar to other vehicle routing problems, pickup and delivery problems have attracted the attention of researchers in the last few decades. Nevertheless, published research tackling these problems is still relatively limited, compared to other variants of vehicle routing problems [137]. Several important surveys of pickup and delivery problems have appeared in the literature. The oldest ones are by Solomon and Desrosiers [142], and Savelsbergh and Sol [137]. Two other surveys appeared recently in [14] and [117]. A survey dedicated to the dial-a-ride problem is in [29]. Another survey on the one-to-one pickup an delivery problems (see Section 4.2 for a description of this problem category) is provided in [31]. In our literature summary, we mostly rely on the information and classification provided by Parragh *et al.* in 2008 [117], unless otherwise indicated.

Parragh *et al.* [117] classify the different types of pickup and delivery problems under two classes: **1)** problems that deal with the transportation of goods from pickup customers (also called *backhaul* customers) to the depot, and from the depot to delivery customers (also called *linehaul* customers). This class is called the **Vehicle Routing Problem with**

Backhauls (VRPB), and **2)** problems that deal with the transportation of goods/people between two pickup and delivery points. This class is called the **Vehicle Routing Problem with Pickup and Delivery (VRPPD)**. Sections 4.1 and 4.2 describe in detail these two categories. Section 4.3 briefly highlights some meta-heuristic techniques that have been applied to pickup and delivery problems. Finally, Section 4.4 concludes this chapter and introduces the rest of the thesis.

4.1 The Vehicle Routing Problem with Backhauls (VRPB)

As mentioned above, this class deals with PD problems that involve the transportation of goods from the depot to customers and vice versa. These problems are further subdivided into four categories, generally based on the required visiting order of pickups and deliveries, and whether a customer can demand both types of services. In all cases the objective is to minimize the total cost (e.g. total travel distance) of the routing plan, while adhering to some pre-specified problem constraints. For example, respecting the capacity of the vehicle at all times, and ensuring that the vehicle starts, if needed, with a load that is equal to the total load to be delivered.

1. **The vehicle Routing Problem with Clustered Backhauls (VRPCB)**: where all delivery customers must be visited before all pickup customers. The single vehicle special case of the same problem is denoted by **(TSPCB)**.

2. **The Vehicle Routing Problem with Mixed Linehauls and Backhauls (VRPMB):** where mixed visiting of pickup and delivery customers are allowed. The single vehicle case of the same problem is denoted by **(TSPMB)**[1]. If the capacity of the vehicle is greater than or equal to the total sum of demands of pickup and delivery customers, the problem coincides with a simple TSP.

3. **The Vehicle Routing Problem with Divisible Delivery and Pickup (VRPDDP):** where each customer is both a pickup and a delivery customer. In addition, two visits to the same customer is allowed. The single vehicle variant of the same problem is denoted by **(TSPDDP)**. A possible scenario in this variant is that the vehicle could initially visit a few delivery customers to get rid of some of the load it is carrying. Afterwards, the vehicle could visit another set of customers to perform both

[1] More details about this problem will be presented in Chapter 9, since this problem is closely related to the One-commodity Pickup and Delivery Problem (1-PDP) that we handled in our research. Nevertheless, we refer to it in our research by the Traveling Salesman Problem with Pickup and Delivery (TSPPD), the name most commonly used for this problem in the literature (e.g. [110] and [75]).

delivery and collection service at each, and finally a collection service would be performed at customers that were initially visited for delivery. This type of solution is called a 'lasso' solution. It is also observed in [117] that a solution method of the VRPMB (the second category) can be applied to the VRPDDP if each customer demanding both delivery and pickup was modeled by two vertices, one for the delivery and another for the pickup. Very little research has been applied to this problem.

4. **The Vehicle Routing Problem with Simultaneous Delivery and Pickup (VRPSDP):** where each customer demands both a delivery and a pickup, but, unlike the VRPDDP (the third category), only *one* visit is allowed for each customer. The single vehicle variant is denoted by **(TSPSDP)**.

We summarize in Table 4.1 the most important features of the different variants of the VRPB described in this section. In this table, we indicate the problem name, whether the customer is allowed to demand both pickup and delivery, whether delivery customers must all be visited before pickup customers, or otherwise that mixed sequences of visits is allowed. The table also shows whether more than one visit is permitted, and if splitting the demand of one customer is allowed. Finally, the last column of the table gives examples of selected published papers that dealt with the underlying problem, together with the solution method adopted. We used the notations: (LNS) for Large Neighbourhood Search, (TS) for Tabu Search, and (VNS) for Variable Neighbourhood Search. If the paper handled the single vehicle case, we indicate this by (SV) and if it handled the multiple vehicle case, we indicated this by (MV).

The authors in [117] concluded from their survey that the LNS algorithm of [132] seems to be the most flexible and accurate meta-heuristic to date, since it can be applied to several problems in the VRPB class, and it provided new best solutions for several benchmark instances. On the other hand, they selected the heuristic in [111] among the fastest algorithms applied to the VRPB.

Table 4.1: The Vehicle Routing Problem with Backhauls (VRPB).

Problem	Customer both P&D	Deliveries Visited before Pickups	Mixed Visiting	One Customer Visit	Demand Splitting	Example Papers and Solution Methods
VRPCB TSPCB	no	yes	no	yes	no	Gendreau *et al.* [57]- heuristic (SV) Thangiah *et al.* [145]- heuristic (MV) Mladenović and Hansen [108]- VNS (SV) Ropke and Pisinger [132]- LNS (MV)
VRPMB TSPMB	no	no	yes	yes	no	Mosheiov [110]- heuristic (SV) Nagy and Salhi [111]- heuristic (MV) Ropke and Pisinger [132]- LNS (MV)
VRPDDP TSPDDP	yes	no	yes	no (at most two)	yes	Hoff and Løkketangen [77]- TS (SV) Salhi and Nagy [134] - heuristic (MV)
VRPSDP TSPSDP	yes	no	yes	yes	no	Alshamrani *et al.* [4]- heuristic (SV) Nagy and Salhi [111]- heuristic (MV) Ropke and Pisinger [132]- LNS (MV)

4.2 The Vehicle Routing Problem with Pickups and Deliveries (VRPPD)

This class refers to problems that deal with the transportation of goods/people between a pickup and a delivery point. Two subclasses can be distinguished in this category:

1. **The Pickup and Delivery Vehicle Routing Problem (PDVRP):** where pickup and delivery points are unpaired, and one type of commodity is transferred. Thus, pickup customers will supply the demands needed by the delivery customers. The single vehicle variant of this problem is denoted by **(PDTSP)**. The objective function in this problem category is to minimize the total cost of the route(s), without violating the capacity constraints. It should be noted that the PDTSP is the same as the **1-PDP** investigated in our research, as explained in detail in Chapters 9 and 10.

2. **The Classical Pickup and Delivery Problem (PDP) and the Dial-a-Ride Problem (DARP):** where a transportation request is associated with both a pickup (origin) and a delivery (destination), i.e., customers are paired and the demand of the pickup is the same as the demand of its associated delivery. Unlike the PDP, the DARP additionally takes customer inconvenience into account. The single vehicle cases of these problems are denoted by **SPDP** and **SDARP** respectively. The objective function for these problems usually tries to minimize the number of vehicles used (in the multiple-vehicle case) as a primary objective, followed by reducing the overall travel distance or service duration. The final solution should also adhere to all underlying problem constraints, such as the vehicle capacity and the visiting time restrictions. Our research handles the PDP problem with the addition of the time window constraint (denoted by **PDPTW**) in Chapters 5 and 6 for its single vehicle variant (denoted in our research by **SV-PDPTW**), and Chapters 7 and 8 for its multiple vehicle variant (denoted in our research by **MV-PDPTW**).

We provide in Table 4.2 a summary of the most important features pertaining to the VRPPD. We indicate in this table the subcategory name, whether the customers are paired or not, whether a homogeneous commodity or multiple commodities are considered, and whether the problem is primarily associated with the transportation of people or goods. In addition, we highlight the most important constraints that are usually taken into consideration while handling the problem. These include: 1) the precedence constraint, in which a pickup request is required to precede its corresponding delivery, 2) the coupling constraint, which requires that the pickup and delivery pair must be visited by the same vehicle (in case of the multiple-vehicle variant), 3) the capacity constraint, which ensures that the load carried by the vehicle at any given time does not exceed its capacity, 4) the time window constraint, which requires that a customer has to be served in a pre-specified time interval, and 5) the maximum ride time constraint, which ensures that when people are transported, the time they spend in the vehicle does not far exceed the direct travel time between their origin and destination. Finally, the table gives examples of some published papers that deal with the problem. Again, we use (SV) for the single vehicle case and (MV) for the multiple-vehicle variant. (LNS) refers to a Large Neighbourhood Search algorithm, and (ALNS) refers to an *Adaptive* version of LNS, (HC) is a Hill Climbing approach, (GA) is a Genetic Algorithm, (GGA) is a Grouping Genetic Algorithm, (SA) is a Simulated Annealing algorithm, and (TS) is a Tabu Search algorithm.

In our opinion, the GA in [158], the LNS in [12], the ALNS in [131], and the GGA in [126] seem to be among the state-of-the-art algorithms applied to the VRPPD, in terms of their solution quality and/or robustness or flexibility. More details about these approaches and other important research in the field will be described in the following chapters, where we address specific pickup and delivery problems in our research. Before we conclude our review of pickup and delivery problems, though, we present in the next section a brief summary of some meta-heuristic techniques that have been applied to this problem class.

Table 4.2: The Vehicle Routing Problem with Pickups and Deliveries (VRPPD).

Problem	Paired Customers	One Commodity	Type of Commodity	Precedence	Coupling	Capacity	TW	Max Ride Time	Example Papers and Solution Methods
PDVRP PDTSP	no	yes	goods	no	no	yes	no	no	Hernández-Pérez and Salazar-González [75]- heuristic (SV) Zhao *et al.* [158] - GA (SV) Dror *et al.* [45]- heuristic (MV)
PDP SPDP	yes	no	goods	yes	yes	yes	yes	no	Jih and Hsu [89]- GA (SV) Hosny and Mumford [84] - HC & SA & GA (SV) Li and Lim [100]- TS (MV) Bent and Van Hentenryck [12] - LNS (MV) Ropke and Pisinger [131] - ALNS (MV) Hosny and Mumford [80]- GA (MV)
DARP SDARP	yes	yes	people	yes	yes	yes	yes	yes	Psaraftis [124]- heuristic (SV) Jaw *et al.* [87]- heuristic (MV) Baugh *et al.* [9]- SA (MV) Rekiek *et al.* [126]- GGA (MV) Jørgensen *et al.* [91]- GA (MV)

4.3 Meta-heuristic Algorithms for Pickup and Delivery Problems

Similar to all routing and scheduling problems, pickup and delivery problems have attracted the interest of researchers who deal with meta-heuristic techniques, since exact algorithms can only be used for small problem sizes and may not be useful for large practical applications. For example, the approach in [46], for solving the multiple-vehicle PDPTW, was only capable of solving problem sizes of up to 55 customers using an exact algorithm. Also, the exact algorithm in [74], solved PDTSP instances of up to 60 customers. On the other hand, heuristic and meta-heuristic approaches are able to solve problems having several hundreds of nodes, which is common in every day routing and scheduling demands for some industrial applications. For example, Paragon software systems recently reported, in their October 2009 newsletter[2], that their optimization software helped Sainsbury's to manage its transport requirement, involving daily deliveries from 19 distribution centers to 527 supermarkets and 276 smaller stores.

Many heuristic and meta-heuristic techniques that have been successfully applied to the VRP have been adapted to fit the PDP variant. Table 4.3 summarizes some meta-heuristic techniques for pickup and delivery problems. In addition to the notations used for the meta-heuristic techniques in Table 4.2, we use (GRASP) for a Greedy Randomized Adaptive Search Procedure, and (VND) for a Variable Neighbourhood Descent algorithm. On the other hand, Table 4.4 summarizes some GA research on pickup and delivery problems. It is important to emphasize again that the review presented here is just intended as a quick introduction to the literature that deals with pickup and delivery problems. As previously mentioned, further investigation of specific work that is closely connected to our research will be presented in later chapters, when we address certain variants of pickup and delivery problems.

[2] http://www.paragonrouting.com/cms/assets/pdf/directionsissue22.pdf

Table 4.3: Meta-heuristics for Pickup and Delivery Problems.

Paper	Problem	Algorithm	Solution Construction	Solution Improvement	Remarks
Hernández-Pérez and Salazar-González [73]	PDTSP (SV)	Hybrid GRASP & VND	Selecting the next element for insertion from a Restricted Candidate List (RCL), with a probabilistic element in the choice of the next node	2-Opt and 3-Opt in the VND phase - Move forward and move backward in a post optimization phase	Combining two optimization heuristics: 1) GRASP (Greedy Randomized Adaptive Search Procedure), which is based on a repetition of a construction phase and a local search phase, 2) VND (Variable Neighbourhood Descent), which is a variant of the VNS, where the local optimum found acts as the new starting point for the local search
Landrieu *et al.* [95]	PDPTW (SV)	TS	Simple insertion heuristic	A swap move and an insertion move	Two tabu search methods compared: a regular deterministic Tabu Search (TS), and a Probabilistic Tabu Search (PTS)
Nanry and Barns [112]	PDPTW (MV)	Reactive TS	Cheapest insertion	Single Paired Insertion (SPI): move all predecessor nodes to better feasible locations, then insert successor nodes in the best possible positions - Swapping Pairs Between Routes (SBR): exchange predecessor nodes between vehicles then successor nodes - Within Route Insertion (WRI): re-order nodes within the same route	Reactive tabu search - Allows tuning of search parameters, such as a short-term memory length, based on an assessment of visited solutions during the search
Li and Lim [100]	PDPTW (MV)	Tabu-embedded SA	Modified Solomon's insertion heuristic - Initializes each route with a P&D pair that satisfies a set of criteria, based on combined TW intervals and distance from the depot	PD-Shift: moves a P&D pair from route1 to route2, and moves another pair from route2 to route1 - PD-Swap: removes a P&D pair from each route, and then reinserts each pair in the other route - PD-Rearrange: removes and then reinserts a P&D pair in the same route	Main meta-heuristic is a tabu-embedded SA with K-restarts, i.e., the algorithm stops when the number of iterations without improvement reaches a pre-defined value K - To prevent cycling, the SA records the accepted solutions in a tabu list
Urban [149]	PDPTW (MV)	Guided SA	Assigning only one pickup and delivery pair to each route. A guided SA is then applied to improve this initial solution by trying to bundle requests into the available routes	Removing a selected request from its current route and inserting it into another - Exchange of requests served on different routes - Repositioning of a pickup and delivery pair in its same route	The cost function takes into consideration both wage and non-wage related traveling and operating costs of the vehicles - The selection of neighbourhood move depends on a certain probabilistic criterion - Calculated measurements control the selection of customers and routes during the generation of the new solution
Cordeau and Laporte [30]	DARP (MV)	TS	Assigning requests to vehicles randomly and inserting source location followed by destination at the end of the route	Moving a request from one route to another	Re-insertion of the removed request in its original route is forbidden by for a number of iterations - Diversification penalizes frequently occurring moves in the objective function - Intensification performed by removing all requests from their respective routes, and re-inserting them in the best possible positions

Table 4.4: Genetic Algorithms for Pickup and Delivery Problems.

Paper	Problem	Initial Population	Crossover	Mutation	Remarks
Zhao *et al.* [158]	PDTSP (SV)	Nearest-neighbour construction heuristic	A new *pheromone-based* crossover - Edges between last node and potential new node that have proved successful in the past are favoured	3-exchange procedure	Initial population is optimized using a 2-Opt move - Pheromone trails are updated each generation - Offspring is optimized using a 2-Opt local search.
Jih and Hsu [89]	PDPTW (SV)	A solution is encoded as a permutation of locations	Two traditional order-based crossover operators - Two merge crossover operators, MX1 and MX2, that use a global precedence vector to guide the inheritance process, as in [16]	Swap two genes selected at random - Select two random sites in the chromosome and reverses the sub-route between them - Shuffle genes that precede a location for which a violation of constraints is observed	A Family Competition Genetic Algorithm (FCGA) - Each individual of the population plays the role of a family father in turn - Another randomly selected individual plays the role of mate for the family father
Jung and Haghani [92]	PDPTW (MV)	Chromosome representation is based on assigning a 4 digit code to each request - First digit represents the number of the vehicle - Remaining 3 digits are used to sort requests according to the visiting order of the vehicle - A pickup request always has a sorting code that is less than its corresponding delivery	Two crossover points generated, and the resulting segments swapped between the two parents	Change the first digit of a pickup and delivery pair to another digit, i.e., assign the request to another vehicle - Vehicle merging also used to reduce the number of vehicles	Customers inconvenience is taken into consideration by assigning a penalty in the objective function to the arrival of the vehicle before or after the specified TW
Pankratz [116]	PDPTW (MV)	Each gene represents a group of requests that are assigned to one vehicle - Routes are represented by an independent data structure associated with each gene	Vehicles with their assigned requests removed from one parent and inserted in the other parent	Removes a random cluster from a chromosome, and reassigns its requests to other clusters	A Grouping Genetic Algorithm (GGA), where the GA is only responsible for the grouping (assignment) problem - Routing is handled using an adaptation of Solomon's cheapest insertion - Chromosome 'clean up' needed after the genetic operators to remove duplicate vehicles and repeated assignment of requests, in addition to re-assigning requests that are no longer assigned
Rekiek *et al.* [126]	DARP (MV)	Chromosome representation is group oriented, where each group represents a vehicle to which a set of requests is assigned	traditional GGA crossover as in [49]	Create a new group - Remove an existing group - Move items between groups - An inversion operator is also used to change the location of groups in the chromosome	A Grouping Genetic Algorithm (GGA) applied to the Handicapped Person Transportation (HPT) problem, which focuses on minimizing clients' inconvenience - The insertion heuristic used to insert requests in routes is the best-fit (BF) heuristic - An unserved client is inserted in the vehicle that will grant him the best service without violating problem constraints

4.4 Chapter Summary

In this chapter, we reviewed an important category in the VRP literature, which deals with pickup and delivery problems. These problems are gaining more attention every day, due the continuous need for optimizing routing and scheduling costs in real-life applications that require two different service types for clients. We briefly analyzed basic problem features and general existing classifications. We also highlighted some published research in the field, mainly focusing on heuristic and meta-heuristic approaches.

In the following chapters, we will start discussing the specific problems that we addressed in our research. The Pickup and Delivery Problem with Time Windows (PDPTW) will be analyzed in detail in Chapters 5 and 6 for its single vehicle variant (SV-PDPTW), and in Chapters 7 and 8 for its multiple vehicle variant (MV-PDPTW). On the other hand, the One-commodity Pickup and Delivery problem (1-PDP) will be thoroughly discussed in Chapters 9 and 10.

Chapter 5

The Single Vehicle Pickup and Delivery Problem with Time Windows: Introduction and a Genetic Algorithm Approach

The Single Vehicle Pickup and Delivery Problem with Time Windows (SV-PDPTW) is a frequently encountered problem in public and goods transport systems. However, only a few researchers seem to have tackled it, possibly due to the difficulty in managing the different underlying problem constraints. As previously mentioned in the introduction to this thesis (Chapter 1), our research concentrates on representational issues and neighbourhood moves within a simple meta-heuristic framework. For this problem, we adopt a solution representation that depends on a duplicate code for both the pickup location and its corresponding delivery. This simple representation will guarantee the satisfaction of the precedence constraint, among the pickup and delivery pair, throughout the search. We also present intelligent neighbourhood moves, that are guided by the time window, aiming to overcome the difficult timing constraint efficiently and produce feasible and good quality solutions in a reasonable amount of time.

This chapter covers two aspects: 1) a detailed analysis of the problem and research challenges and motivations, as well as a summary of some related work on the SV-PDPTW and 2) the initial development of our key ideas using a Genetic Algorithm (GA) as a candidate solution methodology. The next chapter presents a comparison of three different approaches to the SV-PDPTW: a genetic algorithm approach, a simulated annealing approach, and a simple hill climbing heuristic, which all employ the same representation and neighbourhood move that our initial GA approach adopts. The work related to the current chapter, i.e., the initial GA technique, was presented in the *GECCO'07* conference, as a late breaking paper [79]. Also, the research tackling other heuristics and meta-heuristics applied to the SV-PDPTW, as described in the next chapter, was published in the *Journal*

of heuristics [84].

The rest of this chapter is organized as follows: Section 5.1 explains the problem tackled in this part of the research and the motivation behind our selection of the problem and the underlying solution approaches. Then, Section 5.2 formally defines the SV-PDPTW, while Section 5.3 provides a brief summary of some related work from the literature. Section 5.4 highlights the contribution of our research, and focuses on the solution representation and neighbourhood move suggested to facilitate dealing with the SV-PDPTW. Sections 5.5 presents the details of the proposed GA approach to solving this problem, together with the experimental findings of this part of our research. Finally, Section 5.6 concludes the chapter with our intended future work.

5.1 Problem and Motivation

The Pickup and Delivery Problem with Time Windows (PDPTW) is an important variant of vehicle routing problems that is likely to assume even greater prominence in the future. Current concerns over global warming, resource depletion and the social impact of traffic congestion and pollution, are driving companies, government organizations and researchers to improve the efficiency of logistics and distribution operations. In addition, the rapid growth in parcel transportation as a result of e-commerce is likely to have an increasing impact. As previously mentioned in Chapter 4, an important related variant of the PDPTW is the dial-a-ride problem, which is concerned with the transportation of people, especially the elderly and the disabled, from their origins to their destinations, while minimizing customer inconvenience.

The SV-PDPTW is a special case of the PDPTW dealing with a number of customer requests that must be satisfied by *one* vehicle with a known capacity. The route of the vehicle usually starts and ends at a central depot. A request must be collected from a pickup location before being dropped off at a corresponding delivery location, and every pickup and delivery location is associated with a specific time window during which it must be served. If the vehicle arrives earlier than the beginning of the designated time window interval, it must wait until the requested service time begins. All requests must be served in a way that minimizes the total travel cost of the vehicle, without violating precedence, capacity and time windows constraints [137].

In addition to being a sub-problem in the PDPTW, the SV-PDPTW has also practical applications, where small scale companies or individuals may operate one vehicle to serve a set of clients, as for example in a dial-a-ride service. In other applications, the underlying

vehicle could be a helicopter or a small ship. Also, in some multiple vehicle applications, the assignment of requests to vehicles may be restricted by some commodity, vehicle, driver or client conditions. In such cases, certain requests must be assigned to a specific vehicle, making the optimization of a single vehicle route necessary for reducing the overall cost of the logistic operation. Moreover, the analysis of the SV-PDPTW is essential in developing an insight into the more general multiple-vehicle case, and establishing comprehensive solution algorithms.

As a constrained version of the TSP, the SV-PDPTW is known to be $\mathcal{NP}$*-hard* [95], with the presence of time windows making the problem particularly complicated. Since exact algorithms are too slow for large problem sizes, heuristic and meta-heuristic approaches seem to be reasonable alternatives. Several heuristics and meta-heuristics are potentially suitable for this problem. Genetic Algorithms (GAs) are known for their robustness, parallelism, and their ability to perform reasonably well on a wide variety of problems, including ordering and grouping problems, as well as highly constrained problems [64] [105]. Thus, exploring GAs to solve the SV-PDPTW would seem to be a justified option.

Another alternative meta-heuristic approach that could be suitable for this problem is Simulated Annealing (SA), which is analogous to the annealing of solids. As previously discussed in Chapter 2, this approach has been widely applied to many optimization problems, successfully transforming 'random' low quality solutions to stable high quality optimized solutions. One appealing feature of simulated annealing is that it is very easy to implement, since it only requires a method for generating a move in the neighbourhood of the current solution, and an appropriate annealing schedule. A third solution alternative is simple Hill Climbing (HC) that first creates a candidate solution and then iteratively tries to perturb this solution to improve it.

In our research, we mainly focus on developing a meta-heuristic framework for the SV-PDPTW rather than on a specific algorithmic paradigm. Within the context of the meta-heuristic, we employ a problem-specific solution representation and guided neighbourhood moves, aiming to overcome and deal efficiently with the hard problem constraints. Our key ideas were adopted in all three approaches highlighted above, the GA, SA and HC. In all cases, our view was to design an intelligent and robust solution approach that can handle all problem constraints efficiently, while keeping the overall algorithm as simple as possible, a feature often overlooked in most up to date solution methodologies. Ultimately, we aim to obtain feasible and high quality solutions in an acceptable amount of time.

5.2 The SV-PDPTW

Let $G = (N, A)$ be a *digraph*. The node set is $N = \{n_i \in N | i = 0, 1, 2, ..., m\}$, such that m is an even index. The node n_0 denotes the depot, and each $n_i, i = 1, 2, ...m$ denotes a customer location. It is assumed that a customer's request $r_k, k = \{1, 2, ...m/2\}$ consists of a pickup and delivery (P&D) pair. We can consider, without loss of generality, that the set $N^+ = \{n_i \in N | i = 1, 2, ..., m/2\}$ represents pickup locations, and the set $N^- = \{n_i \in N | i = (m/2) + 1, ..., m\}$ represents delivery locations, such that the pickup location n_i has the corresponding delivery location $n_{i+(m/2)}$. Thus, $N = N^+ \cup N^-$ and $|N^+| = |N^-| = m/2$. Each location n_i is associated with:

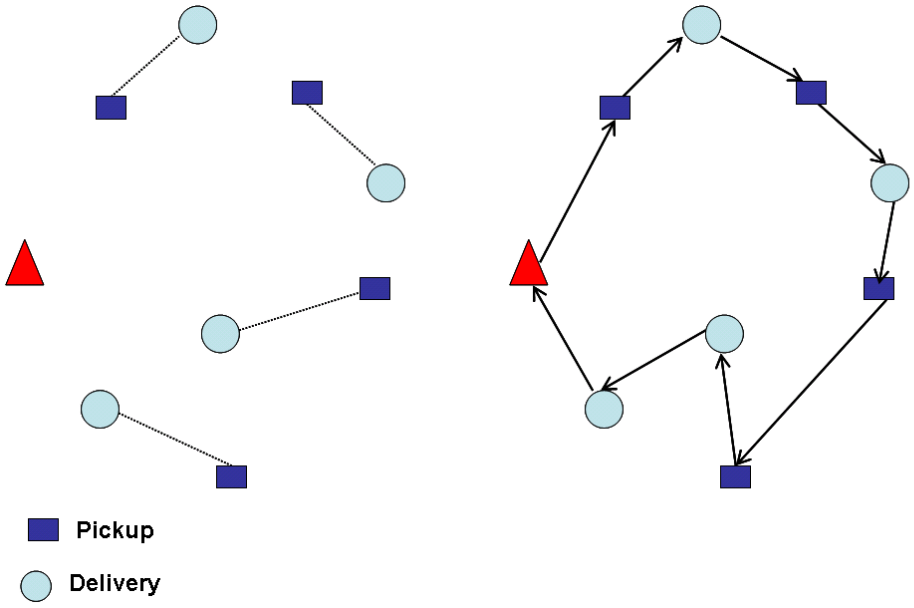
- A customer demand q_i, such that $q_i > 0$ for a pickup location, $q_i < 0$ for a delivery location and $q_i + q_j = 0$ for the same customer's pickup and delivery locations ($q_0 = 0$);

- A service time s_i ($s_0 = 0$), which is the time needed to load or unload a customer demand;

- A Time Window (TW) $[e_i, l_i]$ during which the location must be served, and $l_i \geq e_i$.

For each pair of nodes $< n_i, n_j >$ a travel time t_{ij} and/or a travel distance d_{ij} are specified. We assume here a symmetric case, i.e., $t_{ij} = t_{ji}$ and $d_{ij} = d_{ji}$. In addition, only edges satisfying the TW are allowed. Thus the arc set is $A = \{< n_i, n_j > | n_i, n_j \in N, n_i \neq n_j, t_{0i} + s_i + t_{ij} < l_j\}$. The vehicle has a limited capacity C. The *capacity constraint* ensures that the total load carried by the vehicle at any given time does not exceed its capacity. The vehicle's journey should start and end at the depot, while each location is to be visited exactly once. The *time window constraint* requires that a location must be serviced within the specified TW, i.e., if the vehicle reaches the location before the earliest service time e_i, it must wait until e_i. The *precedence constraint* requires that each pickup location must precede the corresponding delivery location in the visiting order.

The objective function varies depending on the application. In general, one or more of the following objectives are minimized as far as possible: the total traveling distance, the total route duration, or the drivers' total waiting time. Figure 5.1 shows a *simplified* representation of a small instance of this problem before and after solving it.



(a) SV-PDPTW - before solution (b) SV-PDPTW - after solution

Figure 5.1: The single vehicle pickup and delivery problem with time windows.

5.3 Related Work

As indicated above, there are various approaches to handle pickup and delivery problems. Some approaches are *exact* and guarantee to solve the problem to optimality, while others are *approximations* and attempt to find an acceptable solution in a reasonable amount of time. For the SV-PDPTW, an exact algorithm is the dynamic programming approach of [124]. However, this technique has a time complexity of $O(n^2 3^n)$ (where n is the number of locations) and for this reason is normally limited to solving small problems of up to about 10 requests (20 locations). Also, the authors in [38] are able to provide exact solutions to the single vehicle dial-a-ride problem, with precedence, capacity and time windows constraints, using dynamic programming. Their algorithm uses a set of states (S, i), such that S is a subset of nodes from the node set, and i is a selected node. State (S, i) is defined only if there is a feasible route that visits all nodes in S and terminates at i. To reduce the state space, their dynamic programming algorithm uses sophisticated state elimination criteria, based on the state (S, i). Computational experimentation indicated that the elimination criteria work best when the time windows are tight and the vehicle capacity is small, such that narrowing the constraints in their approach helped to reduce the usual exponential running time of dynamic programming to a linear running time. This algorithm can solve instances of up to 40 requests (80 locations).

Approximation algorithms, in which a heuristic or a meta-heuristic is developed to deal

with the problem, make it possible to cope with much larger instances. Among the heuristics for solving the SV-PDPTW is the one suggested in [23]. This heuristic involves a 2-phase algorithm. In the initial phase a feasible solution is constructed, and in the second phase this solution is improved. In both phases a variable depth arc exchange procedure is performed, as a neighbourhood move, in which the number of arcs to be exchanged is not determined in advance, but calculated dynamically during the search. In the route construction phase, the time constraint may be violated as long as the precedence and capacity constraints are satisfied. In the improvement phase, however, only feasible solutions are permitted and route duration is used as an objective function. During the improvement phase, the feasibility of a tour (in terms of precedence and capacity) is verified using a set of global variables and an algorithm of time complexity $O(n^2)$. To check the feasibility of the time constraint and determine the promising arc exchanges, a forward time slack is computed at each node to identify the permissible shift in departure time that can be introduced without causing violations of time windows for other nodes in the route. For problem sizes up to 38 customers, a near optimal solution can be reached by this algorithm in less than 150 seconds. Unfortunately, success is not guaranteed, and low quality or even infeasible solutions can result, if the 2-phase approach gets trapped in a poor local optimum. To handle this possible situation, the authors present an alternative approach, which uses SA. This algorithm accepts time window violations in the early stages, but penalizes them more severely as the search progresses. The SA approach is able to obtain good quality solutions, albeit with a relatively high processing time.

The work reported in [88] also deals with the SV-PDPTW. This time, however, a hybrid strategy is proposed which combines an exact method with a genetic algorithm and both static and dynamic cases are considered. The static case assumes that all requests are determined in advance of the route construction process, while the dynamic case allows some requests to arrive during the construction of the route. The approach in this work consists of three consecutive stages: a pre-planned module, a dynamic programming module, and a GA. The pre-planned module arranges requests and prepares information for the dynamic programming module. The role of the dynamic programming module is to create a set of sub-routes, which it will eventually pass on to a temporary result pool, where the GA module will pick them up, installing these unfinished sub-routes to establish its initial population. In the GA module, a solution is encoded as a permutation of locations, and four crossover operators are compared: two traditional order-based crossover operators, and two merge crossover operators, MX1 and MX2, that use a global precedence vector to guide the inheritance process, as explained in [16] and later in this chapter. The mutation operator is applied only when the offspring is identical to one of its parents. Three mutation operators are used: one swaps two genes selected at random, the

second selects two random sites in the chromosome and reverses the sub-route between the selected sites, finally the third mutation shuffles the genes that precede a location for which a violation of constraints is observed. If an infeasible solution, violating the precedence constraint, is generated as a result of any genetic operator, this solution is repaired by swapping the corresponding pickup and delivery pair. The experimental results on data sets ranging from 10 to 50 requests indicate the merge crossover operator MX1 generally performed better than the other crossover operators tested. The first two mutation operators also achieved better results than the third mutation operator.

In a more recent work [89], the same authors use similar genetic operators to the ones used in [88], but in the context of a Family Competition Genetic Algorithm (FCGA). Again, an order-based representation is adopted. The idea is to allow each individual of the population to play the role of a family father in turn. Another randomly selected individual plays the role of mate for the family father. The two individuals are combined to produce an offspring in a regular GA fashion. The selection of the mate and the reproduction is repeated for a chosen number of iterations to produce a family of offspring. Only the best offspring in the family survives and is added to a temporary population of champions. The new generation is chosen from among the best individuals in both the original population and the champions of the families. Comparing the performance of the FCGA and a traditional GA on data sets created by the authors (ranging from 10 to 100 requests), the results indicate that the merge crossover operators MX1 and MX2 generally worked better in the context of a traditional GA than a FCGA, possibly due to premature convergence in the latter case. On the other hand, a traditional uniform order-based crossover (UOX) worked better within the framework of a FCGA than a traditional GA, possibly due to its uniform, non greedy, nature of exploring the search space.

The authors in [95] present a heuristic based on tabu search to solve the SV-PDPTW. The algorithm first creates a route respecting precedence and capacity constraints, using a simple insertion heuristic. The generated route may be infeasible in terms of the time window constraint, though. Then, two tabu search methods are compared in transforming the initial route to a feasible route, with minimum total distance. The two tabu search methods investigated are a regular deterministic Tabu Search (TS), and a Probabilistic Tabu Search (PTS), which is based on the same principles as the deterministic one with the addition of a buffered memory of potential moves and introducing some probabilistic criteria for the selection of the next move. The creation of a neighbouring solution in both tabu searches is based on classical neighbourhood moves, a swap operation and an insertion operation, which respect the precedence and the capacity constraints. Test results on problem instances created by the authors (from 10 to 40 customers) demonstrate the superiority of TS compared to the PTS in both run time and solution quality. Possible

reasons behind the superiority of TS were not indicated by the authors, though. For the problem sizes tested in their approach, a feasible solution could be reached by both tabu search methods in a reasonable amount of time. However, for larger problem sizes (more than 60 locations), the authors predict that reaching a feasible solution could possibly take more than one hour. According to the authors, an improvement in processing time could be achieved by adapting the parameters of the method, and a better analysis of the neighbourhood structure.

5.4 SV-PDPTW: Research Contribution

Our research tries to overcome some gaps currently existing in the literature that tackles the SV-PDPTW. Existing approaches usually face many difficulties, among them are solution representations that need frequent repair during the search process to correct infeasibility. In addition, handling the difficult constraints puts a huge burden on the search algorithm, usually making it quite complicated, especially if the researcher chooses to limit the search to only feasible solutions in the neighbourhood. This would indeed require time consuming computations to ensure the satisfaction of all constraints in every step during the search process, which will inevitably add to the overhead of the algorithm, making it slow or inefficient for most practical situations.

The main challenge that we are faced with in this research is the development of an appropriate solution representation, reflecting the problem and its constraints in a simple way to avoid complicating our algorithms. In addition, the representation should be coupled with intelligent neighbourhood operators that are capable of directing the search towards high quality and feasible solutions. A major objective is to avoid generating and evaluating a large number of infeasible solutions (that violate one or more of the hard constraints), in order to maximize the efficiency of the search process. Our proposed approach for handling these challenges is presented in the following two subsections.

5.4.1 The Solution Representation

A suitable solution representation for this kind of problem is not as obvious as it seems. The PDPTW is an ordering problem in which a solution could be represented as a permutation of locations, representing an order in which these locations will be visited. In the PDPTW, however, the issue of precedence should be addressed in the representation of the solution, because for this particular problem, no delivery location is allowed to precede

its corresponding pickup location. Nevertheless, the precedence order may not be maintained following the application of a neighbourhood move to a solution. For example, a simple swap of locations could disturb the precedence order and result in an infeasible solution. Consequently, a repair method would be needed to restore the solution feasibility, as done, for example, in [88] and [89]. Inevitably, such an approach would increase the processing time and complicate the algorithm.

We have developed a solution representation which avoids the precedence issue: we simply assign the same code to both the pickup and its associated delivery location, and rely on a simple decoder to identify its first occurrence as the pickup and the second as the delivery. This straightforward representation eliminates the problem of backtracking to repair an infeasible solution, and solves the precedence constraint issue in an effective way. As a result, more effort can be directed towards harder constraints such as the vehicle capacity and time windows. Figure 5.2 shows an example of a solution with 3 requests[1] following this representation.

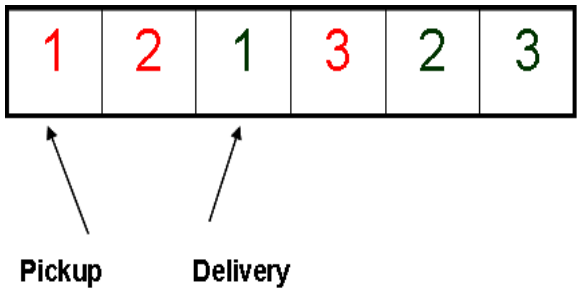


Figure 5.2: Solution Representation.

5.4.2 The Neighbourhood Move

As previously mentioned, while creating a neighbouring solution during the search process, we are faced with a major challenge: the possible violation of one or more of the problem constraints that may follow such moves. A neighbourhood move should be intelligent enough to direct the search towards high quality and feasible solutions, and thus avoid valuable time being wasted evaluating a large number of infeasible solutions. When designing such moves, all problem constraints should be considered. Nevertheless, in our research we found the time window constraint the most difficult to deal with. Recall that our duplicate encoding scheme renders the precedence constraint trivial. Furthermore, the capacity constraint tends to be easy to satisfy in most problem instances, because half of

[1] Recall from Section 5.2 that a request refers to a pair of locations (pickup and delivery).

the locations visited in any route are delivery locations whose demands are removed from the vehicle.

In our research we adopt neighbourhood search operators that apply 'randomness', yet at the same time take account of the time windows, and bring the more 'urgent' locations earlier in the visiting order, where this is possible. This neighbourhood idea is adopted in all our search algorithms, with slight variations depending on the context in which it is applied.

The rest of this chapter and the next chapter are dedicated to our suggested solution methodologies for handling the SV-PDPTW. The following section addresses our problem-specific genetic algorithm that we adopted for solving this problem. On the other hand, our investigation of the SV-PDPTW continues in Chapter 6, which presents the proposed operators within the framework of three different heuristic and meta-heuristic techniques to deal with this problem.

5.5 A Genetic Algorithm Approach for Solving the SV-PDPTW

The focus of this part of our research is to investigate the potential of using genetic algorithms to solve the SV-PDPTW. In particular, herein we experiment with a *duplicate gene encoding* that guarantees the satisfaction of the precedence constraint, between the pickup and the delivery, throughout the search. Thus, our chromosome encoding simply follows the representation depicted in Figure 5.2. In addition, several problem-specific genetic operators are tested and compared on a number of data sets with various sizes.

The rest of this section presents the details of our GA approach. Section 5.5.1 describes the fitness function that the GA relies on during the evolutionary process. Section 5.5.2 introduces our suggested genetic operators. Section 5.5.3 explains the procedure adopted to create test data for the problem. Finally, Section 5.5.4 details the experimental findings of this part of the research,

5.5.1 The Fitness Function

Following [89], the fitness function treats the constraints as soft constraints, meaning that an infeasible solution that violates either the capacity and/or the time window constraint

will be penalized by an added term in the fitness function. The fitness function of a route r is:

$$F(r) = w_1 \times D(r) + w_2 \times TWV(r) + w_3 \times CV(r), \tag{5.1}$$

where $D(r)$ is the total route duration including the waiting time and the service time at each location, $TWV(r)$ is the total number of time window violations in the route, and $CV(r)$ is the total number of capacity violations. w_1, w_2, and w_3 are weights in the range $[0, 1]$ assigned to each term in the fitness function, and $w_1 + w_2 + w_3 = 1.0$.

The fitness function will thus try to minimize infeasibility as well as the total route duration. Also, the choice of appropriate weights depends on the importance of each term in the fitness function. Specifically, the highest penalty must be imposed on the number of time window violations, since it is the most difficult constraint to satisfy in the SV-PDPTW. We used the following values for the weights of the fitness function: $w_1 = 0.001$, $w_2 = 0.7$, and $w_3 = 0.299$. Our experimentation with different values in the range $[0, 1]$ indicated that if a smaller weight value was assigned to the TWV term, the best individual resulting at the end of the evolutionary process is often infeasible, in terms of violating the time window constraint.

5.5.2 The Operators

Two crossover operators have been used in our GA, the first crossover operator is a **Merge Crossover (MX1)**, first suggested in [16]. The second crossover operator is a new problem specific crossover that we called the **Pickup and Delivery Crossover (PDPX)**. Also, two mutation operators were used in our research. A regular **gene swap mutation** and another constraint oriented **directed mutation**. These four operators are explained below.

Crossover

MX1 Crossover: Several crossover operators were considered potentially suitable for this kind of problem. The first crossover operator we tried follows the merge crossover operators suggested in [16] for the VRPTW[2], and used in [88] and [89] for the SV-PDPTW. Unlike traditional crossover operators for ordering problems, which depend only on the order of genes in a chromosome, the merge crossover operators depend on a global precedence among genes, such as the time window or distance ordering. Traditional

[2]Recall from Chapter 3 that in the VRPTW, the same service type applies to all locations, either pickup or delivery.

order-based crossover operators are not very effective for highly constrained problems like the SV-PDPTW, since they frequently produce infeasible solutions. Merge crossover operators, however, were shown to be superior to the traditional ones for these types of problems [16] [88] [89].

In the current research we have slightly modified the MX1 operator used by [88] and [89]. Instead of creating just one child, giving priority to a parent's gene having an earlier time window lower bound, we have created two children: the first child favouring genes that have an earlier lower bound, while the second child favours genes that have an earlier upper bound. The idea is that visiting a location just before its deadline could be more beneficial than visiting it as early as possible in its allowed interval. This may help reduce the waiting time that would result if the vehicle arrives too early at a location, and may consequently decrease the total route duration. Creating two children instead of one was suggested in [16], and may help improve the quality of the new generation and accelerate the optimization process.

To illustrate how the MX1 operator works, assume that the vector shown in Figure 5.3 defines the precedence order, in terms of the *lower bound* of the time window, among all pickup and delivery locations, where a (+) following a request number indicates that this is its pickup location and a (-) indicates the delivery location for the same request.

$$\{2+, 1-, 3+, 1+, 4-, 2-, 3-, 4+\}$$

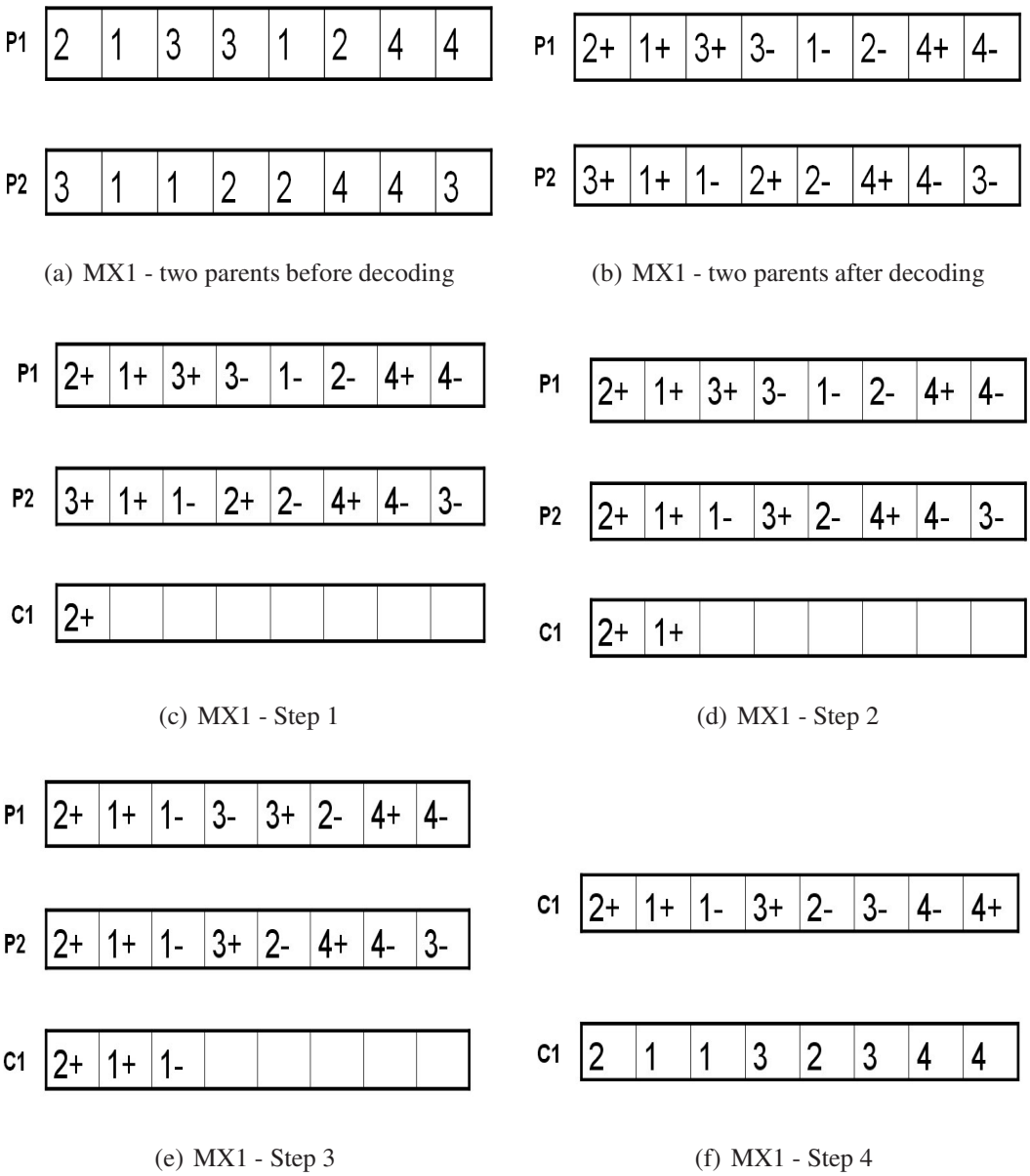
Figure 5.3: Time window precedence vector.

Figure 5.4 illustrates the steps of the MX1 operator. Consider the two parent solutions, P1 and P2, shown in Figure 5.4(a). The MX1 crossover starts by decoding the parent solutions to differentiate the pickup locations from the delivery locations, as illustrated in Figure 5.4(b). Then, the MX1 proceeds as follows:

- **Step 1**: since 2+ has a higher precedence than 3+ (as indicated by the precedence vector depicted in Figure 5.3), the child will inherit 2+ as the first gene, as shown in Figure 5.4(c);
- **Step 2**: to maintain feasibility, 2+ in P2 will be swapped to the first location. Now the second gene in both parents is identical, so it is copied to the child, as shown in

Figure 5.4(d), and we move on to the next gene in order;

- **Step 3**: now since the next gene in P2 (1-) has a higher precedence than 3+ of P1, 1- is copied to the third gene in the child, and 1- is swapped with 3+ in P1, as shown in Figure 5.4(e);
- **Step 4**: continuing in the same manner, we obtain the child C1 shown in Figure 5.4(f), together with its final format, using the duplicate encoding. Note that after the MX1 crossover, the last 2 genes of C1 were out of order (4- followed by 4+), but this is of no concern since the first one is automatically considered as the pickup.



(a) MX1 - two parents before decoding

(b) MX1 - two parents after decoding

(c) MX1 - Step 1

(d) MX1 - Step 2

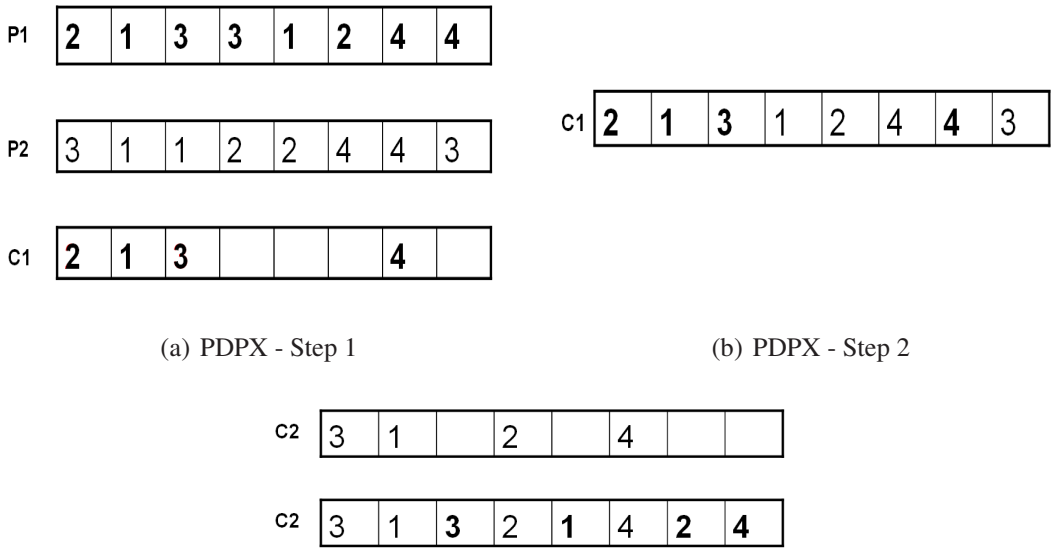
(e) MX1 - Step 3

(f) MX1 - Step 4

Figure 5.4: Merge Crossover (MX1).

The second child is created in a similar manner but with the precedence vector defined by the *upper bound* of time window intervals instead.

PDPX Crossover: In addition to the MX1 operator, we designed a new problem-specific crossover operator for the SV-PDPTW. The idea is that, since any crossover operator essentially tries to create a child that inherits half of its genes from the first parent and half of its genes from the second parent, we will try to transmit pickup locations to the child in an order close to the order of their appearance in the first parent, and try to transmit delivery locations to the child in an order close to the order of their appearance in the second parent. A second child is created by simply reversing the role of parents. This kind of crossover might prove useful in satisfying the capacity constraint in particular, since the satisfaction of this constraint is mainly dependent on the ordering of pickups and deliveries. Figure 5.5 illustrate how the PDPX crossover works.



(a) PDPX - Step 1

(b) PDPX - Step 2

(c) PDPX - second child

Figure 5.5: Pickup and Delivery Crossover (PDPX).

Starting from the two parent solutions, P1 and P2, shown in Figure 5.5(a), the PDPX crossover proceeds as follows:

- **Step 1**: first all pickups (first occurrences of genes) will be copied to the child in the exact order and locations as the first parent, as shown in Figure 5.5(a);
- **Step 2**: now, we want to take the order of delivery locations from the second parent. Since a delivery is always the second occurrence, we will start processing P2 from

the last gene rather than the first. If we encounter a gene that appeared in the child only once, i.e., it has already been picked up, we will copy it in the first available location in the child (also processed starting from the last gene). If, on the other hand, we encounter a gene that has already appeared twice in the child, this gene will be ignored because pickup and delivery has already occurred for this gene, and we move on to the next gene in order. Following this, we obtain the child shown in Figure 5.5(b).

The second child is obtained by taking the pickups order from the second parent and the deliveries order from the first parent, as shown in Figure 5.5(c).

Mutation

As mentioned above, we experimented with two mutation operators in our GA. The first is a random gene swap mutation, which selects two genes at random and swaps them. Using the duplicate gene encoding eliminates the possibility of infeasible solutions, in terms of precedence order, that may result following such a swap.

We also implemented, a new problem oriented mutation operator, named ***Directed Mutation***. As mentioned above, it appears that the TW constraint is the most challenging to satisfy among all other problem constraints. To deal with the TW constraint, a mutation operator can try to bring a location that may be more urgent, to an earlier point in the visiting order. This may result in a better ordering of locations, which should be beneficial in satisfying the timing requirement. Instead of a traditional random swap of two genes, this new mutation operator only swaps genes if they are out of order in terms of their late TW bounds, i.e., if the later one has a deadline that precedes the earlier one. The directed mutation operator is shown in Figure 5.6.

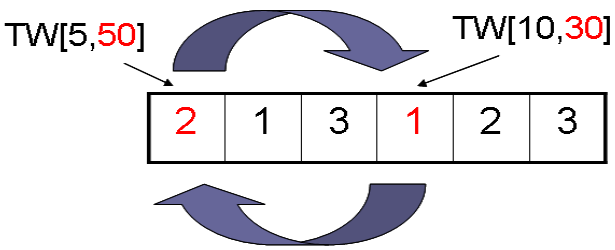


Figure 5.6: Directed Mutation.

5.5.3 Test Data

Since there are no standard benchmark test cases available for the SV-PDPTW, our GA was tested on a data set obtained from the authors of [89]. This data set has a number of customer requests ranging from 10 to 100 (20 to 200 locations). In addition, for a more extensive and thorough testing of the operators, we have created a new data set with larger numbers of customer requests ranging from 100 to 200 (200 to 400 locations).

To create test data for the SV-PDPTW, it is essential to ensure the existence of at least one solution that satisfies all problem constraints. Similar to [89], our algorithm first creates a route that respects precedence and capacity constraints, then a time window interval is generated for each location based on the arrival time realized in the created route. Unlike [89], however, we rely on our GA to create the initial feasible solution, while they create their starting feasible solution randomly. More specifically, the following steps were followed to create the test data:

1. Generate a random vehicle capacity within a certain predetermined range;
2. Generate random x and y coordinates for the depot;
3. For all pickup and delivery locations:
   (a) Generate random x and y coordinates;
   (b) Generate a random demand (load) within a certain allowable range, such that the demand of a delivery location is the same as the demand of its corresponding pickup location, but with a negative sign;
   (c) Assume a very large time window interval that could not possibly be violated;
4. Run the GA to obtain a feasible solution. Note that, the precedence constraint is always satisfied in any generated solution, thanks to our duplicate gene encoding. Moreover, due to the nature of the problem, the satisfaction of the capacity constraint can be easily accomplished by our genetic operators. The timing constraint can also be easily satisfied because the TW intervals are very large at this point;
5. Calculate the arrival time at each location in the feasible route obtained;
6. Create a random time window interval for each location such that the arrival time falls within the created time window. The width of the permitted time window interval should be determined in advance.

Algorithm 5.1 describes the initial steps carried out for the creation of test data for the problem without the final time window intervals, i.e., Steps 1 to 3 above. In this algorithm, the allowable ranges for the random values were determined empirically. Note that all ranges were scaled according to the number of requests currently generated, i.e., the larger the number of requests, the larger the allowable range for the vehicle capacity and the demand of the location. Algorithm 5.2 describes the procedure for determining the TW intervals, i.e., Step 6 above.

Algorithm 5.1: Create Test Data.

1: Choose values for constants $MinCapacity$, $MaxCapacity$, $MinCoord$ and $MaxCoord$ {We used $MinCapacity = 1$, $MaxCapacity = 10$, $MinCoord = 0$ and $MaxCoord = 200$}
2: Choose value for constant $MinTW$ {A very small value (we used 0)}
3: Choose value for constant $MaxTW$ {A very large value (we used 100000)}
4: Let $nrequests = n/2$ be the number of requests in the set, where n is the total number of nodes (locations) in the data set, excluding the depot
5: $VehicleCap = Random(MinCapacity \times nrequests, MaxCapacity \times nrequests)$ {Function $Random$ generates a random value between its two parameters}
6: $AverageDemand = VehicleCap/nrequests$ {The average demand of a request}
{Generate coordinates, demand, and TW for the depot and the pickup locations}
7: **for** $(pickloc = 0; pickloc <= nrequests; pickloc++)$ **do**
8: $x[pickloc] = Random(MinCoord, MaxCoord)$ {The x-coordinate}
9: $y[pickloc] = Random(MinCoord, MaxCoord)$ {The y-coordinate}
10: **if** $(pickloc = 0)$ **then**
11: $demand[pickloc] = 0$ {The demand of the depot is 0}
12: **else**
13: $demand[pickloc] = Random(AverageDemand \times m_1, AverageDemand \times m_2)$ {The choice of m_1 and m_2 depends on how big we want the demand of the location}
14: $ETW[pickloc] = MinTW$ {The lower bound of the TW}
15: $LTW[pickloc] = MaxTW$ {The upper bound of the TW}
{Generate coordinates, demand, TW for delivery locations}
16: **for** $(delloc = nrequests + 1; delloc <= n; delloc++)$ **do**
17: $x[delloc] = Random(MinCoord, MaxCoord)$
18: $y[delloc] = Random(MinCoord, MaxCoord)$
19: $demand[delloc] = -demand[delloc - nrequests]$ {Same as its pickup demand but -ve}
20: $ETW[delloc] = MinTW$ {The lower bound of the TW}
21: $LTW[delloc] = MaxTW$ {The upper bound of the TW}

Algorithm 5.2: Generate Time Windows.

1: Choose value for constant $CurrentWidth$ {We used 10}
2: Calculate the arrival time of the vehicle at each location in the feasible route obtained by the GA
3: **for** $(loc = 1; loc <= n; loc++)$ **do**
4: $ETW[loc] = Arrival[loc] - Random(CurrentWidth \times m_1, CurrentWidth \times m_2)$ {The choice of m_1 and m_2 depends on whether we want the time window interval to be wide or narrow}
5: **if** $(ETW[loc] < 0)$ **then**
6: $ETW[loc] = Arrival[loc] - Random(1, Arrival[loc])$
7: $LTW[loc] = Arrival[loc] + Random(CurrentWidth \times m_1, CurrentWidth \times m_2)$

5.5.4 Experimental Results

We implemented a steady state GA with a replacement percentage of 100% and a population size of 1000. Algorithm 5.3 outlines the GA used in our implementation.

Algorithm 5.3: The SV-PDPTW Genetic Algorithm.

1: Initialize a random population POP of candidate solutions to the SV-PDPTW, using the duplicate encoding of pickup and delivery locations
2: **while** (stopping condition is not reached) **do**
3: **for** (i=0; i<NumCrossovers;i++) **do**
4: Select parents P_1 and P_2 from POP, using roulette wheel selection
5: Apply crossover to parents P_1 and P_2 to produce two children C_1 and C_2
6: With some probability, apply mutation to C_1 and C_2
7: Update POP by integrating the new generation and eliminating some worst individuals {i.e., steady state GA with overlapping populations}

To test our GA, different combinations of crossover and mutation operators were compared on two data set samples. The first sample, which we will call **SET 1**, is the sample obtained from [89] and includes 30, 80, 90 and 100 customer requests[3]. It should be noted that while solving the problem for this data set, the authors of [89] assumed that the vehicle's journey is open path. Thus, although the vehicle should start its journey from the depot, it could end at any of the delivery locations. In order to compare our results with their results we also make this assumption in our GA approach. An alternative, and more 'standard', assumption is for the vehicle to start and end at the depot.

[3]Source: `http://wrjih.wordpress.com/2006/12/09/pdptw-test-data/`

The second data set used to test our algorithm, which we will call **SET 2**, is obtained from the instances created by us, and includes 130, 170, and 200 customer requests [4]. Note that the number of locations is always double the number of requests. The algorithm was run 10 times on each test case, with a crossover rate of 1.0 and a mutation rate of 0.4[5].

The following combinations of crossover and mutation operators were compared:

1. MX1 crossover and random swap mutation.
2. MX1 crossover and directed mutation.
3. PDPX crossover and random swap mutation.
4. PDPX crossover and directed mutation.
5. Directed mutation without any crossover[6].
6. MX1 crossover without any mutation.

The genetic algorithm was run to convergence or for a maximum of 3000 generations if no convergence can be reached. Since robustness is one of our main goals in this research, the GA was run with the same parameters on all problem instances, i.e., it was not specifically tuned or optimized for each instance separately.

The results for the 10 runs are recorded in Table 5.1 as follows, for each test case under each GA version: the best score found (in terms of *total route duration* only), and the percentage of feasible solutions obtained during the 10 runs. The last column of the table shows the previous best result. For SET 1, the previous best results are the ones reported in [89], during their computational experimentation with these problem instance[7], while for SET 2, the previous best results are the best results achieved in the current experiment. A score followed by a (*) indicates that this solution is infeasible in terms of time window constraint violation. When the obtained result is the same as or better than the previous best result, it is highlighted in boldface.

In terms of the objective function values, Table 5.2 shows the average, best, and worst objective function values, as calculated by Equation 5.1, for all versions of our GA. the

[4]Source: `http://users.cs.cf.ac.uk/M.I.Hosny/PDP.zip`

[5] A higher than usual mutation rate was found necessary to avoid being trapped in a local optimum. One reason could be the duplicate encoding, a side effect of which is that mutation may swap identical genes producing the same offspring.

[6]Mutation rate was set to 1 and crossover rate to 0

[7] The results shown are truncated as they appear in [89].

Table 5.1: GA Results Summary (Total Route Duration).

Data	Task	MX1&Swap		MX1&Directed		PDPX&Swap		PDPX&Directed		Directed Mut		MX1		Prev Best
		Best	%Feas	Best	%Feas	Best	%Feas	Best	%Feas	Best	%Feas	Best	%Feas	
SET 1	**30**	**3696.51**	100	3738.73	100	3863.27*	0	3734.54	100	3747.47	100	3741.10	100	3713
	80	**7838.21**	100	7858.72	100	9163.92*	0	**7838.21**	100	7879.35	90	7867.68	80	7849
	90	8619.01	100	8619.01	100	10057.0*	0	8623.23	100	8662.20	100	8619.01	100	8618
	100	**10600.10**	100	10600.10	100	11886.10*	0	10608.20	80	10608.20	90	10600.10	90	10600
SET 2	**130**	**13856.20**	100	15913.10	20	19185.40*	0	14791.70	60	14679.20	100	21626.0*	0	13856.20
	170	**19861.30**	100	20674.50	60	25817.10*	0	20359.30	90	20470.20	90	30575.60*	0	19861.30
	200	**24512.80**	80	26058.10	60	32009.70*	0	24911.20	40	25091.70	70	37148.50*	0	24512.80

Table 5.2: GA Results Summary (Objective Function Values).

Task	MX1&Swap				MX1&Directed				PDPX&Swap			
	Avg	Best	Worst	SD	Avg	Best	Worst	SD	Avg	Best	Worst	SD
30	3.72	3.70	3.74	0.02	3.74	3.74	3.74	0	15.82	13.66	17.34	1.19
80	7.84	7.84	7.85	0	7.88	7.86	7.89	0.01	81.52	72.86	87.10	4.36
90	8.62	8.62	8.62	0	8.62	8.62	8.62	0	102.78	94.76	114.5	6.68
100	10.60	10.60	10.60	0	10.60	10.60	10.60	0	114.43	101.49	127.16	8.90
130	14.06	13.86	14.23	0.14	56.33	15.91	108.21	35.03	124.0	118.59	128.68	3.88
170	20.17	19.86	20.6	0.24	21.98	20.67	23.95	1.04	183.66	174.22	192.86	5.93
200	24.72	24.51	25.4	0.34	29.27	26.06	47.70	6.61	225.55	214.71	245.10	8.32
Avg	**12.82**	**12.71**	**13.01**	**0.11**	**19.78**	**13.35**	**30.1**	**6.10**	**121.11**	**112.9**	**130.39**	**5.61**
Task	**PDPX&Directed**				**Directed Mutation**				**MX1**			
	Avg	Best	Worst	SD	Avg	Best	Worst	SD	Avg	Best	Worst	SD
30	3.77	3.73	3.80	0.02	3.80	3.75	3.86	0.03	3.74	3.74	3.75	0
80	7.91	7.84	7.97	0.04	8.0	7.88	8.63	0.22	8.03	7.87	8.59	0.29
90	8.69	8.62	8.78	0.06	8.70	8.66	8.76	0.03	8.62	8.62	8.62	0
100	10.80	10.61	11.38	0.3	10.73	10.61	11.37	0.23	10.67	10.60	11.30	0.22
130	15.42	14.79	16.59	0.66	15.14	14.68	15.49	0.27	124.53	111.23	132.67	6.85
170	20.84	20.36	22.07	0.49	20.95	20.47	23.03	0.74	178.85	169.48	189.61	6.51
200	25.91	24.91	28.06	1.01	25.46	25.09	26.35	0.39	229.47	214.25	240.92	6.96
Avg	**13.33**	**12.98**	**14.09**	**0.37**	**13.26**	**13.02**	**13.93**	**0.27**	**80.56**	**75.11**	**85.07**	**2.98**

Standard Deviation (SD) value of the objective function for the 10 runs is also shown in this table.

As can be observed in Table 5.1, the best results are obtained from the **MX1 crossover and random swap mutation**. Together they achieved a 100% feasibility rate in all test cases except for the largest task (200 requests) in which the feasibility rate was 80%. In two cases the results obtained were even better than the previous best results reported in [89]. These are the results for test cases 30 and 80. Table 5.2 also shows clearly that the overall average objective function results of the MX1 crossover and swap mutation were better than those obtained by all other GA versions, for all categories demonstrated in the table.

Table 5.1 shows that the worst results are obtained by the **PDPX crossover and swap mutation**. These two operators together failed to achieve any feasible results, in terms of the time window constraint, even for the small sized tasks. Also, the overall average objective function results for these two operators were considerably worse than the results obtained by all other GA versions, as Table 5.2 indicates. These inferior results may be explained by noticing that both PDPX crossover and swap mutation lacked any guidance towards the time window ordering. Although they were able to satisfy the capacity constraint in all test cases, they failed to satisfy the time window constraint. Clearly, they worked blindly with respect to the timing order, and consequently were not able to find an ordering that services all requests in their desired intervals.

When directed mutation replaced the random swap mutation, however, the results dramatically improved for the PDPX. Table 5.1 shows that **PDPX together with directed mutation** were able to obtain feasible solutions in all test cases. The feasibility ratio ranged from 40% for the largest task to 100% for the smallest task. The overall results, though, were not able to beat the results obtained from the MX1 crossover and swap mutation, which can be seen in Table 5.2. Figure 5.7 sustains the above observation, by showing how directed mutation was able to guide the search and helped the PDPX crossover to find much better solutions for the largest task tested (200 requests), when used in place of the random swap mutation.

The situation is different, however, when directed mutation replaced random swap mutation with the MX1 crossover. As shown in Tables 5.1 and 5.2, **MX1 crossover and directed mutation** produced lower quality solutions than those obtained by MX1 crossover with the random swap mutation. Possibly, the introduction of directed mutation caused the algorithm to rapidly converge to a local optimum. Figure 5.8 bears this out, and shows the rapid convergence of the MX1 crossover with directed mutation compared to the gradual convergence of the MX1 crossover with the random swap mutation.

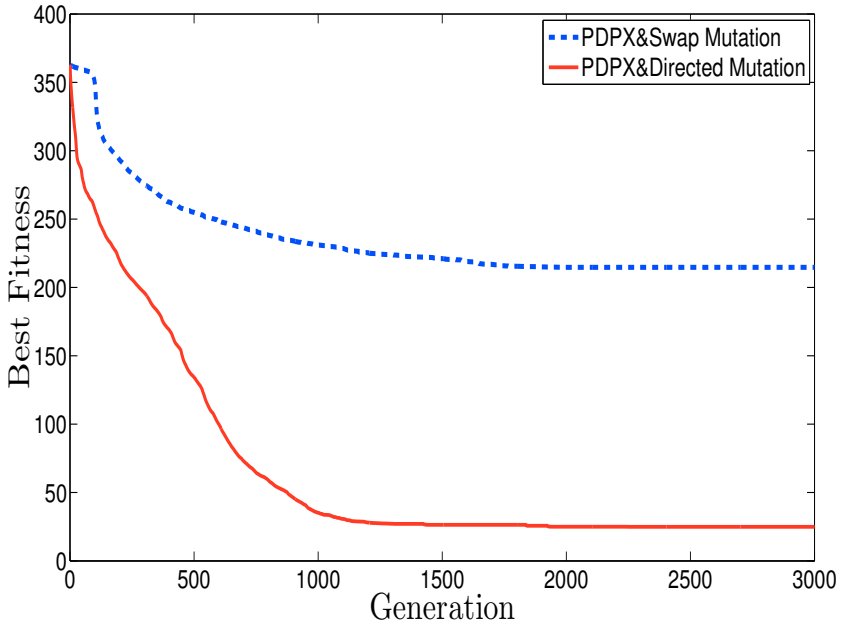


Figure 5.7: PDPX & Swap against PDPX & Directed - Task 200.

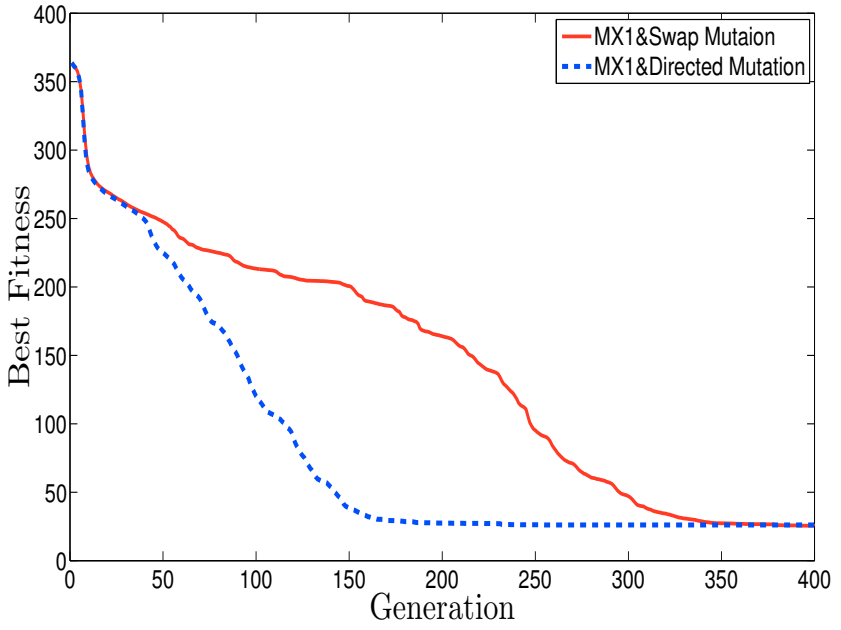


Figure 5.8: MX1 & Swap against MX1 & Directed- Task 200.

The MX1 crossover thus seems capable of guiding the GA towards feasible solutions without the help of an 'intelligent' mutation operator. Its success is possibly due to the fact that it creates not only one but two children that have a better visiting order than their parents. These children are biased towards the more urgent requests in terms of the allowed service time. All that this crossover needs is a simple random swap to introduce a bit of a diversity in the population. Any more 'intelligent' interference does not seem to help and actually appears to drive the search away from better solutions.

The importance of the random swap mutation to help introduce diversity can also be noticed in Tables 5.1 and 5.2, by observing the results when **MX1 worked alone without any mutation**. The results in Table 5.1 show that MX1 failed to find some previously achieved best results in the small tasks (SET 1), and even failed to achieve any feasible results in all large tasks (SET 2). In addition, the overall average objective function results produced by MX1 alone were considerably worse than the results produced when MX1 worked together with the random swap mutation, which can be clearly realized from Table 5.2.

Finally, when **directed mutation was tested without any crossover**, the results obtained were, to some extent, surprising. As shown in Table 5.1, this mutation alone was able to obtain feasible solutions for all test cases. The feasibility rate was 90% or 100% for all test cases except for the largest one, in which the feasibility rate was 70%. Table 5.2 shows that in terms of the objective function results, the GA with directed mutation obtained the second best results (after the MX1 and swap mutation results). Both tables indicate that this mutation is an intelligent operator that has a great potential, since it was able to obtain good quality and feasible solutions without the help of any crossover operator.

It should be noted, though, that when directed mutation was tested with MX1 crossover, only one swap movement was needed to inject the necessary level of diversity and to reduce the chance of being trapped in a local optimum. On the other hand, when directed mutation was tested alone or with the PDPX crossover, it was found that a larger number of swaps was needed to achieve good results. For these cases, the number of swaps was taken to be a random number between 1 and the number of requests in the data set.

Table 5.3 shows the average number of generations of the 10 runs needed to reach the best individual in the population, for all tasks and all GA versions. It is clear from this table that the GA with MX1 crossover (alone) had the smallest average number of generations, in all test cases. This again indicates the lack of diversity in the population in this GA version, due to the absence of mutation, which led to the rapid convergence to a low quality solution. On the other hand, the GA with PDPX crossover and random swap mutation had the largest average number of generations in most test cases. This could

be the result of some kind of 'chaotic' search that these two operators followed, since they lacked any guidance towards the promising areas of the search space, as previously discussed. Also, the GA with directed mutation (alone) had the largest average number of generations in two test cases (130 and 200 requests). This is most probably due to the absence of crossover, which is usually beneficial in exploring wide areas of the search space and finding good solutions within a fewer number of generations. Similarly, the GA with PDPX and directed mutation needed a relatively large number of generations in all test cases. This sustains our previous observation that the crossover operator in this GA version seems to be ineffective, and the search could be relying mostly on directed mutation to obtain good quality solutions.

Table 5.3: Average Number of Generations to Reach the Best Individual.

Task	Mx1 & Swap	Mx1 & Directed	PDPX & Swap	PDPX & Directed	Directed Mutation	MX1
30	31	15	704	39	35	13
80	116	15	943	414	267	14
90	23	18	1811	421	341	17
100	17	14	1696	617	562	14
130	444	230	1000	893	1432	53
170	446	382	2000	1769	1572	54
200	851	365	2000	2268	2320	49

5.6 Summary and Future Work

In this part of our research we introduced the first variant of pickup and delivery problems that we tried to handle, namely the single vehicle with time windows. After describing the problem and the motivation behind the research in addition to a summary of some related work, we explained our view regarding the solution representation and the neighbourhood moves that we believe can help conquer the difficult problem constraints. We then examined the suggested 'ideas' in the context of a problem-specific genetic algorithm.

Specifically, the encoding used in our GA is a duplicate gene encoding that guarantees the satisfaction of the precedence constraint throughout the search, alleviating the need for backtracking to solve the infeasibility that may result following any genetic manipulation. Four genetic operators were tested: a 2-child merge crossover (MX1) guided by

precedence of both time window bounds, a new pickup and delivery crossover (PDPX) that depends on the order in which the pickup and delivery locations appear in parent solutions, a regular random swap mutation, and a directed mutation that swaps genes according to the urgency of service time.

The experimental results on two data set samples indicate that the **MX1 crossover** and the **directed mutation** are each effective as genetic operators (although they do not work very well together). We believe their success is due to the guidance they take from the time window information. On the other hand, the PDPX crossover was not able to find feasible solutions to any of the test instances, when tested in combination with the random swap mutation. This was probably due to the absence of guidance towards the desired service time. However, the results for PDPX dramatically improved, when used in combination with directed mutation. Indeed, the directed mutation operator seems to show great promise, and would appear to be useful in guiding the search towards feasible solutions, in cases where a crossover operator is disruptive, ineffective or absent. Our upcoming research will attempt to investigate the potential of this operator for the SV-PDPTW in the context of other heuristics and meta-heuristics. The details of this investigation is presented in the next chapter.

Chapter 6

Several Heuristics and Meta-heuristics Applied to the SV-PDPTW

Taking into consideration the challenges of developing an appropriate solution representation and effective neighbourhood moves to facilitate dealing with the constraints of the SV-PDPTW, in this part of our research we investigate and compare three approaches to the problem. The first approach is a GA approach, similar to that introduced in Chapter 5. As previously mentioned, this approach adopts a solution representation having the same code for both the pickup and its associated delivery, and applies some problem-oriented genetic operators. The second solution methodology we investigated is Simulated Annealing (SA), which adopts the same solution representation as the one used in the GA. Also inspired by the GA mutations, two SA neighbourhood strategies are tested: a random blind move, and an intelligent move that is directed by the time window. The third approach is a simple Hill Climbing (HC) heuristic that creates a starting solution and then tries to improve it, replacing the current solution with better solutions generated during the search. The same solution representation and the directed neighbourhood move used in both the GA and the SA are also adopted in the HC. As previously indicated in Chapter 5, the details of the three approaches and the results of this part of our research were published in the *Journal of Heuristics* [84].

Before we start the discussion of our solution methodologies, we present, in the following section, the objective function used in this part of the research. The three sections that introduce our selected approaches to the problem are Sections 6.2 (GA), 6.3 (SA) and 6.4 (HC). Section 6.5 then details our findings when the three approaches were analyzed and compared, and Section 6.6 elaborates on some further analysis of the SA algorithm, which is the most promising approach among those tested. Finally, Section 6.7 concludes this chapter with some suggested future work.

6.1 The Objective Function

In this part of the research, we slightly modified the objective function used by [89] and also by our GA in Chapter 5 (i.e., Equation 5.1). The current objective function additionally penalizes the amount of time delay, together with the number of time window and capacity violations. The idea is that adding a penalty for the extent of total tardiness in the route could possibly direct the search towards better quality and feasible solutions, especially if the problem size increases. Thus, the objective function of a route r is now described by the following equation:

$$F(r) = w_1 \times D(r) + w_2 \times TWV(r) + w_3 \times CV(r) + w_4 \times TD(r), \qquad (6.1)$$

where $D(r)$ is the total route duration including the waiting time, if the vehicles arrives at a location before the start of its service time. $TWV(r)$ is the total number of time window violations in the route. $CV(r)$ is the total number of capacity violations, and $TD(r)$ is the total amount of delay if the vehicle arrives at a location later than the specified deadline. w_1, w_2, w_3 and w_4 are weights in the range $[0, 1]$, and $w_1 + w_2 + w_3 + w_4 = 1.0$. While experimenting with this objective function, we found that in order to eliminate infeasibility, a higher penalty should be imposed on the time window violations and the total delay than the capacity violations or the total route duration. We used the following values for the penalty weights in our objective function: $w_1 = 0.001$, $w_2 = 0.6$, $w_3 = 0.099$, and $w_4 = 0.3$.

6.2 The Genetic Algorithm (GA) Approach

The genetic algorithm used in the current research is identical to the GA used in the first part of the research, as presented in Chapter 5. Nevertheless, here we only applied the two most promising genetic versions. Namely, the GA with MX1 crossover and random swap mutation, and the GA with directed mutation only. Also, the GA adopted here uses the new objective function described in Equation 6.1, instead of the previous objective function (Equation 5.1).

6.3 The 3-Stage Simulated Annealing (SA) Approach

As previously explained in Section 2.3.2, the theoretical foundation of simulated annealing was established in [93]. Since then, the algorithm has been successfully used in

solving many combinatorial optimization problems, including vehicle routing and scheduling problems.

To solve the SV-PDPTW using SA, we first need to construct an initial solution, and then try to optimize this solution using a method analogous to the annealing of solids. Some researchers choose to apply different heuristic techniques to create an initial solution with minimum constraint violations, for example [23] and [95]. In our research, we construct an initial solution s by simply generating a large number of random routes and selecting the route with minimum cost (according to Equation 6.1) to be our starting solution[1]. This technique is straightforward to implement and also computationally inexpensive. Although the quality of the initial solution may be poor, the main focus of the algorithm is on the route improvement phase, accomplished during the SA search. A good neighbourhood move should be capable of directing the SA algorithm to reach a final high quality solution, irrespective of the quality of the starting solution. The solution representation adopted in the SA approach is the same as the solution representation of the GA, i.e., assigning the same code to the pickup and its delivery as shown in Figure 5.2.

The choice of an appropriate annealing schedule is critical to the performance of SA. Ideally it is desirable to devise a scheme that is adaptable for all test cases and problem sizes, eliminating the need for 'arbitrary' parameter tuning, which can be very time consuming. Following [40], we created the annealing parameters for each test case individually as shown in Algorithm 6.1. To generate a neighbouring solution s' in this algorithm, a simple random swap between two different locations in s is performed.

The main SA algorithm is shown in Algorithm 6.2. This algorithm starts with the same solution s used to calculate the annealing parameters in Algorithm 6.1. This time, however, to get a new state s' from the current state s, an intelligent neighbourhood move, that depends on the upper bound of the time window, is used. First two locations in the current solution are selected at random. Then, these two locations are swapped, only if their deadlines are out of order, i.e., if the latter has a more urgent deadline than the earlier one. This is exactly the same idea adopted in the directed mutation used in the GA (see Section 5.5.2 and Figure 5.6), and its purpose is to arrange requests in a way that will best satisfy the timing constraint.

[1] We generated 10000 random solutions and selected the best one among them.

Algorithm 6.1: Calculating Annealing Parameters [40].

Given an initial solution s
{Initialize $Pstart$ the starting acceptance probability and $Pend$ the final acceptance probability}
Let $Pstart$ = a large value {We used 0.999}
Let $Pend$ = a small value {We used 0.001}
$\Delta_{avg} \leftarrow 0$
{Generate n neighbouring solutions of s}
for $(i = 0; i < n; i++)$ **do**
    Select two random locations in s
    Swap the current two locations in s to get a new solution s'
    $\Delta \leftarrow |cost(s') - cost(s)|$
    $\Delta_{avg} \leftarrow \Delta_{avg} + \Delta$
$\Delta_{avg} \leftarrow \Delta_{avg}/n$
$T_0 \leftarrow -\Delta_{avg}/\log(Pstart)$ {T_0 is the initial temperature}
$T_N \leftarrow -\Delta_{avg}/\log(Pend)$ {T_N is the final temperature}
$\alpha \leftarrow \exp^{(\log(T_N)-\log(T_0))/N}$ {α is the temperature reduction factor, and N is the number of iterations desired}
Return T_0, T_N and α

Algorithm 6.2: The Main SA Algorithm.

Given an initial solution s, a starting temperature T_0, and a temperature reduction factor α
$T \leftarrow T_0$ {Initialize the current temperature}
repeat
    Select two random locations in s
    if (The latter location has a smaller value of the upper bound of the time window) **then**
        Swap the current two locations in s to get a new solution s'
        $\Delta \leftarrow cost(s') - cost(s)$
        if $(\Delta < 0)$ **then**
            $s \leftarrow s'$
            $T \leftarrow \alpha \times T$ {Reduce the current temperature}
        else
            $p = Random(0, 1)$ {generate a random number in the interval (0,1)}
            if $(p < \exp^{(-\Delta/T)})$ **then**
                $s \leftarrow s'$
until (Frozen){Stop when no improvement is achieved for a pre-specified number of iterations (we used 5000 iterations)}
Return s

During any iteration of the main SA algorithm, if the currently selected locations are not out of sequence in terms of their deadlines, however, no swap of locations will take place. As a result, the current solution will remain intact, and no change in cost is evaluated. The temperature value T also remains unchanged and is carried forward to the next iteration. In fact, as will be explained shortly, the present temperature value is only reduced following the replacement of the current solution s with a new improved solution s'.

Nevertheless, since the final solution obtained after the main SA procedure may still be a low quality or an infeasible solution, we extended our algorithm with two further SA stages, in order to more fully exploit all the guidance that the time windows can give us. The second stage adopts a neighbourhood move depending on the *lower bound* of the time window instead of the upper bound, while the third stage uses a neighbourhood move based on the *centre* of the time window interval. Specifically, the second and third SA stages will only differ from the first SA stage (which is shown in Algorithm 6.2) in Step 5 of the algorithm, since the *lower bound* and the *centre* of the time window will be used respectively in these two stages.

The idea is that, each of these different neighbourhood moves may help introduce some improvement to the fitness of the current solution, for example by reducing the total delay or the total waiting time, which could ultimately lead to obtaining high quality solutions. Each new SA stage starts from the final solution reached by the end of the previous stage, and its starting temperature is the final temperature reached by the previous stage. We chose to start with a route improvement move that is guided by the upper bound of time window, in order to reduce the total number of time window violations in the route by visiting the more urgent requests first. However, a different improvement order could also be attempted (See Section 6.6 for a further investigation of different orders of time window moves).

In the first two stages of the SA procedure (in which the neighbourhood move depends on the upper and then the lower bound of the time window), the temperature reduction schedule was slow. As shown in Step 10 of Algorithm 6.2, we chose to reduce the temperature only when a better solution was found, i.e., several neighbouring solutions are explored for the same temperature value. The idea is to allow the algorithm a thorough exploration of the neighbourhood by accepting a large number of worse moves during the early stages of the search. However, during the final stage of the SA (the one that adopts the centre of the time window to perturb the solution), the solution was approaching stability, and there was a danger of losing the best solution obtained if the temperature was slowly reduced. Thus, during this final stage, the temperature reduction was rapid. This was achieved by reducing the temperature at every iteration of the search, which will make the probability

of accepting worse solutions very small.

To evaluate the performance of the 3-stage SA algorithm described above, it was compared with another simple SA algorithm in which the same annealing schedule and a slow cooling is used, but the neighbourhood move was a simple random swap of locations that does not take the timing order into consideration. The comparison results are reported in Section 6.5.

6.4 The Hill Climbing (HC) Approach

The final approach we used to solve the SV-PDPTW problem is a simple hill climbing heuristic. The algorithm basically has two phases: a route construction phase, and a route improvement phase. As with our SA algorithm, the initial solution is simply created by generating a large number of random solutions and selecting the best generated solution (i.e., minimum cost solution according to Equation 6.1). Again this procedure will avoid unnecessary complications and increased processing time that may result if we try to generate a high quality solution with minimum infeasibility. A guided route improvement phase, which repeatedly replaces the current solution with better solutions generated during the search, should be able to transform the starting low quality and possibly infeasible solution to a high quality and feasible solution. Our main HC algorithm is described in Algorithm 6.3.

Algorithm 6.3: The Main HC Algorithm.

1: Given an initial solution s
2: **repeat**
3: **for** (Each possible pair of locations in s) **do**
4: **if** (The latter location has a smaller value of the upper bound of the time window) **then**
5: Swap the current two locations in s to get a new solution s'
6: $\Delta \leftarrow cost(s') - cost(s)$
7: **if** ($\Delta < 0$) **then**
8: $s \leftarrow s'$
9: **until** (Done){Stop if no improvement has been achieved in the previous pass}
10: Return s

Again, in our HC we adopt the same solution representation and neighbourhood moves that were used in both the GA and the SA. Similar to the SA algorithm, the HC algorithm was divided into three stages. The first stage generates a neighbouring solution by swapping the two locations currently under consideration, only if their deadlines (i.e., the upper

bounds of the time window) are out of order. On the other hand, if the deadlines are in sequence, the current solution remains intact, and two new locations are considered in the next iteration. When no further improvement is possible using this neighbourhood move, the second stage starts from the final solution achieved in the first stage and repeats the same HC procedure, but with a neighbourhood move that adopts the lower bound of the time window to decide the swapping. Finally, the third stage starts with the final solution obtained in the second stage, but with a neighbourhood move that swaps locations if they are out of order in terms of the centre of the time window interval.

6.5 Experimental Results

To test our algorithms, we used the same data set samples on which we tested our GA, as previously explained in Section 5.5.4. Each of the following algorithms was run 10 times on each test case:

1. The GA with MX1 crossover and random swap mutation (GA1).
2. The GA with directed mutation only (GA2).
3. The 3-stage SA (SA1).
4. The simple random-move SA (SA2).
5. The Hill Climbing (HC) algorithm .

All algorithms were run on all problem instances using the same set of parameters (e.g. GA parameters, simulated annealing schedule, and stopping conditions), i.e., no particular tuning of parameters was performed for each test case separately. The results for the 10 runs are recorded in Table 6.1 as follows: for each test case under each algorithm, the best result found (in terms of *total route duration* only), and the percentage of feasible solutions obtained during the 10 runs. The last column of the table shows the previous best results, attributed to [89] and our GA approach reported in Chapter 5. Whenever the result achieved in the current experiment is better than the previous best result, it is highlighted in boldface.

Table 6.1: Results Summary for all Algorithms (Total Route Duration).

Data	Task	GA1		GA2		SA1		SA2		HC		Prev Best
		Best	%Feas	Best	%Feas	Best	%Feas	Best	%Feas	Best	%Feas	
SET 1	**30**	3738.73	100	3751.21	100	**3299.40**	100	**3684.21**	100	3772.82	70	3696.51
	80	7838.21	100	7867.68	100	**7267.0**	100	7852.25	80	7847.67	90	7838.21
	90	8619.01	100	8619.01	100	**8292.67**	100	8619.01	100	8671.54	100	8618.0
	100	10600.10	100	10600.10	90	**10544.60**	90	10611.10	70	10600.10	90	10600.10
SET 2	**130**	14041.0	100	14343.0	100	**13826.0**	100	14081.20	90	14031.10	100	13856.20
	170	20618.10	100	20333.80	100	**19690.90**	100	19964.80	100	20003.60	100	19861.30
	200	25799.50	100	24730.0	100	**23841.10**	100	**24461.60**	100	**24304.30**	100	24512.80

As shown in Table 6.1, the best route duration results were achieved by the 3-stage SA (SA1). In all test cases the best results achieved by this version of SA were better than the previous results achieved in [89] and by our GA explained in Chapter 5. The results were also superior to the results achieved by all the other algorithms tested in the current research. A very high feasibility rate and good quality solutions were obtained in all tasks tested, with an improvement of approximately 11%, over the previous best[2], for the smallest task and approximately 3% for the largest task.

In terms of the overall objective function values, Table 6.2 shows the average, best, and worst objective function values, as calculated by Equation 6.1, for the GA1 and GA2 algorithms. the Standard Deviation (SD) value of the objective function for the 10 runs is also shown in this table. Table 6.3 shows the same objective function information for the remaining three algorithms (SA1, SA2, and HC).

Table 6.2: Results Summary for GA1 and GA2 (Objective Function Values).

Task	GA1				GA2			
	Avg	**Best**	**Worst**	**SD**	**Avg**	**Best**	**Worst**	**SD**
30	3.74	3.74	3.74	0	3.81	3.75	3.89	0.05
80	7.86	7.84	7.87	0.01	7.92	7.87	7.97	0.04
90	8.62	8.62	8.62	0	8.69	8.62	8.77	0.05
100	10.60	10.60	10.60	0	11.08	10.60	14.69	1.27
130	14.51	14.04	15.28	0.35	14.85	14.34	15.16	0.25
170	20.94	20.62	21.25	0.21	20.67	20.33	20.88	0.16
200	26.20	25.80	26.45	0.21	25.04	24.73	25.33	0.21
Avg	**13.21**	**13.04**	**13.4**	**0.11**	**13.15**	**12.89**	**13.81**	**0.29**

Tables 6.2 and 6.3 also support the conclusions drawn from Table 6.1. Observing the overall average results in the last row of Table 6.2 and 6.3, we can clearly see that the SA1 algorithm outperformed all other heuristics in the average, best and worst objective function values. On the other hand, its overall average SD value was only slightly larger than the overall average SD of the GA1 algorithm. The GA1 algorithm obtained a smaller SD, due to the fact that the population converged to a sub-optimal solution in 3 out of the 7 test cases (30, 90 and 100 requests), as shown in Table 6.2.

[2]The improvement percentage is calculated using the formula: $((R^* - R)/R^*) \times 100$, where R is the current result and R^* is the previous best result.

Table 6.3: Results Summary for SA1, SA2 and HC (Objective Function Values).

Task	SA1				SA2				HC			
	Avg	**Best**	**Worst**	**SD**	**Avg**	**Best**	**Worst**	**SD**	**Avg**	**Best**	**Worst**	**SD**
30	3.53	3.30	3.66	0.1	3.71	3.68	3.74	0.02	3.83	3.77	3.96	0.06
80	7.63	7.27	7.85	0.19	9.25	7.85	18.78	3.43	9.55	7.85	24.45	5.24
90	8.46	8.29	8.60	0.11	8.69	8.62	8.75	0.04	8.69	8.67	8.72	0.02
100	10.63	10.54	11.0	0.13	21.49	10.61	75.31	20.70	10.90	10.60	13.40	0.88
130	13.95	13.83	14.12	0.11	18.68	14.08	59.05	14.19	14.51	14.03	15.08	0.37
170	20.06	19.69	20.43	0.22	20.16	19.96	20.46	0.16	20.31	20.0	20.89	0.25
200	24.37	23.84	24.68	0.25	24.71	24.46	24.94	0.15	24.63	24.30	24.99	0.22
Avg	**12.66**	**12.39**	**12.91**	**0.16**	**15.24**	**12.75**	**30.15**	**5.53**	**13.2**	**12.75**	**15.93**	**1.01**

It appears from these results that the SA1 algorithm was able to escape the trap of local optima and to gradually find better solutions by progressing from one stage to another. Each stage of the 3-stage SA (SA1) seems to contribute to the improvement process. To illustrate this, Table 6.4 shows the progress of the 3-stage SA in the best run for our largest task of 200 requests. The table shows the total route duration, the number of capacity violations (CV) and the number of time window violations (TWV) at the beginning of the run, and after the termination of each of the three stages. It is clear from this table how the solution progressively improves after each stage.

Table 6.4: 3-stage SA Progress for Task 200.

Stage	Route Duration	CV	TWV
Initial Solution	52969.3	64	387
After Stage 1	37768.2	0	344
After Stage 2	23841.1	0	3
After Stage 3	23841.1	0	0

Regarding the other four algorithms (GA1, GA2, SA2 and HC), it appears from Tables 6.1, 6.2 and 6.3 that the best results obtained by both versions of the GA were slightly worse than the results obtained by the random-move SA (SA2) and the HC in most test cases. The GAs, nevertheless, achieved a better feasibility ratio, and their overall average objective values were better than the average obtained by the SA2 algorithm. The tables also show that the performance of the HC algorithm was in general very good. Its overall average objective function result (shown in the last row of Table 6.3) was comparable to those obtained by the GAs, but better than the overall average objective of the SA2 algorithm. The SA2 algorithm has the worst overall average objective value, obviously due to the higher percentage of infeasible solutions obtained by this algorithm, as indicated in Table 6.1.

Similar to the 3-stage SA, the three stages applied in the HC algorithm seem to contribute to guiding the search towards better solutions. To see this, consider again the largest task of 200 requests. Table 6.5 shows the progress of the best run for this task from one stage to another, highlighting the total route duration and the number of capacity and time window violations. It is again clear from this table how each stage played its role in improving the solution it received from the previous stage.

Table 6.5: HC Progress for Task 200.

Stage	Route Duration	CV	TWV
Initial Solution	55013.0	117	387
After Stage 1	26462.9	1	1
After Stage 2	24315.9	0	0
After Stage 3	24304.3	0	0

Figure 6.1 also bears out the above observations. This graph shows the average objective value (as calculated by Equation 6.1) achieved during the 10 runs, by each of the algorithms tested. For all test cases, the graph demonstrates the superiority of the 3-stage SA, since its average objective value is the lowest compared to the other algorithms tested.

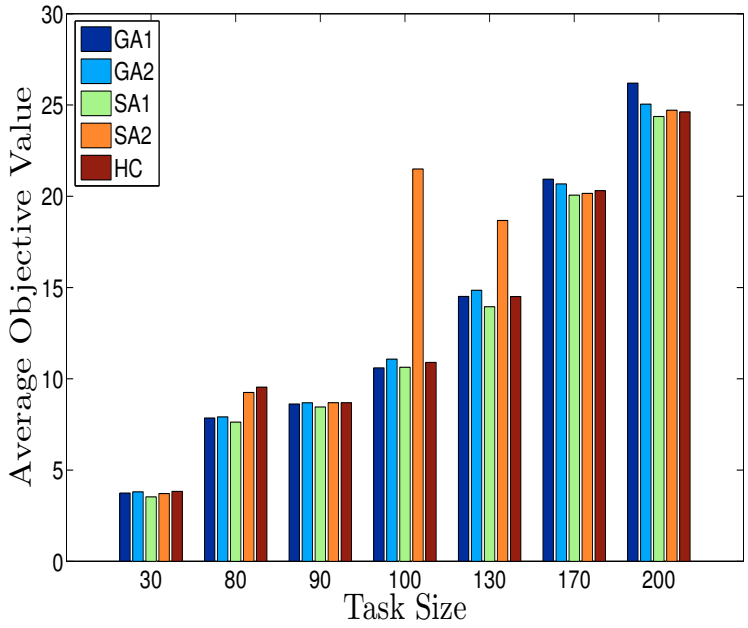


Figure 6.1: Average objective value for all algorithms.

It should be noted that infeasible solutions are seldom produced by our algorithms. However, since an infeasible solution usually has a very large objective value, the presence of one or more such solutions often results in a large increase in the average objective value (as an example, notice the average objective value produced by the random-move SA algorithm (SA2) for tasks 100 and 130 shown in Figure 6.1). Specifically, if the infeasible solution has a very large number of TW violations, its overall objective value will be very

high, given the large penalty imposed on the TW violations in Equation 6.1, and also the penalty on the amount of delay. On the other hand, a feasible solution will only have the route duration (multiplied by its small weight) as its total objective value.

Figure 6.2 shows the best run of the HC algorithm against the best run of 3-stage SA (SA1) algorithm for the largest task of 200 requests. The graph demonstrates the current objective value, as calculated by Equation 6.1, in each iteration of the run. It seems in this graph that the 3-stage SA was able to explore a wider area of the search space, albeit with a larger number of iterations needed to reach convergence.

Figure 6.3 shows the best run of the random-move SA algorithm (SA2) against the best run of the 3-stage SA (SA1), again showing the current objective value in each iteration of the run for the 200-requests task. In this graph, it is clear that the 3-stage SA was immediately directed towards lower cost solutions during the early phases of the run. The random-move SA, however, spent quite a long time during these early phases investigating low quality solutions, and only started discovering the promising areas of the search almost halfway through the run[3].

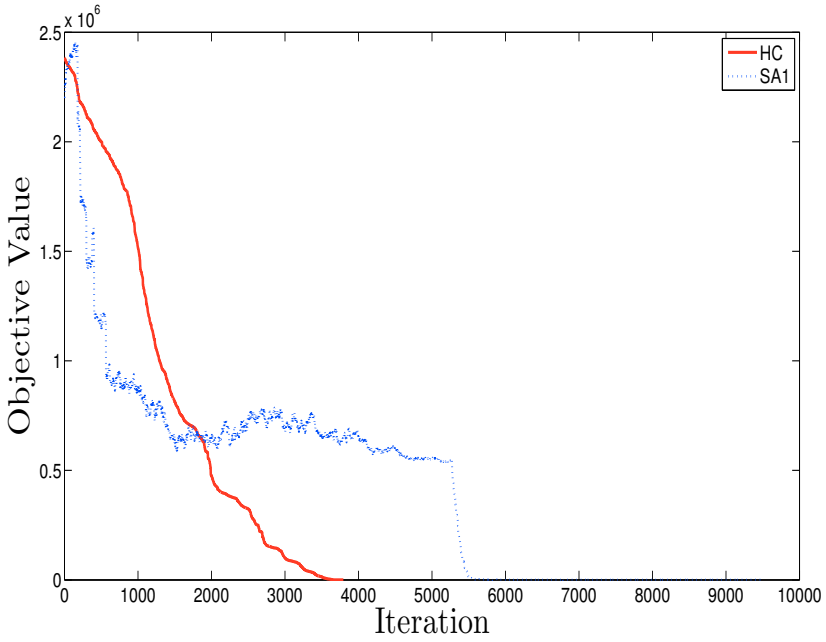


Figure 6.2: HC against 3-Stage SA (SA1) for task 200.

[3]It should be noted that the objective values only appear to go to zero in Figures 6.2 and 6.3, due to the scale of the y-axis. However, the final objective values are in fact very small. This is because the solution reached by the end of each run in both figures is a feasible solution whose cost is only measured by the duration of the route (multiplied by its small weight), according to Equation 6.1.

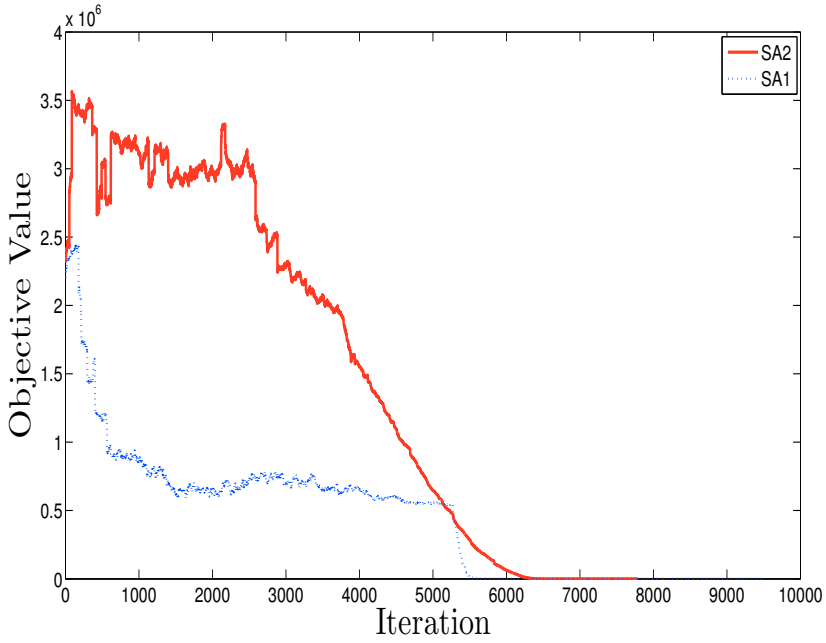


Figure 6.3: Random-move SA (SA2) against 3-Stage SA (SA1) for task 200.

Table 6.6 shows the average processing time in seconds for all the algorithms tested. The table shows that the HC algorithm has the fastest average processing time among all algorithms, in all test cases. The table also indicates that the overall average processing time of the 3-stage SA was slightly longer than the overall average processing time of the random-move SA. In addition, the table clearly indicates that both GA versions were very slow compared to all other algorithms, which is expected due to the need to maintain a large population of individuals throughout the search. The SA and HC algorithms, on the other hand, have lower computational costs due to focusing on a single solution.

Finally, comparing the results obtained by the 3-stage SA with the results obtained by previous researchers for the same problem, our algorithm seems impressive, in terms of run time as well as solution quality. We appreciate the difficulty in making such comparisons when different platforms have been used. Nevertheless, our run times are to a large extent acceptable. For example, as shown in Table 6.6, the average run time of the 3-stage SA for the 100-requests task was 44.5 seconds, while the best algorithm tried in [89] appears to have an average CPU time of approximately 300 seconds for the same task.

Table 6.6: Average Processing Time in Seconds.

Task	GA1	GA2	SA1	SA2	HC
30	92.6	42.65	5	4.9	1
80	114.5	73.15	23.8	11.7	4.3
90	121	105.65	39.1	14	5.4
100	43.55	111.9	44.5	14.4	6.2
130	422	581	26.3	30.1	9.2
170	479.7	699.4	38.8	42.1	13.8
200	1018.8	760.95	46.4	53.9	19.4
Average	**327.44**	**339.24**	**31.99**	**24.44**	**8.47**

6.6 The 3-Stage SA: Further Investigation

As mentioned above, the 3-stage SA seems to be quite a promising approach. Thus, it would be justified if we perform a thorough investigation of this algorithm by trying to examine all its potential. The choice of the sequence of the three neighbourhood moves was only based on our judgment. Starting with a move that is guided by the upper bound of the time window may give the algorithm better guidance, since it would potentially reduce the infeasibility incurred as a result of over delay. This, however, may not be the best choice. A different order could prove more beneficial in the optimization process, either in the quality or the speed of the final solution obtained. Thus, this part of our research tries to investigate all possible sequences of neighbourhood moves, by testing them on selected tasks from the data sets used in the previous part of the research. Our aim is to reach a conclusion as to whether there is any significance to the order of the underlying neighbourhood moves. The following discussion details our experimental findings when the different sequences of stages were investigated.

3-Stage SA Experimental Results:

To evaluate the performance of the 3-stage SA when different orders of neighbourhood moves are considered, we selected 4 tasks from our two data set samples. These are tasks 30, 90, 130, and 200. For each task, we ran the 3-stage SA algorithm under each possible sequence 10 times. In the upcoming analysis of results, we use a combination of the first letters of the three time window (TW) moves to denote the sequence under consideration. For example (**LEC**) denotes the sequence: **L**ate time window followed by **E**arly time

window then **C**entre time window.

Table 6.7 shows the average objective value[4], of the 10 runs achieved by the different TW orders for all tasks. Table 6.8 shows the best and worst average objective values achieved in each task, together with the sequence of moves that obtained these averages. This table also shows the best objective value achieved in each task and the sequence which obtained this best result. The last column of the table shows the previous best objective value obtained using the preliminary experimentation order (LEC), as explained in Section 6.5.

Table 6.7: Average Objective Value for all Sequences.

Task	LEC	LCE	ELC	ECL	CEL	CLE
30	3.53	3.75	**3.49**	3.70	3.57	3.78
90	**8.46**	8.62	8.55	8.52	8.48	8.75
130	**13.95**	14.03	13.97	13.96	14.22	14.19
200	24.37	24.40	24.30	**24.27**	24.32	24.49

Table 6.8: Best and Worst Results.

Task	Best Avg Obj	Worst Avg Obj	Best Obj	Prev Best Obj
30	3.49 - ELC	3.78 - CLE	2.97 - ELC	3.30 - LEC
90	8.46 - LEC	8.75 - CLE	8.23 - CEL	8.29 - LEC
130	13.95 - LEC	14.22 - CEL	12.93 - ECL	13.83 - LEC
200	24.27 - ECL	24.49 - CLE	23.43 - ELC	23.84 - LEC

As shown in Table 6.8, the sequence LEC (used in Section 6.5) achieved the best average in 2 tasks, tasks 90 and 130. On the other hand, the sequence CLE achieved the worst average objective in 3 tasks, 30, 90 and 200, while the sequence CEL achieved the worst average for task 130. Note that both sequences that obtained the worst averages start with the centre of the time window, while all sequences that obtained the best averages started with either ends of the TW interval. Possibly, adopting the centre of time window as the starting phase may not be the best choice for our algorithm. This deduction, however, still needs further evidence, specially when we notice in Table 6.8 that the sequence CEL was able to achieve the best solution among all sequences for task 90. For the remaining three

[4] As calculated by Equation 6.1.

tasks, 30, 130 and 200, the sequences that obtained the best results all started with a move that depends on the early TW bound. Apart from the above observations, there seems to be no significant difference between the averages obtained by the different sequences. This can also be seen in Figure 6.4, which shows graphically the same information as Table 6.7. The graph shows that all sequences obtained more or less comparable averages in all test cases.

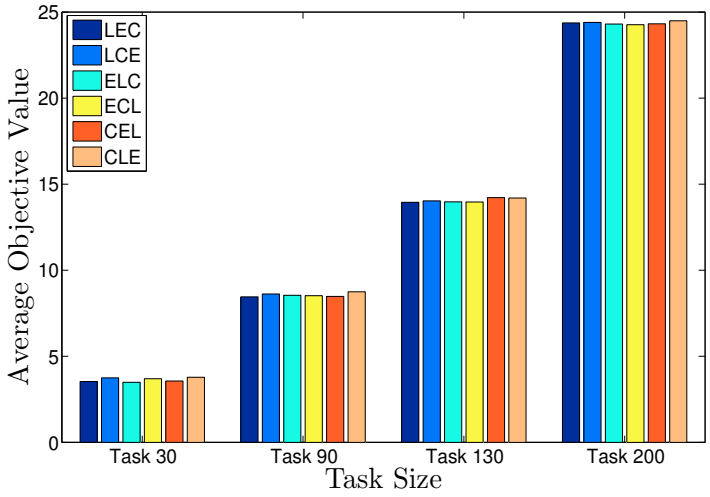


Figure 6.4: Average objective value for all sequences .

Also, observing the resulting feasibility ratio of all sequences, we found that the feasibility ratio obtained by all sequences was 100% for all tasks, with only one exception, the sequence CLE. This sequence had an 80% feasibility for task 30, and a 90% feasibility for task 90. Again this may provide an additional evidence that starting with the centre of the time window may, in some cases, drive the algorithm away from the promising solutions. Possibly, the middle of the time window is not a very good indicator of how urgent the request is. Thus, starting with it may lead to a solution that does not have the best visiting order of requests and may have a large number of time window violations. This order may be difficult to change later in the run.

It can also be noticed in Table 6.8 that the best results obtained in the current experiment were better than the previous best results in all test cases. A possible indication of this is that the 3-stage SA is a stable and robust algorithm which consistently produces high quality solutions even if different orders of neighbourhood moves were employed.

Moreover, it should be noted that in most test cases the three stages seem to contribute positively to the optimization process. Each stage often added further improvement to the

result of the previous stage, although the extent of improvement obviously varies depending on several factors, such as the quality of the starting solution, the current annealing schedule, and the neighbourhood structure.

Table 6.9 shows the progress of the SA optimization process during the three stages, for the best run of each task. The table demonstrates the objective value[5], the number of capacity violations (CV), and the number of time window violations (TWV) before and after each stage of the algorithm. The sequence of TW stages that obtained these best solutions is also shown next to the size of each task. This table shows that, in some cases, for example task 90, the best solution reached by the end of one stage may be lost during the following stage. This is normal in any SA algorithm, and especially when slow cooling is employed, as we did in our first two stages[6]. Nevertheless, the final result obtained by the end of the last stage was better than the lost solution. Obviously, when the algorithm starts a fresh stage, using a new move, after the stagnation of the previous stage, it may escape a local optimum, only by passing through worse solutions.

In terms of processing speed, nevertheless, we might be able to reach a slightly better distinction. Table 6.10 shows the average processing time in seconds for the different sequences, together with the best average speed and the sequence that achieved this best average. The table shows that the sequence LCE achieved the best average speed in 3 cases, 30, 130, and 200.

Table 6.10: Average Processing Time in Seconds for all Sequences.

Task	LEC	LCE	ELC	ECL	CEL	CLE	Best Average	Sequence of Best
30	5.0	1.5	29.2	5.0	5.6	1.8	1.5	LCE
90	39.1	21.6	18.9	18.1	19.7	15.9	15.9	CLE
130	26.3	18.0	45.8	36.9	29.5	33.4	18.0	LCE
200	46.4	33.7	85.1	73.1	63.4	37.7	33.7	LCE

[5]The objective value shown in Table 6.9 is calculated by Equation 6.1. Notice that this value is very large for the initial solution, due to the presence of a very large number of capacity and TW violations, which are penalized in the objective function. When the violations are removed by the end of the run, though, the remaining objective value will only be the route duration multiplied by the weight assigned to it.

[6]See the discussion in Section 6.3.

Table 6.9: Solution Progress through Stages.

Task	Initial Solution			After Stage 1			After Stage 2			After Stage 3		
	Obj	CV	TWV	Obj	CV	TWV	OBJ	CV	TWV	OBJ	CV	TWV
30 - ELC	18689.70	4	48	2.97	0	0	2.97	0	0	2.97	0	0
90 - CEL	258255.0	83	171	8.84	0	0	9.68	0	1	8.23	0	0
130 - ECL	792709.0	23	232	112.24	0	2	12.93	0	0	12.93	0	0
200 - ELC	2345240.0	130	382	696.82	0	7	23.44	0	0	23.43	0	0

6.7 Summary and Future Work

In this part of our research, we investigated three different approaches to the SV-PDPTW. First, a GA approach equipped with problem-specific genetic operators was implemented. One GA version had a TW directed merge crossover working together with a random swap mutation, while the other depended only on an intelligent directed mutation that is guided by the TW. Second, two versions of SA were tested: the first operated in three stages and adopted neighbourhood moves that are guided by the time window, and a second adopted a random unguided neighbourhood move. The third approach is an HC heuristic which also operated in a manner similar to the 3-stage SA, but only accepted better solutions encountered during the search.

The experimental results indicated that both GA versions had more or less comparable results in terms of both quality and processing speed. This could only indicate that the directed mutation operator is an intelligent operator that can guide the search towards better solutions even without the help of any crossover, which is usually the main GA operator. However, the performance of the GA in general was slightly inferior to the other algorithms tested in this research, in terms of the quality of the best solution in most test cases, but more so in terms of processing speed.

On the other hand, the 3-stage SA seems to be superior to all other algorithms tested in this research in the quality of the solutions obtained. Moreover, its results were better than the best results from previous research on the problem, both in terms of quality and speed, in all test cases. However, this SA version was slightly slower than the random-move SA version and the HC heuristic tested. The success of the 3-stage SA is possibly due to its dependence on intelligent neighbourhood moves that contributed to guiding the search towards better solutions. These neighbourhood moves were also successful in the context of hill climbing but with a less dramatic effect than their effect in the context of simulated annealing, possibly due to the trap of local optima. However, using hill climbing can still give us good quality solutions in a very short processing time, which may be preferable in real world applications, and if the algorithm is used repeatedly, as usually done when the problem is generalized to the multiple vehicle case.

We also further investigated the 3-stage SA algorithm by examining all possible time window sequences and analyzing their outcome on selected instances. The results in general indicated that the 3-stage SA is a stable algorithms that usually produces high quality solutions under various orders of stages. Nevertheless, all sequences seem to produces comparable average results in most test cases. It was not possible to reach a definite conclusion as to whether one or more sequences are preferred during the application of

the SA algorithm. Some results, though, indicated that the initial phase should probably start with either bounds of the TW interval, rather than its centre. In some cases, this will probably provide the algorithm a push ahead towards more promising solutions. In terms of processing speed, however, the sequence LCE seems to have the fastest average speed in most test cases.

The ideas introduced so far for handling the SV-PDPTW problem constraints can be incorporated in any heuristic or meta-heuristic approach that tackle this problem, and also its multiple vehicle variant, as will be shown later in this thesis. The robustness and portability of these simple tools have been demonstrated on several occasions within the current research.

Our next plan is to broaden our scope and investigate the more general Multiple Vehicle Pickup and Delivery Problem with Time Windows (MV-PDPTW). The work done so far for the single vehicle case will be integrated within our upcoming study. The details of this investigation are presented in the next two chapters.

Chapter 7

New Solution Construction Heuristics for the Multiple Vehicle Pickup and Delivery Problem with Time Windows

The Multiple Vehicle Pickup and Delivery Problem with Time Windows (MV-PDPTW) is a generalization of the SV-PDPTW introduced in Chapters 5 and 6. Thus, the problem definition in Section 5.2 applies to the MV-PDPTW except that we assume here the availability a homogenous fleet of vehicles, with identical capacity, to serve the requests. In addition to the precedence, capacity and time window constraints, applied to the single vehicle case, the multiple vehicle case has an added *coupling constraint*. This constraint ensures that the pickup location and its corresponding delivery are served by the *same* vehicle. Also, the objective function of the MV-PDPTW usually tries to minimize the number of vehicles used in the solution, as a primary objective, followed by minimizing the total traveling distance and/or the total schedule duration. Figure 7.1 shows a *simplified* representation of a small instance of this problem before and after solving it.

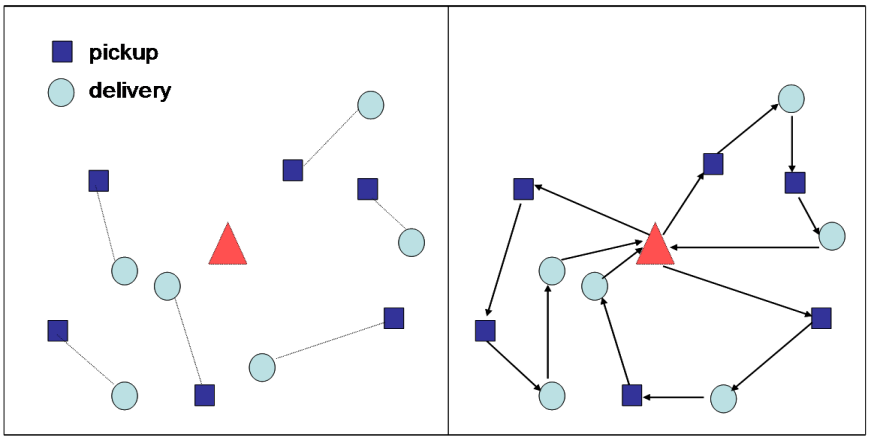


Figure 7.1: MV-PDPTW before solution (left) and after solution (right).

The MV-PDPTW is known to be $\mathcal{NP}$-*hard* [131], and the presence of many constraints makes the problem particularly complicated. Indeed, generating feasible and good quality solutions to the problem in a reasonable amount of time is often a hard challenge for researchers. The MV-PDPTW is both a ***grouping problem*** (assigning requests to vehicles), and a ***routing problem*** (finding the best route for each vehicle). Thus, an intelligent solution methodology should be able to handle these two aspects efficiently. Similar to other vehicle routing problems, researchers usually try to solve the problem in two stages: the first stage is *solution construction*, while the second is *solution improvement*. This chapter is dedicated to the solution construction phase of the MV-PDPTW, while our investigation of the solution improvement phase is detailed in the next chapter. The findings of this part of our research were published in the *MIC2009* conference [81].

The rest of this chapter is organized as follows: Section 7.1 gives a general idea about solution construction for the MV-PDPTW and introduces our upcoming research in this area. Section 7.2 summarizes some related work. Section 7.3 emphasizes the motivation and the objectives of this part of our research. Section 7.4 explains the routing algorithm embedded within the different construction heuristics suggested. Section 7.5 details the construction heuristics implemented in this research. Section 7.6 reports the experimental results of the algorithms tested. Section 7.7 sheds light on some implementation issues and complexity analysis of the suggested algorithms. Finally, Section 7.8 concludes this chapter with our future plans.

7.1 Solution Construction for the MV-PDPTW

Like the construction heuristics for the VRPTW, a solution construction algorithm for the MV-PDPTW usually selects at each iteration an un-assigned customer whose insertion is predicted to cause the least increase in the overall cost of the solution. The selected customer is then inserted in its best (least cost) feasible insertion position found among all available routes. This kind of insertion may require multiple calculations to estimate the effect of the insertion, in terms of the increase in travel distance and time delay, on all customers already existing in that particular route who could be affected by the insertion (see Section 3.3 for more details about construction heuristics). Additional decisions during the construction of the solution include whether to build routes sequentially or in parallel, and possibly the selection of seed customers to initialize the routes. Some construction algorithms order customers before the insertion, and the initial order is also an important factor that may affect the quality of the generated solution. Common approaches include sorting customers according to the distance from the depot, or according to the time win-

dow. Finally, the selection of a cost function to assess the quality of the whole solution[1] during the construction is sometimes needed, so that insertions which greatly affect the solution cost can be identified and appropriately handled.

While these considerations also apply to the general VRPTW, where all requests are of the same type (either pickups or deliveries), the MV-PDPTW in itself entails additional considerations. This is due to the presence of a pair of related locations for each individual request and the precedence and coupling issues resulting thereof. For example, the decision regarding the best insertion position for a certain request should ideally take both the pickup and the delivery into account. The sorting criteria for requests may likewise be based on either the pickup or the delivery location, or perhaps combine both. It is also frequently the case with the MV-PDPTW that the initial solution is drastically changed during the improvement phase. For example, both the algorithms in [12] and [131] produced very good results using an approach that is based on a Large Neighbourhood Search (LNS). The algorithm removes and then relocates a large number of requests (30% - 40%) in each iteration. This could possibly indicate that sophisticated construction algorithms, that are usually time consuming, parameter dependent, and hard to implement, may not actually merit their cost, compared to more straightforward and faster algorithms.

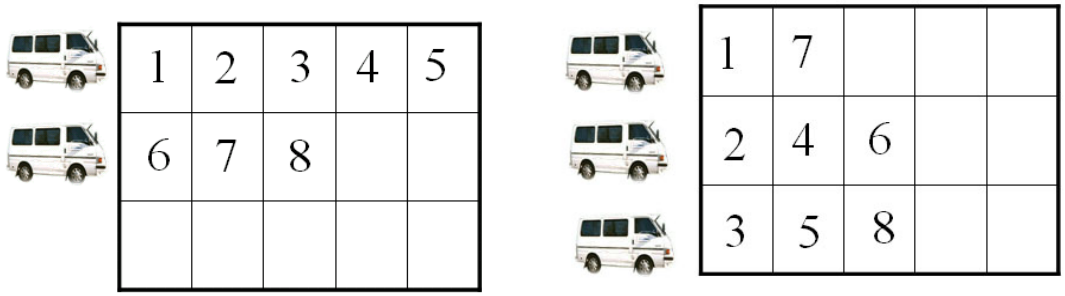
In an attempt to overcome the difficulties inherent in the construction of a feasible solution, which are mainly due to the hard problem constraints and the complex problem-specific decisions, we propose, in this part of our research, four different construction heuristics that aim to build initial feasible solutions to the MV-PDPTW. To the best of our knowledge, our research is the first attempt in the literature to analyze and compare different initial solution construction methods for the MV-PDPTW. The aim of the research is to decide which construction algorithm has more potential as a preliminary step towards a complete solution methodology to the problem. A promising construction algorithm should demonstrate a suitable balance between quality of the generated solution, processing speed and simplicity of implementation.

All suggested algorithms make use of a simple and efficient routing algorithm to generate feasible individual vehicle routes. This routing heuristic was based on our investigation of the SV-PDPTW, as previously detailed in Chapters 5 and 6. These algorithms, nevertheless, differ in whether the construction of vehicle routes is performed sequentially or in parallel. They also differ in the criteria according to which the next un-routed request is selected for insertion in a particular route.

[1] For example, the cost function may combine the number of routes and the total cost of each route.

7.2 Related Work

As previously mentioned in Section 3.3, solution construction, for vehicle routing problems in general, can be done either sequentially or in parallel. A sequential construction builds routes one after another, while a parallel construction builds a number of routes simultaneously. Figure 7.2(a) shows a typical solution, consisting of 8 locations, that is constructed sequentially, while Figure 7.2(b) shows a solution with the same number of locations that is constructed in parallel.



(a) Sequential construction (b) Parallel Construction

Figure 7.2: Solution Construction.

To construct initial solutions for the MV-PDPTW sequentially, researchers usually adapted Solomon's sequential insertion heuristics of the VRPTW [141]. A weighted sum of the extra travel distance and total time delay resulting from the insertion is often used to estimate the cost of the insertion (see Section 3.3.3 for more details about Solomon's insertion heuristics). This type of construction was used by [100] for the MV-PDPTW, and was followed by a solution improvement phase called a tabu-embedded simulated annealing.

As previously mentioned in Section 3.3.3, a parallel construction heuristic was first introduced in [121] for the VRPTW. In a parallel construction, several routes are initialized with seed customers and requests are subsequently inserted into any of the initialized routes. Accordingly, the algorithm needs an initial estimate of the number of vehicles to be used. Routes are later added as needed if the initial estimate does not yield a feasible solution. The authors also introduced an additional complex measure in the cost function, which is called "a generalized regret value", in order to compare the difference between the cost of an immediate insertion versus a postponed insertion. Requests with a large regret value are given priority of insertion. The regret value is a kind of a 'look-ahead' measure to estimate the difference between inserting the request in its best route and inserting it in its second best route. The measure may be generalized by comparing the

insertion costs of the best k routes of a particular request, rather than only the best two routes. In a sense, we choose the insertion (of a request) that we will 'regret' most if it was not performed now. This regret measure was also used by [131] for the MV-PDPTW, and was embedded within an Adaptive Large Neighbourhood Search (ALNS) technique to improve the solution quality.

The work in [102] presents a sequential construction algorithm for the MV-PDPTW. The algorithm repeats a cycle of three components. The first component is a constructor, which uses a sequential greedy algorithm to add pairs of customers in the order they appear in a priority sequence that is initially random. The analyzer afterwards analyzes the solution and assigns a certain 'blame' value for each customer based on its contribution to the total solution cost. Finally, the prioritizer reorders the customers, such that customers with a high blame value are moved forward in the priority sequence.

A parallel construction heuristic that solves the MV-PDPTW is presented in [104]. The algorithm starts by finding the largest set of customers, where it is impossible to serve any two customers with the same vehicle due to constraint violation. These customers are then used as seed customers in the initial set of routes. To insert the remaining customers afterwards, the algorithm takes into consideration the effect of insertion on both the classical increase in distance measure, and also the remaining time window slack in the route, i.e., priority is given to insertions that do not use much of the available time slack, allowing for more feasible latter insertions. The authors also use a non-standard measure of the visual attractiveness of the route to select the most desired insertions, by trying to minimize the number of crossings (intersection points) between the generated routes.

7.3 Research Motivation and Objectives

We noticed during our literature survey of the MV-PDPTW that researchers who adopt a 2-phase approach (i.e., construction of an initial solution, followed by an improvement phase) to solving the problem often pay more attention to the solution improvement phase, such that the results of the initial solution construction phase are seldom reported, if at all. This makes it difficult to assess the contribution of the construction method to the success or the failure of the overall algorithm. It is also important to note that the role of the construction algorithm is not only limited to the initialization phase. The construction algorithm is often applied at various stages during the improvement phase to create or modify new or partial solutions, as done for example in [12] and [116]. Therefore, a good choice of the construction algorithm is vitally important.

The research reported in this part of the thesis will help identify the construction heuristic(s) that seems to be most appropriate for this problem, and decide whether sophisticated and computationally expensive methods actually warrant their cost, as opposed to other simpler and less expensive algorithms. In the following section we explain our simple routing algorithm, which is the core of the different construction algorithms proposed in this research. Section 7.5 then discusses in detail these construction algorithms.

7.4 The Routing Algorithm

A crucial part of the MV-PDPTW is the routing algorithm that will generate a feasible route for each individual vehicle. A major concern is how to handle all problem constraints efficiently. The routing algorithm we used here was selected based on our research on the SV-PDPTW, detailed in Chapters 5 and 6. This algorithm relies on an *iterative improvement* of individual routes, and is embedded in the overall constructive algorithm that could either be sequential or parallel. The main difference between our routing algorithm and other routing (insertion) heuristics in the literature is that our algorithm does not try to find the best insertion position for each request in the route, but accepts *any* feasible insertion. As a result, many complex calculations and problem-specific decisions, that are related to the association between the pickup and the delivery, can be avoided. For example, our algorithm eliminates the bias towards either the pickup or the delivery location, which is one of the major drawbacks of 'classical' insertion methods. Clearly, when the best insertion position for one location (pickup or delivery) is chosen first, the choices available for its partner will be restricted accordingly.

Based on the representation attempted in our SV-PDPTW heuristics, rather than representing the visiting order of locations in each route by a one-dimensional permutation of all the different locations, we assign the same code to both the pickup location and its delivery, and we refer to the pair as a *request*. We then rely on a simple decoder to always identify the first occurrence of the code as the location of pickup and the second as the location of its delivery. This representation will handle both the precedence and the coupling constraints of the MV-PDPTW, since it will be no longer necessary to ensure that both the pickup location and its delivery are assigned to the same vehicle, and that the pickup comes before the delivery in the visiting order. On the other hand, the capacity and time windows constraints may be violated in this representation, but they are penalized in the objective function, as will be explained shortly (see Equation 7.1 below).

Recall from Chapter 5, that the different routing algorithms for an individual vehicle tried to improve the current route by rearranging nodes based on their time window intervals.

For the purpose of the MV-PDPTW, we selected as a route improvement heuristic, the simple Hill Climbing (HC) algorithm which tries to gradually modify the current route until no further improvement is possible. In fact, we chose the HC algorithm to improve individual routes, because it was very fast compared to the other heuristics investigated for the SV-PDPTW, and it also gave good quality results. As such, it seemed more suitable for repeated application within the various construction algorithm, as will be explained in Section 7.5. Also, only one stage of time window improvement was applied here to accelerate the generation of multiple routes. The neighbourhood move used by the HC algorithm in this part of our research depends only on the upper bound of the time window interval to decide the swapping of locations. Algorithm 7.1 describes this simple heuristic.

Algorithm 7.1: The HC Routing Algorithm.

1: Given a route r
2: **repeat**
3: **for** (Each possible pair of locations in r) **do**
4: **if** (The latter location is more urgent in its upper time window bound) **then**
5: Swap the current two locations in r to get a new route r'
6: $\Delta \leftarrow cost(r') - cost(r)$
7: **if** ($\Delta < 0$) **then**
8: $r \leftarrow r'$
9: **until** (Done){Stop if no improvement has been achieved in the previous pass}

The cost function to evaluate the quality of a route r, in Step 6 of the HC algorithm, is described by the following equation:

$$F(r) = w_1 \times D(r) + w_2 \times TWV(r) + w_3 \times CV(r) \ , \tag{7.1}$$

where $D(r)$ is the total route duration, including the waiting time and the service time at each location. $TWV(r)$ is the total number of time window violations in the route, and $CV(r)$ is the total number of capacity violations. The constants w_1, w_2, and w_3 are weights in the range [0,1], and $w_1 + w_2 + w_3 = 1.0$. As previously mentioned, the largest penalty should be imposed on the time window violations, in order to direct the search towards more feasible routes. We used the following weights for the route cost function: $w_1 = 0.201$, $w_2 = 0.7$ and $w_3 = 0.099$ [2].

[2]Comparing the current weights to those assigned in Equation 5.1, the penalty on the capacity violations was reduced, while the weight assigned to the total route duration was increased. This is due to the fact that routes in the multiple vehicle case are usually shorter than routes in the single vehicle case, and the satisfaction of the capacity constraint tends to be easier as a result.

7.5 Solution Construction Heuristics

In all our construction heuristics we first start by sorting customers according to the distance from the depot (farthest first). However, since in our approach we deal with customers in pairs, where each pair consists of a pickup location and its associated delivery, the distance measure, in relation to the depot, could either be the distance between the depot and the pickup location, or the distance between the depot and the delivery location. We arbitrarily chose the distance separating the depot and the delivery location for the initial order of requests.

7.5.1 The Sequential Construction Algorithm

The sequential construction heuristic tries to build routes one after another. Requests are taken one by one in order, and each request (pickup and delivery pair) is initially inserted at the end of the current route. Our HC routing heuristic (Algorithm 7.1) is then called to try to improve the current route. If the HC algorithm returns an improved route that can 'feasibly' accommodate the newly inserted pair, this insertion is accepted and we move on to the next request. However, if the improved route is still infeasible, the newly inserted pair is removed from the current route to wait for another insertion attempt in a new route[3]. Thus, unlike the 'traditional' insertion methods, our algorithm relies on the HC heuristic to improve the quality of the current route, without actually having to calculate the cost of each and every possible insertion position in order to select the best one among them.

Algorithm 7.2 describes the sequential construction procedure. It is important to note in Step 7 of this algorithm that, besides overcoming the precedence and the coupling issues, inserting a request (a pickup and delivery pair) at the end of the route has the added advantage of accelerating the insertion process, since two locations instead of one are simultaneously inserted.

[3] Although the HC algorithm is usually very successful in 'deciding' whether a new pair can be feasibly inserted in the route, there is still a small chance that the wrong decision is made. In other words, a pair may be removed from the route although it could have been feasibly inserted there, had the cost of all possible insertion positions been calculated and the best position identified, as done in 'traditional' construction methods.

Algorithm 7.2: The Sequential Construction.

```
1: Let M ← 0 {M is the number of vehicles used}
2: repeat
3:    Initialize an empty route r
4:    M = M + 1
5:    for (All unassigned requests) do
6:       Get the next unassigned request i
7:       Insert request i at the end of the current route r
8:       Call the HC routing heuristic (Algorithm 7.1) to improve r
9:       if (r is a feasible route) then
10:         Mark i as inserted
11:      else
12:         Remove i from r
13: until (All requests have been inserted)
```

7.5.2 The Parallel Construction Algorithms

As mentioned previously, for a parallel construction, several routes are considered simultaneously for inserting a new request, and an initial estimate of the number of vehicles is required. Potvin and Rousseau in their parallel construction algorithm for solving the VRPTW [121], estimate the initial number of vehicles by first running Solomon's sequential construction [141]. The number of vehicles in the resulting solution is then used as an estimate of the initial number of vehicles for the parallel heuristic.

In our research, we adapted the parallel construction heuristic of the VRPTW in [121] to the MV-PDPTW. However, to avoid extra processing time, we estimated the initial number of vehicles using a simple formula that divides the total demand of the pickup requests in the problem instance by the capacity of the vehicle, as shown in Equation 7.2.

$$M = \lfloor (\sum_{i \in N^+} q_i)/C \rfloor \ , \tag{7.2}$$

where M is the estimated initial number of vehicles, N^+ is the set of pickup customers, q_i is the demand (load) of a pickup request, and C is the capacity of the vehicle. However, this estimate seems to be more suitable for instances with a critical schedule horizon, i.e., those having a short time window width. Instances with more flexible (long) time window intervals, on the other hand, may require fewer vehicles to start with. More specifically, when the width of the time window interval is short, there will be a limited number of alternative feasible locations for scheduling the request. Accordingly, instances with such

a property usually need a large number of vehicles to be able to serve all requests without any constraint violation, as opposed to instances having a long time window width. Thus, to be able to handle the different types of test cases, we introduced a small modification to this formula for some problem instances, as will be explained in Section 7.6.

Similar to the parallel approach for the VRPTW in [121], which initializes each route with a seed *customer*, our parallel algorithms initialize each route with a seed *request* (pickup and delivery pair) from the sorted list of requests. We then take the remaining requests in order and attempt to insert the next request in one of the partial routes created. If the next request cannot be feasibly inserted in any of the already created routes, a new route is added to accommodate this request. This process is repeated until all requests have been inserted.

As previously explained, traditional parallel construction algorithms for both the VRPTW and the PDPTW, usually select the customer who has the current minimum insertion cost among all remaining un-routed customers to be inserted next. This cost is often a measure of the extra travel time and distance, which would result from inserting the customer in the best possible (feasible and minimum cost) insertion position found in all available routes. Our parallel algorithms, on the other hand, differ among each other in how they select the next request to be inserted, and also in selecting the route in which this request will be inserted. Following is an explanation of the different parallel construction algorithms proposed in our research.

Parallel Construction - First Route:

In our first parallel construction algorithm, the next request in order is inserted in the first route in which a feasible insertion of this request is found, i.e., no attempt is made to find the best route for the current request. Thus, our first parallel construction uses a fast first acceptance criterion for insertion. Algorithm 7.3 describes this procedure.

Algorithm 7.3: Parallel Construction: First Route.

```
1: Calculate M (the initial estimate of the number of vehicles)
2: Initialize M routes with seed customer pairs from the sorted list of customers
3: for (All remaining unassigned requests) do
4:    Get the next unassigned request i
5:    r = 0 {start with the first route}
6:    while ((r < M) and (i not yet inserted) ) do
7:       Insert the request i at the end of the current route r
8:       Call the HC routing heuristic (Algorithm 7.1) to improve r
9:       if (r is a feasible route) then
10:         Mark i as inserted
11:      else
12:         Remove i from r
13:      r = r + 1
14:   if (i was not inserted) then
15:      Initialize a new route r'
16:      M = M + 1 {increase the number of vehicles}
17:      Insert the request i in the new route r'
18:      Mark i as inserted
```

Parallel Construction - Best Route:

In our second parallel construction algorithm, the next request in order is inserted in the best route in which a feasible insertion of this request is found. The best route for each request is the route that causes the least increase in the overall cost of the solution (the routing schedule) due to the insertion process. To calculate the overall cost of the solution, we used an objective function that is suggested by Bent and Hentenryck in [12]. The objective function consists of three components: the first component tries to minimize the number of vehicles used in the solution, the second component tries to minimize the total distance traveled, while the third component is a measure that tries to maximize the square of the number of nodes visited by each vehicle. This last component is intended to favour routes that are rather full and those that are rather empty, as opposed to an even distribution of nodes among routes. The idea is to try to get rid of some vehicles that are under-utilized during subsequent route improvement phases. The objective function of a solution S is described by Equation 7.3.

$$O(S) = \alpha \times M + \beta \times \sum_{r \in S} Dist(r) - \gamma \times \sum_{r \in S} |r|^2 \ , \tag{7.3}$$

where M is the number of vehicles used in the current solution, $Dist(r)$ is the total distance traveled by each vehicle, and $|r|$ is the number of nodes visited by each vehicle. The constants $\alpha,\ \beta,\ and\ \gamma$ are weights in the range $[0, 1]$ assigned to each term in the objective function, and $\alpha + \beta + \gamma = 1.0$. In our research we try to minimizing the number of vehicles as our primary objective followed by the total distance, thus we chose $\alpha > \beta > \gamma$. We used the following weight values for Equation 7.3: $\alpha = 0.7$, $\beta = 0.29$ and $\gamma = 0.01$. Algorithm 7.4 describes the second parallel construction algorithm.

It is important to note, in Step 10 of Algorithm 7.4, that since the insertion process only affects one route, the calculation of the new solution cost does not require evaluating all routes in the current solution. The calculation is simply done by removing the old cost of the current route (before the insertion), and adding the new cost resulting from the insertion.

Algorithm 7.4: Parallel Construction: Best Route.

1: Calculate M (the initial estimate of the number of vehicles)
2: Initialize M routes with seed customer pairs from the sorted list of customers
3: **for** (All remaining unassigned requests) **do**
4: Initialize $LocalMin$ to an arbitrary large value
5: **for** $(r = 0; r < M; r++)$ **do**
6: Get the next unassigned request i
7: Insert the request i at the end of the current route r
8: Call the HC routing heuristic (Algorithm 7.1) to improve r
9: **if** (r is a feasible route) **then**
10: calculate $\Delta cost$ {$\Delta cost$ is the change in solution cost due to the insertion (where cost is estimated using Equation 7.3) }
11: **if** ($\Delta cost < LocalMin$) **then**
12: $LocalMin = \Delta cost$
13: $r^* = r$ { r^* is the current best vehicle for request i}
14: Remove i from r {temporarily remove i until the insertion cost of the current request in all routes is calculated}
15: **if** (r^* is found) **then**
16: Insert i in r^*
17: Mark i as inserted
18: **else**
19: Initialize a new route r' {Since no feasible insertion is found for i in any of the available routes, allocate a new route}
20: $M = M + 1$ {increase the number of vehicles}
21: Insert the request i in the new route r'
22: Mark i as inserted

Parallel Construction - Best Request:

Our next parallel construction heuristic does not only try to find the best route for each request, but also selects the best un-routed request to be inserted next. The best un-routed request is the one whose insertion (in its best route) causes the least increase in the overall cost of the solution, where the solution cost is again evaluated using Equation 7.3. Algorithm 7.5 describes this procedure.

Algorithm 7.5: Parallel Construction: Best Request.

Calculate M (the initial estimate of the number of vehicles)
Initialize M routes with seed customer pairs from the sorted list of customers
repeat
  Initialize $GlobalMin$ to an arbitrary large value
  for (All remaining unassigned requests) **do**
    Initialize $LocalMin$ to an arbitrary large value
    for ($r = 0; r < M; r++$) **do**
      Get the next unassigned request i
      Insert the request i at the end of the current route r
      Call the HC routing heuristic (Algorithm 7.1) to improve r
      if (r is a feasible route) **then**
        calculate $\Delta cost$
        if ($\Delta cost < LocalMin$) **then**
          $LocalMin = \Delta cost$
          $r^* = r$ { r^* is the current best route for request i}
      Remove i from r
    if (r^* is found) **then**
      if ($LocalMin < GlobalMin$) **then**
        $GlobalMin = LocalMin$
        $i^* = i$ {i^* is the current best request}
        $v^* = r^*$ {v^* is the best vehicle (route) for i^*}
    else
      Initialize a new route r'
      $M = M + 1$
      Insert i in the new route r'
      Mark i as inserted
  if (i^* is found) **then**
    Insert i^* in v^*
    Mark i^* as inserted
until (All requests have been inserted)

7.6 Computational Experimentation

7.6.1 Characteristics of the Data Set

To test our algorithms, we used several instances from the benchmark data of the MV-PDPTW created by Li and Lim in [100]. The authors of [100] created this data set based on Solomon's test cases of the VRPTW in [141]. There are 6 different categories of problem instances in this data set: LR1, LR2, LC1, LC2, LRC1, and LRC2. Problems in the LR category have randomly distributed customers, problems in the LC category have clustered customers, and problems in the LRC category have partially random and partially clustered customers. On the other hand, problems identified with the number '1' have a short scheduling horizon (tight time window width), while problems identified with the number '2' have a long scheduling horizon (large time window width). Each category has 6 different problem sizes: 100, 200, 400, 600, 800, and 1000 customers[4]. There are between 56-60 files from each problem size. The total number of files in the data set is 354. The data and the best known results can be downloaded from `http://www.sintef.no/Projectweb/TOP/Problems/PDPTW/Li--Lim-benchmark/`. For the purpose of testing our algorithms we selected the first 6 files from each category for each problem size. The total number of files used to test our algorithms is 216. The files used for testing our algorithms are summarized in Table 7.1.

As mentioned in Section 7.5.2, we used a simple formula (Equation 7.2) to estimate the initial number of vehicles needed for the parallel construction heuristics. However, during our preliminary experimentation, we found that this estimate does not suit the different types of problem instances. Apparently, problems that have a long schedule horizon allow for a more flexible visiting schedule, and generally require fewer vehicles. We also found during our experimentation that an underestimate of the initial number of vehicles is usually preferred to an overestimate, since reducing the total number of vehicles used is our primary concern. As a result, to estimate the initial number of vehicles for problems with a long schedule horizon (problems of category '2') we reduced our initial estimate by 50%. Thus, Equation 7.4 was used instead of Equation 7.2.

$$M = \lfloor (1/2)(\sum_{i \in N^+} q_i)/C \rfloor \ . \tag{7.4}$$

However, this estimate is to some extent arbitrary and remains under consideration for future reassessment.

[4]The original data set in [100] contained only 56 100-customers problems. Larger problem sizes were later added to the original data set.

Table 7.1: Test Files.

Category	100 customers	200 customers	400 customers
LC1	LC101 to LC106	LC1-2-1 to LC1-2-6	LC1-4-1 to LC1-4-6
LC2	LC201 to LC206	LC2-2-1 to LC2-2-6	LC2-4-1 to LC2-4-6
LR1	LR101 to LR106	LR1-2-1 to LR1-2-6	LR1-4-1 to LR1-4-6
LR2	LR201 to LR206	LR2-2-1 to LR2-2-6	LR2-4-1 to LR2-4-6
LRC1	LRC101 to LRC106	LRC1-2-1 to LRC1-2-6	LRC1-4-1 to LRC1-4-6
LRC2	LRC201 to LRC206	LRC2-2-1 to LRC2-2-6	LRC2-4-1 to LRC2-4-6
Category	**600 customers**	**800 customers**	**1000 customers**
LC1	LC1-6-1 to LC1-6-6	LC1-8-1to LC1-8-6	LC1-10-1 to LC1-10-6
LC2	LC2-6-1 to LC2-6-6	LC2-8-1to LC2-8-6	LC2-10-1 to LC2-10-6
LR1	LR1-6-1 to LR1-6-6	LR1-8-1 to LR1-8-6	LR1-10-1 to LR1-10-6
LR2	LR2-6-1 to LR2-6-6	LR2-8-1 to LR2-8-6	LR2-10-1 to LR2-10-6
LRC1	LRC1-6-1 to LRC1-6-6	LRC1-8-1 to LRC1-8-6	LRC1-10-1 to LRC1-10-6
LRC2	LRC2-6-1 to LRC2-6-6	LRC2-8-1 to LRC2-8-6	LRC2-10-1 to LRC2-10-6

7.6.2 Comparing the Construction Heuristics

Throughout this discussion, we use the following notations to refer to each algorithm

1. Sequential Construction: ***SEQ***
2. Parallel Construction - First Route: ***PFR***
3. Parallel Construction - Best Route: ***PBR***
4. Parallel Construction - Best Request: ***PBQ***

Since the construction algorithms are all deterministic, each algorithm was run only once on each test file. Table 7.2 shows the percentage of time each algorithm produced the smallest number of vehicles (Min-Vehic), and the smallest total distance (Min-Dist), that are found in the current experiment, over all 216 problem instances[5]. Table 7.3 shows the average number of vehicles, the average total distance, and the average processing time (in seconds), produced by each algorithm for each problem size separately.

[5]Some ties are produced and counted in the results.

Table 7.2: Frequency of Generated Best Solutions .

Algorithm	Min-Vehic	Min-Dist
SEQ	48%	31%
PFR	24%	6%
PBR	19%	8%
PBQ	49%	56%

The following observations can be realized from Tables 7.2 and 7.3:

- Regarding the number of vehicles generated, SEQ and PBQ produced the best results, with SEQ producing better results than PBQ in large size problems, while both PFR and PBR were slightly inferior in this respect.
- Regarding the total distance traveled, PBQ was able to beat all other algorithms, followed by PBR and SEQ.
- PFR produced the worst average distance in all test cases, but it was slightly better than PBR in the average number of vehicles used.
- SEQ and PFR have comparable average processing time in all test cases. Their processing time on average is faster than the other two algorithms, with PBQ being the slowest among all.

In summary, the results in Tables 7.2 and 7.3 suggest that PFR and PBR are inferior to SEQ and PBQ, both in terms of the number of vehicles used and the total distance traveled[6]. As a result, PFR and PBR can be eliminated from further consideration, and we can focus our attention on SEQ and PBQ.

As can be noticed from the average results in the last row of Table 7.3, SEQ produced better results than PBQ in the number of vehicles used. The PBQ algorithm, however, was able to beat the SEQ algorithm in minimizing the total distance traveled. This was obviously due to the fact that the SEQ algorithm was more concerned with fitting the largest possible number of requests in each vehicle before allocating a new one, while the PBQ algorithm relied on a cost function that has the total travel distance among its components. The PBQ algorithm was, nevertheless, much slower than the SEQ algorithm.

[6]However, PBR was able to slightly improve upon PFR with respect to the total distance traveled, while PFR was slightly better than PBR in the number of vehicles used.

Table 7.3: Average Results for all Algorithms.

Problem Size	SEQ			PFR			PBR			PBQ		
	Vehic	Dist	Time	Vehic	Dist	Time	Vehic	Dist	Time	Vehic	Dist	Time
100-customers	11.78	2662.92	0.02	11.83	2767.19	0.02	11.83	2711.89	0.03	11.69	2564.09	0.34
200-customers	17.33	8887.08	0.08	17.69	8954.06	0.08	18.17	8816.33	0.1	17.14	8132.84	3.62
400-customers	33.56	22215.14	0.32	34.64	23010.53	0.3	34.69	21898.96	0.38	33.72	19758.38	26.93
600-customers	48.22	44949.4	0.72	49.89	46644.49	0.69	50.69	45234.96	0.86	49.53	41791.82	147.57
800-customers	63.53	74650.07	1.24	65.94	77895.32	1.23	66.44	74056.45	1.65	64.89	68713.31	438.97
1000-customers	77.25	108513.19	1.88	81.97	115106.93	1.93	81.75	108662.01	2.54	81.58	103751.31	952.34
Average	**41.95**	**43646.30**	**0.71**	**43.66**	**45729.75**	**0.71**	**43.93**	**43563.43**	**0.92**	**43.09**	**40785.29**	**261.62**

The average processing time of the SEQ algorithm ranged from 0.02 seconds for 100-customers problems to 1.88 seconds for 1000-customers problems. The PBQ algorithm, on the other hand, had a processing time ranging from 0.34 seconds to 952.34 seconds for the same problem types, which indicates beyond doubt the huge difference in the computational effort needed for both algorithms.

It should also be noted that the SEQ algorithm neither requires an initial estimate of the number of vehicles, nor does it need a solution evaluation mechanism during the construction process (the SEQ is unlike the PBR and the PBQ algorithms for instance, in which the cost of the whole solution must be calculated using Equation 7.3 at each step of the construction process). The only advantage that the PBQ algorithm offers, which is a slight reduction in the total travel distance, does not seem to justify its added cost in terms of the complexity of the algorithm and the increase in processing time. Another advantage of the SEQ algorithm is that it can be easily adapted to population-based heuristics or meta-heuristics by randomizing the initial order of requests to generate different diverse solutions. The PBQ algorithm, on the other hand, is expected to produce a limited diversity, even if the initial order of requests is randomized, because of the selection criteria and the cost function it relies on during the insertion process. Most likely, requests that are hard to insert, and thus cause a large increase in the solution cost, will always remain the same, despite the change in the insertion order.

7.6.3 Comparing with Previous Best Known

Although our algorithms are primarily intended for constructing *initial* solutions to the MV-PDPTW, it would still be useful to compare our results with the best known solutions. This would give us a general idea about the expected effort in the solution improvement phase.

Table 7.4 shows the relative gap (in percentage) between the average results produced by both the SEQ algorithm and the PBQ algorithm and the average best known results. We used the following formula to calculate the relative gap

$$gap = ((Result - BestKnown)/BestKnown) \times 100 \tag{7.5}$$

The relative gap is measured with respect to both the number of vehicles and the distance[7]. The table shows that the SEQ algorithm produced, on average, a slightly smaller gap with respect to the number of vehicles, and a slightly larger gap with respect to the total

[7]For example, a gap of 50% in the average number of vehicles means that the result of the construction heuristic produced 50% more vehicles than the best known result.

distance. Together with the fact that the SEQ algorithm is quite simple and fast compared to the PBQ algorithm, the results in Table 7.4 would again seem to justify its preference as a solution construction method over the PBQ algorithm.

Table 7.4: Average Relative Distance to Best Known.

Problem Size	Vehic-Gap		Dist-Gap	
	SEQ	PBQ	SEQ	PBQ
100-customers	58%	57%	146%	137%
200-customers	63%	61%	187%	163%
400-customers	66%	67%	207%	173%
600-customers	64%	69%	206%	185%
800-customers	66%	70%	205%	181%
1000-customers	65%	74%	191%	178%
Average	**64%**	**66%**	**191%**	**170%**

It may also be beneficial to analyze the results produced by the construction heuristics for each benchmark category separately. This may give an insight into what problem types would require more effort in the solution improvement phase. Figure 7.3 shows the average gap produced by the SEQ algorithm for all tasks, organized by problem categories. Figure 7.4 shows the average gap produced by the same algorithm with respect to the distance traveled.

Both figures show that the SEQ construction heuristic seems to be more ‘successful’ in instances with a short schedule horizon, i.e., instances identified with ‘1’ in the data set, since these instances always have a smaller gap than instances of type ‘2’. Regarding the primary objective, which is the number of vehicles used, the algorithm seems to do a better job for instances that have clustered customers, as opposed to instances that have random or partially random customers. It is clear that instances in the LC category always have the smallest gap compared to other problem types. Problems with random customers and a long time window interval appear to be the most challenging for the SEQ algorithm, and possibly all solution algorithms. The reason could be that the solution space for these problems seems to be larger, due to the randomness of locations and the large width of time windows involved in this case. It also appears from both graphs that the gap in the number of vehicles is inversely proportional to the gap in the total travel distance, in most test cases.

It is also worth mentioning that the results in Table 7.4 and Figures 7.3 and 7.4 indicate

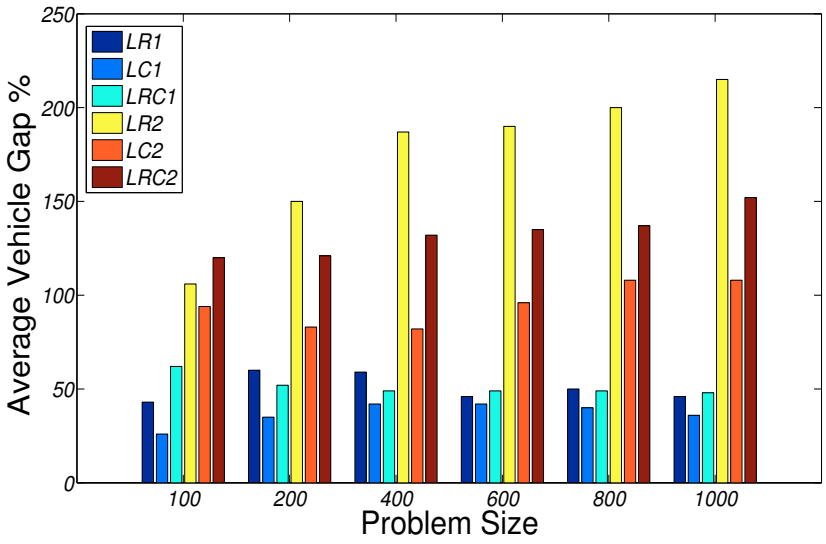


Figure 7.3: SEQ algorithm - average vehicle gap for all problem categories.

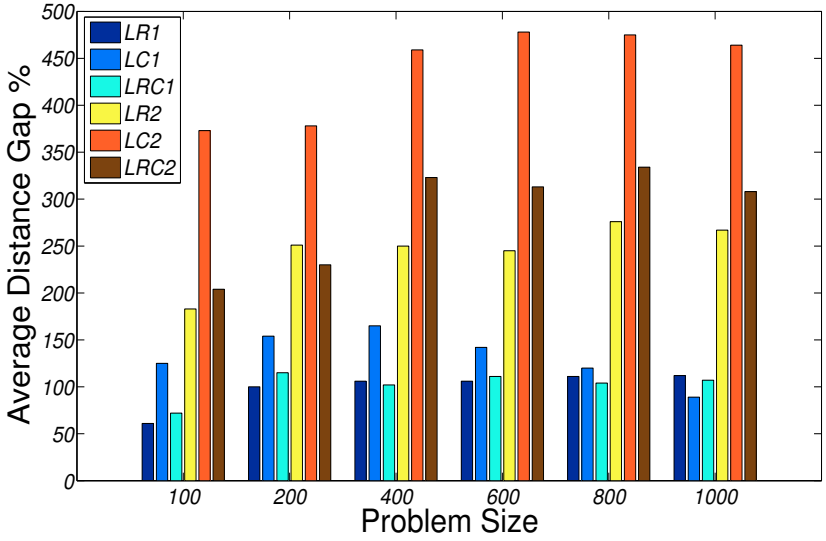


Figure 7.4: SEQ algorithm - average distance gap for all problem categories.

that a lot of work still needs to be done in the improvement phase, in order to reach the anticipated standard for the final problem solutions. This is evident by the relatively large gap between the current initial solutions and the final best known results. Designing an 'intelligent' improvement phase seems to be inevitable, in order to cope with the difficult problem constraints and the various types of problem instances.

Finally, Table 7.5 shows the average processing time of the SEQ algorithm, for each

problem size in each benchmark category. It appears in this table that problems involving random customers with a long schedule horizon, LR2 and LRC2, generally require a longer processing time than the other problem categories, which sustains our previous observation regarding the large solution space for these problems.

Table 7.5: Average Processing Time (seconds) of the SEQ Algorithm for all Tasks.

Category	Problem Size					
	100	**200**	**400**	**600**	**800**	**1000**
LR1	0.02	0.06	0.22	0.53	0.95	1.54
LC1	0.02	0.05	0.22	0.48	0.92	1.43
LRC1	0.02	0.06	0.21	0.50	0.93	1.44
LR2	0.03	0.13	0.43	1.04	1.71	2.72
LC2	0.03	0.06	0.39	0.61	1.06	1.57
LRC2	0.02	0.12	0.47	1.12	1.84	2.53

7.7 The SEQ Algorithm: Complexity Analysis and Implementation Issues

Before we conclude this part of our research, we present in this section some remarks concerning the complexity and feasibility checking of the SEQ algorithm in relation to the common construction methods. Analyzing our SEQ algorithm we find that: the routing heuristic in Algorithm 7.1 needs $O(n^2)$ time for accessing each pair of locations in the route, where n is the number of requests in the problem instance. Also, the cost function of Equation 7.1, which checks the feasibility of the whole route as well, needs $O(n)$ time. The SEQ algorithm (Algorithm 7.2) needs $O(n)$ since its major iteration processes all requests in order. This will make the run-time complexity of the whole algorithm $O(n^4)$.

For the sake of comparison, Algorithm 7.6 describes a basic construction method for vehicle routing problems in general. The algorithm appears in [24].

Algorithm 7.6: Basic Construction Algorithm for the VRP [24].

1: N = set of unassigned customers
2: R = set of routes {initially contains one route}
3: **while** $N \neq \emptyset$ **do**
4: $p^* = -\infty$
5: **for** $j \in N$ **do**
6: **for** $r \in R$ **do**
7: **for** $(i-1, i) \in r$ **do**
8: **if** $(Feasible(i,j)\ and\ Profit(i,j) > p^*)$ **then**
9: $r^* = r$
10: $i^* = i$
11: $j^* = j$
12: $p^* = Profit(i,j)$
13: $Insert(i^*, j^*)$ {insert j^* between (i^*-1) and i^*}
14: $N = N \setminus j^*$
15: $Update(r^*)$

According to [24], Algorithm 7.6 is of $O(n^3)$, provided that the *feasibility* and *profitability* can be performed in constant time. *Feasibility* makes sure that the current position adheres to all problem constraints, while *Profitability* is usually measured as a weighted combination of extra travel distance and time delay resulting from the insertion. When it is more profitable to insert a customer in a new route, a new route will be allocated. As explained in [24], a special algorithm can be applied to reduce the TW feasibility test of the VRPTW from a linear time to a constant time. The same algorithm can also be applied to the PDPTW. Nevertheless, this algorithm requires that extra information is kept for each customer already existing in the route. In general, two quantities have to be maintained: the earliest and the latest possible times the service can take place, relative to the customer's current location in the route. This information is not fixed and subject to change after each insertion, which accounts for the existence of an *update* function to maintain the desired quantities (Step 15 of Algorithm 7.6).

Unlike the VRPTW, however, checking the capacity feasibility for the PDPTW can only be done in linear time [86], due to the presence of two different types of customer services in the route. As a result, the basic construction algorithm when applied to the PDPTW also results in $O(n^4)$ complexity. The algorithm can also include a selection of seed customers for route initialization, which usually does not change the complexity of the algorithm.

As mentioned above, it is possible to reduce the TW feasibility check for the PDPTW from a linear time to a constant time, by maintaining and frequently updating extra route

information, as done for example in [37]. However, since our route cost function (Equation 7.1) tests the feasibility of both the TW and the capacity concurrently, it would be redundant to calculate and store additional service timing information to accelerate the TW feasibility test, since an $O(n)$ testing would still be needed for the capacity feasibility.

In addition, since our SEQ algorithm accepts any feasible insertion, it does not have to check the feasibility nor estimate the profitability of each and every possible insertion position, as done in Algorithm 7.6. In our algorithm, the cost of the route as a whole will be calculated, if at all, only if the route has been changed. This is due to the restriction imposed by the TW condition in Step 4 of Algorithm 7.1.

Finally, during the insertion process, i.e., Step 7 of the SEQ algorithm (Algorithm 7.2), two locations (a pickup and delivery pair) are simultaneously inserted, then Algorithm 7.1 handles the feasibility checking and the improvement of the underlying route altogether. Besides overcoming the precedence and the coupling issues, this insertion has the added advantage of accelerating the solution construction process, since only half the number of locations is processed in the main iteration of Algorithm 7.2.

On the other hand, the parallel construction algorithms implemented in this research seem to be one order of magnitude higher than the SEQ algorithm, due to the presence of an extra loop that passes through all available vehicles, although the number of vehicles is always less than n (the number of nodes in the data set).

7.8 Summary and Future Work

In this research we investigated several initial solution construction heuristics for the MV-PDPTW, aiming to identify the best heuristic that can be used as part of a comprehensive solution methodology. In our opinion, existing approaches in the literature often overlook and perhaps underestimate this vital component of the overall solution algorithm.

The experimental results on a large number of benchmark instances indicate that the sequential construction heuristic (SEQ) seems to be the most favourable solution construction method, which can be by easily embedded in a heuristic or a meta-heuristic technique to reach final good quality solutions. With just a few simple lines of code, and without a pre-determined number of vehicles or a solution evaluation mechanism, this algorithm produced good quality results, that are sometimes even better than the results obtained by the most sophisticated parallel algorithm tested in our research (the PBQ algorithm). The SEQ algorithm also had an impressive speed, with a processing time that is at most 6% of

the time needed by the PBQ algorithm, making it even more suitable for population-based solution algorithms.

The experimental results, nevertheless, show that probably a costly improvement phase is still needed to achieve final good quality solutions, as evident by the relatively large gap to best known results produced by the SEQ construction algorithm. This, however, further supports the need for a fast solution construction method to achieve an overall reasonable computation time for the complete solution algorithm.

The construction algorithms developed in this part of our research are distinguished by their simplicity and ease in coding and replication, compared to many construction methods that are adopted from the VRPTW literature. All of our algorithms are general portable frameworks that can be used within other heuristics and meta-heuristics that solve the PDPTW and its related variants.

In the next part of our research, we will start investigating the solution improvement phase for the MV-PDPTW, using the SEQ algorithm for the solution construction phase. Details of our investigation are presented in the next chapter.

Chapter 8

Two Approaches for Solving the Multiple Vehicle Pickup and Delivery Problem with Time Windows: A Genetic Algorithm and a Simulated Annealing

In this chapter we continue our investigation of the MV-PDPTW, introduced in Chapter 7. This part of our research will augment the solution construction heuristic developed in the previous chapter with an improvement method, using both a Genetic Algorithm (GA) approach and a Simulated Annealing (SA) heuristic.

This chapter is organized as follows: Section 8.1 highlights the motivation behind the research carried out in this part of the thesis. Section 8.2 provides a brief summary of some related work from the literature. Section 8.3 discusses genetic algorithms as a solution approach form the perspective of the MV-PDPTW, and introduces our specific GA suggested for the improvement phase of this problem. Within this section, we describe the solution representation and the objective function used in Section 8.3.1, how the initial population is created in Section 8.3.2, the genetic operators in Section 8.3.3, the overall GA in Section 8.3.4, some attempts to improve the results of the GA in Section 8.3.5, and finally Section 8.3.6 reports the experimental results of the GA when tested on published benchmark data and compared with previous GA attempts from the literature. After that, Section 8.4 explains the SA approach adopted for solving the problem, together with its experimental results compared to the results of the GA approach. Finally, Section 8.5 gives some concluding remarks and possible future work. Our findings, related to the GA approach, in this part of our research were published in the *MIC2009* conference [80].

8.1 Research Motivation

As previously mentioned in Chapter 2, Genetic Algorithms (GAs) are intelligent search methods that have been successfully used for solving many hard combinatorial optimization problems. In the first part of our research, though, the 3-stage SA approach performed better than the GA for the SV-PDPTW, especially in terms of processing time. Despite this, using a GA for the improvement phase of the MV-PDPTW still appeared to be promising for several reasons:

Firstly, GAs have been used for solving the related VRPTW, producing good results in many cases, for example [144], [120], [16] and [15]. Secondly, the parallel nature of GAs and its population based mechanism can probably make it more appropriate than other meta-heuristic approaches (that focus on improving only one solution), for solving very hard optimization problems. The multiple-vehicle case of the PDPTW is in fact considerably harder than the single vehicle case [137], which makes applying GAs an attractive option for solving this problem. In addition, our simple sequential construction algorithm (SEQ), developed in Chapter 7, seems to be most appropriate for population-based meta-heuristics, due to its simplicity, speed and its potential for creating a diverse population by randomizing the initial order of requests.

Finally, as previously mentioned in the introduction to this thesis (Chapter 1), the main focus of our research is on developing appropriate representations and neighbourhood moves that can help guide the search towards good quality solutions and manage infeasibility throughout the search. If the appropriate techniques are designed, they should be widely applicable, and may be used within other heuristics or meta-heuristics, for solving the underlying problem. Accordingly, we tried to follow the same approach used in our research for solving the SV-PDPTW, by first developing the representation and neighbourhood moves (represented as genetic operators here) that we believe can handle the difficult problem constraints. These operators are first tried within a GA approach for solving the MV-PDPTW, but they were also adapted and employed within an SA approach, as previously done for the single vehicle case.

8.2 Related Work

When dealing with the MV-PDPTW, some researchers use *exact* methods to solve the problem to optimality, but these are limited to small size problems. For example, the work presented in [46] is an optimization techniques which formulates the problem as a set partitioning problem, and then employs a column generation method to solve a linear

relaxation of this problem to optimality. Problem sizes of up to 55 customers and 22 vehicles were solved using this approach. Also, the work in [130] presents a new mixed integer programming formulation for both the MV-PDPTW and the dial-a-ride problem, and the formulations are solved using two branch-and-cut algorithms. New problem instances have been created, and the results of the algorithms were compared with upper bound solutions obtained by applying the heuristic in [131] to the same problem instances. Problem sizes up to 194 nodes were solved to optimality.

Other approaches adopt *approximation* techniques to deal with large size problems. As explained in the previous chapters, the solution process, in this case, often starts by constructing one or more initial solutions to the problem, usually using techniques adopted from the VRPTW construction algorithms, and then these solutions are improved using heuristics or meta-heuristics. In general, Simulated Annealing (SA) and Tabu Search (TS) have been the most popular approaches for solving the MV-PDPTW.

The work in [112] is one of the first attempts to solve the MV-PDPTW. The technique is based on a reactive tabu search which allows the tuning of the search parameters, such as a short-term memory length, based on an assessment of visited solutions during the search. The algorithm also tries to detect and escape possible local optima. The solution representation is a vector that includes customer nodes separated by vehicle nodes to which customers are assigned. The objective function consists of three components: the total schedule duration, the number of capacity violations, and the number of time window violations. Thus, during the search process, overloads and tardiness in the solutions are treated as soft constraints, while precedence and coupling constraints are strictly enforced. To construct an initial feasible solution, a predecessor-successor pair (PS) is inserted in the best feasible insertion position in the current vehicle's route. If no feasible insertion can be found, a new vehicle route is added to the solution. Three neighbourhood moves are used in this algorithm. The first move is the "Single Paired Insertion" (SPI), which tries to move all predecessor nodes to better feasible locations in other routes. Successor nodes are inserted after their predecessors in the best possible positions, although vehicle capacity or time windows may still be violated. The second move is the "Swapping Pairs Between Routes" (SBR), which first tries to exchange a pair of predecessor nodes between two different vehicles, and then exchanges their successor nodes. The last move is the "Within Route Insertion" (WRI), which tries to reorder nodes in the same route to reduce infeasibility in the solution and improve its quality. The sequence of neighbourhood moves is selected dynamically during the search, based on a calculated relationship between the average time window length (the tightness of the time window) and the average route duration.

The authors in [100] present a tabu-embedded simulated annealing approach to solve the MV-PDPTW. To construct an initial feasible solution, they modified Solomon's Insertion heuristic [141] by initializing each route with a pickup and delivery (PD) pair that satisfies a set of criteria, based on combined time window intervals and distance from the depot. Each un-routed PD pair is then inserted into the partial route in the best possible feasible and minimal cost insertion position. The objective function tries to minimize, in priority order, the number of vehicles used, the total travel distance, the total service duration, and the drivers' total waiting time. To create a neighbouring solution, three local search methods are presented. The first is a "PD-Shift" operator. In this operator, a PD pair is first removed from route 1 and then it is inserted in route 2. After that, another PD pair is removed from route 2 to be inserted in route 1. The second operator is a "PD-Swap" operator, which simultaneously removes a PD pair from each route, and then reinserts each pair in the other route. The third is a "PD-Rearrange" operator which first removes and then reinserts a PD pair in the same route. In all three moves, only feasible insertions are allowed. For a more thorough neighbourhood search, they extend their local search method to a descent local search (DLS), which tries to improve the current solution for a number of iterations. When no further improvement is possible, the DLS algorithm returns the best current solution. The main meta-heuristic algorithm is a tabu-embedded simulated annealing procedure with K-restarts, i.e., the algorithm stops when the number of iterations without improvement reaches a pre-defined value K. To prevent cycling, their simulated annealing procedure records the accepted solutions in a tabu list. The authors generated test data for the problem based on Solomon's test cases for the VRPTW [141]. The authors consider their approach as the first efficient attempt to solve practical size MV-PDPTW instances. Their generated data set also became the standard benchmark test data for the MV-PDPTW.

The work in [98] presents a two-phase method to solve the MV-PDPTW. In the first phase the algorithm creates an initial feasible solution using a strategy that combines the benefits of Solomon's classical insertion heuristic [141], and a sweep heuristic, first suggested in [58], which was particularly adapted to fit the MV-PDPTW. The authors call their insertion heuristic "Partitioned Insertion Heuristic". The objective functions is based on the number of vehicles used and the total travel distance. Their local search is based on three moves, similar to the moves described in [112], as explained above. The second phase of the algorithm is a tabu search that tries to improve the current initial solution using a set of composite neighbourhood moves.

In a relatively recent work [12], the authors present a two-stage algorithm to deal with the MV-PDPTW. The first stage uses a simple SA approach, whose focus is to minimize the number of vehicles used in the solution. The second stage uses a Large Neighbourhood

Search (LNS) strategy, first suggested in [139], to minimize the total travel cost of the entire solution. The primary objective function (used in the LNS stage) minimizes the number of vehicles first, then the total travel cost. However, according to the authors, using this objective function is more effective in reducing the total travel cost than minimizing the number of routes in the solution. Therefore, they added an SA phase to boost the performance of the overall algorithm by reducing the number of routes in the solution first. To this end, they used a different objective function in the SA stage than the one used by the LNS. The SA objective function included three components: the first component tries to minimize the number of routes. The second component tries to maximize the number of routes that are rather full and those that are rather empty, with a view to eliminating routes with few customers during the search. Finally, the third component of the SA evaluation function reduces the total travel cost[1]. The neighbourhood move adopted in their SA is a simple pickup and delivery pair relocation operator, which was also used by previous researchers (e.g. [98], [100] and [112]). The LNS of their algorithm is actually a sequence of local searches, where a group of customers are selected for relocation based on a certain relatedness measure between pickup customers. The neighbourhood generated after the relocation of customers is then exhaustively explored, using a branch and bound algorithm, to find its best solution. The best solution found replaces the current solution, if it has a better objective value. The algorithm was tested on benchmark data obtained from [100], where the results show the effectiveness of the approach since it was able to produce many new best solutions for instances with 100, 200, and 600 customers.

An interesting solution methodology is presented in [131], where a variant of the MV-PDPTW is considered. It is assumed here that the starting and ending depots of a vehicle's journey need not be the same, and the depots could be different for different vehicles. It is also assumed that the number of vehicles is limited, so it may not be possible to serve all requests, in which case the unserved requests are placed in a request bank. Thus, minimizing the number of unserved requests in the request bank is one of the objectives of the algorithm, added to minimizing the number of vehicles used and the total travel distance. It is also assumed that there could be some special requests that cannot be served by some vehicles, for example if transferring the request requires a special characteristic that is not available in some vehicles (e.g. transferring frozen food or some medical samples). The basic solution methodology in this work is a Large Neighbourhood Search (LNS) technique, used in both [139] and [12], where a number of requests are first removed then re-inserted into the solution. Unlike previous researchers, however, the authors apply a number of different heuristics for both the removal and the insertion methods.

[1] Recall that this 3-component evaluation function is the same as the cost function used by our PBR and PBQ construction heuristics (Equation 7.3), as explained in Section 7.5.2

They present three different removal heuristics. First, the "Shaw Removal" removes requests according to a certain relatedness measure, which is based on differences between distances, arrival times, loads and the numbers of vehicles that can serve these requests. Second, the "Random Removal" heuristic removes randomly selected requests. Finally, the "Worst Removal" heuristic removes requests with high costs, i.e., requests that appear to be placed in unfavourable positions in the solution. They also present two different insertion techniques to insert the removed requests in the partial routes. The *basic greedy* heuristic inserts the request with minimum insertion cost in the best possible insertion positions. On the other hand, the *regret insertion* heuristic gives priority for insertion to requests whose insertion would seem to be more costly if it was delayed[2]. To select which heuristic to use, the authors assign weights to the different heuristics and use a roulette wheel selection method. These weights are adaptively adjusted during the search based on the performance of the different heuristics employed. Accordingly, the authors call their algorithm an Adaptive Large Neighbourhood Search (ALNS), as opposed to the basic LNS search method which only applies one removal and one insertion heuristic. During the search, a newly generated solution replaces the current solution using an SA acceptance criterion. Also, similar to [12], the authors first apply a preliminary stage to minimize the number of vehicles used in the solution before applying the basic ALNS. During this stage, they create an initial solution using a sequential insertion heuristic. They then try to remove one or more routs from this solution, using the LNS algorithm, until the minimum possible number of routes is reached. The algorithm was tested on the benchmark data of [100]. In addition, new problem instances were created, which takes into account their modified variant of the MV-PDPTW. When compared with the LNS, which uses only one removal and one insertion heuristic, the ALNS seems to be superior. In addition, their ALNS was able to outperform previous heuristics applied to the MV-PDPTW. To the best of our knowledge, the LNS technique of [12] and the ALNS of [131] are the current state-of-the-art, since they have both produced best known results for many benchmark problem instances.

Besides SA and Tabu Search, GAs have been applied by some researchers for solving the MV-PDPTW and the related dial-a-ride problem. In the early GA approach by [92], the chromosome representation is based on assigning a four digit code to each location. The first digit of this code represents the number of the vehicle to which the location is assigned. Thus, both the pickup location and its associated delivery will be assigned the same first digit. The other three digits are used to sort locations according to the visiting order of the vehicle, and a pickup location always has a sorting code that is less than its corresponding delivery. The crossover operator works by generating two crossover points,

[2]See Section 7.2 for more details about the regret measure of the parallel construction heuristics.

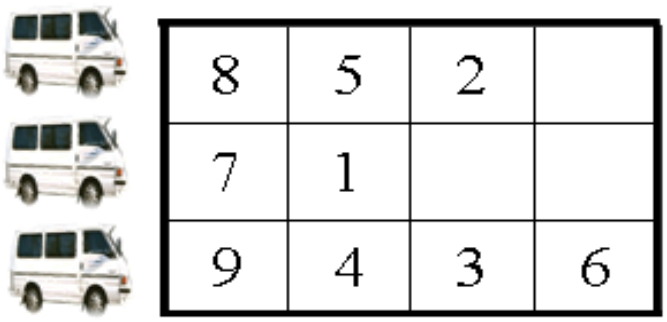


Figure 8.1: A chromosome in a GA that handles both grouping and routing for the MV-PDPTW.

and swapping the resulting segments between the two parents. The mutation operator, simply changes the first digit of a pickup and delivery pair to another digit, i.e., assigns the request to another vehicle. Another operator called vehicle merging is also used to reduce the number of vehicles. The algorithm was tested on 24 randomly generated problems with numbers of requests ranging from 5 to 30.

The authors in [33] introduce an attempt to handle the MV-PDPTW using an evolutionary algorithm, for both the grouping and the routing aspects of the problem. The solution representation is a list of vehicles routes, where each route consists of a sequence of pickup and delivery locations, and all problem constraints are enforced in the solutions throughout the search. Figure 8.1 shows how a typical chromosome may look in their suggested evolutionary approach. The objective function tries to minimize the number of vehicles in the solution, the total travel distance and the total travel time. Two crossover operators are tried, one exchanges fragments of routes between parents, while the other exchanges complete routes. Infeasible solutions that may be created following crossover are repaired by removing repeated requests or adding requests that are not served. If this is not possible, the offspring is discarded. Two mutation operators are presented, the first tries to reduce the number of vehicles in a solution by selecting a route and moving all its requests to other routes in the same solution. The second mutation tries to improve a route by rearranging its requests.

The work in [116] presents what seems to be a first attempt to apply a Grouping Genetic Algorithm (GGA) to the MV-PDPTW. In a GGA, the genetic representation is based on a set of genes, where each gene represents a group of objects rather than a single object. For the MV-PDPTW, it is assumed that each gene represents a group of requests that are assigned to one vehicle. Thus an individual solution only covers the grouping aspect of the problem. The routing aspect, on the other hand, is handled by an independent data

structure associated with each gene. Figure 8.2 is a visual representation of how a chromosome in their GGA approach may look. The objective function is to minimize the total travel distance, irrespective of the number of vehicles used. The crossover is an adaptation of the original crossover mechanism for the GGA, presented in [49], where clusters (vehicles with their assigned requests) are removed from one parent and inserted into the other parent. This is then followed by a chromosome clean up, to remove duplicate vehicles and the repeated assignment of requests, in addition to re-assigning requests that are no longer assigned. The mutation operator removes a cluster from a chromosome, and reassigns its requests to other clusters, creating additional clusters if necessary to maintain feasibility. The embedded insertion heuristic applied in several stages of this algorithm is based on inserting a request in the best feasible and minimum cost position, among all possible insertion positions in the chromosome [3]. In a recent work [126], another GGA was applied to the Handicapped Person Transportation problem (HPT), which focuses on minimizing clients inconvenience.

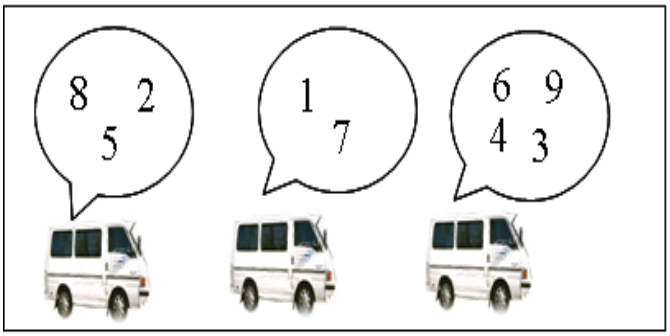


Figure 8.2: A chromosome in a GA that handles only grouping for the MV-PDPTW.

The authors in [91] deal with the dial-a-ride problem using a cluster-first route-second approach. The clustering is handled using a GA, and the routing is handled using a space-time nearest neighbour heuristic. In the clustering phase, the chromosome representation is a two-dimensional array where rows represent routes and columns represent customers. A cell values of 1 or 0 indicates whether or not a customer is assigned to a particular route. Crossover is performed by selecting two random routes from parent solutions and creating a new child route using a traditional uniform crossover. The remaining routes in the child are inherited from the first parent without change. The newly created child may be infeasible, though. Accordingly, a repair method is followed to assign unassigned customers or remove duplicate assignments of customers. The mutation operator just

[3] More details about [33] and [116] will be given in Section 8.3.6 when our algorithm is compared with their approach.

moves a random customer to another cluster (route). On the other hand, the work in [34] tries to use a GA for the whole dial-a-ride problem, i.e., for both the grouping and the routing phases. In their representation a gene consists of a vehicle-passenger pair. They used a one-point crossover and a Partial Match Crossover (PMX). For mutation they use a bit-level mutation and a 2-Opt operator.

A recent paper [37] presents an indirect search method for the MV-PDPTW. The idea is to separate the meta-heuristic search strategy from the feasibility checking routines, to simplify and accelerate the optimization process. This technique facilitates the application of simple problem-independent moves during the meta-heuristic search, while a problem-specific greedy decoder handles the construction of the corresponding feasible solution using the information (encoding) given from the meta-heuristic search engine. For example, in the PDPTW, the encoding is a permutation of transportation requests, which determines the sequence in which these requests will be scheduled. Given a certain permutation, the greedy decoder creates a feasible solution (schedule) using a sequential construction algorithm and a cheapest insertion rule. To create a neighbouring solution during the meta-heuristic search, a simple 2-exchange move is applied to the current permutation and the new permutation is supplied again to the greedy decoder to create the corresponding new schedule. A comparison of the indirect search technique with two special-purpose algorithms for the MV-PDPTW shows that the technique is competitive in both solution quality and speed. This may indicate the potential of applying the technique to other rich combinatorial optimization problems.

The above literature summary shows that the MV-PDPTW is in fact a hard problem, for which most solution methods tend to be rather complex and hard to explain and replicate. Developing an appropriate solution technique is indeed a challenge for researchers. An intelligent solution methodology should be able to handle both the grouping and the routing aspects of the problem efficiently. In addition, in both the construction and the improvement phases of the problem, the researcher is faced with many decisions that should be made. These decisions include, among others, the components of the objective function, the permissibility of infeasible solutions, how to adhere with the constraints of the problem, how to generate new solutions, and what acceptance criteria should be applied to replace the current solution during the improvement phase. It is thus often difficult to discover good quality solutions that do not violate any problem constraint, in a reasonable processing time.

8.3 A Genetic Algorithm (GA) for the MV-PDPTW

Although Genetic Algorithms (GAs) have been successfully used for solving many routing and scheduling problems, research using GAs for solving the MV-PDPTW is generally scarce. In addition, the results reported by most GA techniques attempted are often disappointing in some respects. As previously explained, the MV-PDPTW consists of two related problems: the *grouping* or the *clustering* problem tries to find the best allocation of requests to vehicles, while the *routing* problem is concerned with finding the best feasible route for each vehicle, given the requests assigned to it. When trying to solve this problem using a GA, it is often hard to tackle these two aspects simultaneously. Moreover, a major issue is finding a suitable genetic encoding and designing intelligent genetic operators that are capable of handling all the difficult problem constraints and may encourage the formation of meaningful genetic building blocks, which may help in generating individuals of better fitness [116][4]. In the MV-PDPTW, the genetic operators should be smart enough to transfer the favourable genetic traits from parent solutions to their offspring, while trying to avoid the frequent generation and evaluation of infeasible problem solutions. As previously mentioned, infeasible solutions are handled in many solution algorithms using a repair method to fix infeasibility during the search, which will inevitably increase the processing time and complicate the algorithm.

Most previous GA research, for example [116] for the MV-PDPTW and [91] for the dial-a-ride, tried to tackle the difficulties encountered in the GA encoding and operators by allowing the GA to handle only the grouping aspect. The routing aspect, on the other hand, was handled by an independent routing algorithm that is hidden from the GA and is called when a chromosome is decoded. The genetic operators in this case are usually general-purpose and do not apply any problem-specific knowledge. Attempts to use a GA for both the grouping and the routing aspects, for example [33] for the MV-PDPTW and [34] for the dial-a-ride problem, generally produced discouraging results.

We present in this part of our research our experimentation with a new GA for solving the MV-PDPTW. Our GA tries to face the challenge of handling both the routing and the grouping aspects of the problem simultaneously. Unlike the most popular approaches in the literature, in which the GA is only aware of how requests are clustered, but is not aware of how they are routed, our chromosome representation more naturally accommodates each group of requests together with suggested routes. By explicitly monitoring and manipulating all the solution information, we aim to preserve the distinctive characteristic of GAs in identifying the desirable genetic material and transferring it from

[4]For details about the "GA schema theorem" and "the building block hypothesis", the reader is referred to [64].

generation to generation during the evolutionary process. Our GA, thus, does not rely on a separate decoder for interpreting the chromosome contents and creating the subordinate routing information for each group of requests. Instead, the algorithm has a simple embedded construction heuristic that allows individual routes to dynamically change, within the chromosome itself, during the search . Also, with our solution representation in mind, we developed new simple genetic operators. Using problem-specific knowledge, such as the quality of the generated routes, these operators try to create good quality feasible solutions throughout the search. In addition, since no parallel repair method is needed to fix the infeasibility of solutions, the overall algorithm is simple and elegant, a feature often missing from most up-to-date solution algorithms.

Besides comparing our GA with the SA approach that will be introduced in the next section, we also compare here in detail our GA approach with what seems to be the only other two GA attempts in the recent literature for solving the MV-PDPTW. We highlight, based on the experimental findings, the promising aspects of our approach, and also point out to where further improvement could be achieved.

8.3.1 The Solution Representation and the Objective Function

As mentioned in the introduction to this thesis (Chapter 1), one of our goals in this research is to develop a solution representation that facilitates handling the difficult problem constraints. Following our approach for solving the SV-PDPTW, as explained in Chapter 5, we adopt a simple representation for each individual route. A route is simply a list of visited locations in order. However, when we assign requests to each route, both the pickup and its delivery location are given the same code. For more details, the reader is referred to Section 5.4.1.

In our GA, the *chromosome* represents a problem solution. It is simply a collection of individual routes. Thus, each gene is actually a complete vehicle route. Both the route (gene) length and the chromosome length are variable depending on the number of requests to be visited and the number of vehicles in the solution. Thus our representation is just a problem solution upon which the genetic operators are directly applied. Our chromosome is very much like the chromosome structure depicted in Figure 8.1, but only differs in assigning the same code to both the pickup and the delivery. Thus, the code of each request will appear twice in the chromosome, once showing where its pickup location will be visited and another for its delivery location. Figure 8.3 shows a typical chromosome in our GA.

v1	0	1	2	3	3	2	1	4	4	0
v2	0	5	5	6	6	7	7	0		
v3	0	8	9	8	9	0				
v4	0	10	11	12	11	10	12	0		

Figure 8.3: Our chromosome representation for the MV-PDPTW.

Most solution methods from the literature, for example [100], try to minimize the number of vehicles used in the solution as a primary objective, followed by either or both the total distance traveled and the total service duration. We used the following objective function of a solution S to achieve this goal:

$$O(S) = N(S)^2 \times TotDist(S) \times TotDur(S) \ , \tag{8.1}$$

where $N(S)$ is the number of vehicles used in the solution, $TotDist(S)$ is the total distance traveled by all vehicles, and $TotDur(S)$ is the total schedule duration, which includes the total travel time, the waiting time of the vehicles, and the service time at each location. The number of vehicles is squared in this objective function, so that solutions that use more vehicles will have a considerably higher cost than solutions that use less vehicles. Initial experimentation with this objective function indicated that it adequately serves the purpose of giving priority to minimizing the number of vehicles. In addition, no fine tuning or adjustment of different weights is needed in this objective function.

8.3.2 The Initial Population

To create a solution for our GA, requests are first placed in a relocation pool in a random order, before being inserted into the solution. Based on our investigation of several solution construction methods, as explained in Chapter 7, we chose the sequential construction algorithm ***(SEQ)***, to create each solution in the initial GA population. The same algorithm was also used to create or modify solutions during the evolutionary process, as will be explained later when we address the genetic operators. The underlying Hill Climbing (HC) routing heuristic and the overall sequential construction algorithm are explained in detail in Sections 7.4 and 7.5.1, respectively, and outlined in Algorithms 7.1 and 7.2.

The cost function, in Step 6 of the HC algorithm (Algorithm 7.1), is used to evaluate the quality of each route, and is different from the objective function (8.1) used to eva-

luate the overall solution (chromosome). This cost function tries to minimize the total route duration as well as the degree of infeasibility in capacity and TW constraints. The cost function of a route r is described by the following equation (same as Equation 7.1, repeated for convenience):

$$F(r) = w_1 \times D(r) + w_2 \times TWV(r) + w_3 \times CV(r) \ , \tag{8.2}$$

where $D(r)$ is the total route duration, $TWV(r)$ is the total number of time window violations in the route, and $CV(r)$ is the total number of capacity violations, while w_1, w_2 and w_3 are assigned weights in the range $[0, 1]$, and $w_1 + w_2 + w_3 = 1.0$. For this route cost function, we used the same weights used in Equation 7.1, i.e., $w_1 = 0.201$, $w_2 = 0.7$ and $w_3 = 0.099$.

Our routing algorithm allows routes to dynamically change during the search, i.e., previous routing decisions (locations) for some requests already existing in the route may be altered as new requests are added to the route. The new routing information is copied back to the chromosome whenever a change in the route occurs.

8.3.3 The Genetic Operators

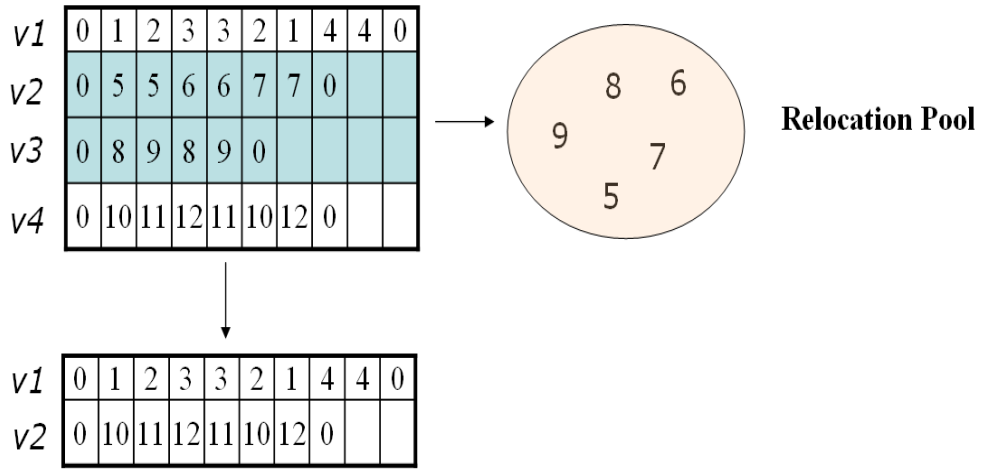
As previously mentioned in the introduction to this thesis, we try to focus on neighbourhood moves (represented by genetic operators here) as a tool for satisfying the hard problem constraints, and maintaining the feasibility of solutions throughout the search. Two crossover operators and one mutation operator have been designed to achieve this purpose. In what follows we explain in detail our proposed genetic operators.

Mutation

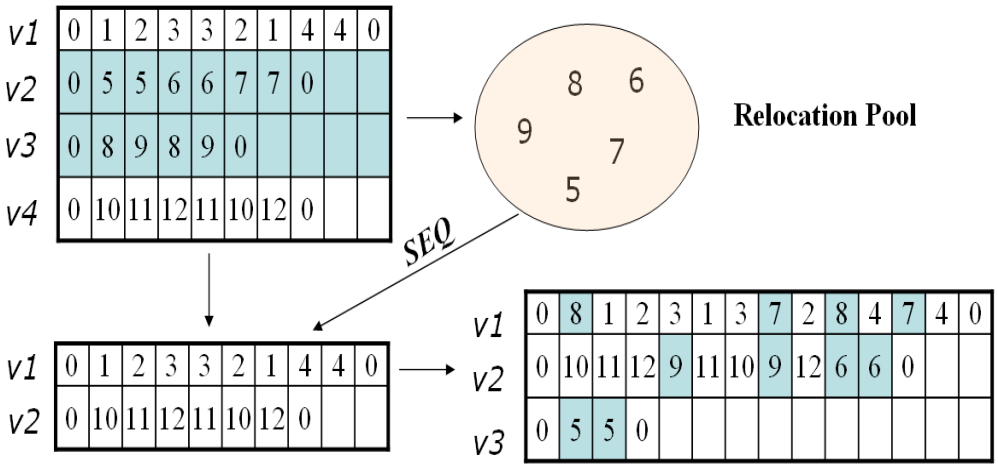
The mutation operator, which we will call the ***Vehicle Merge Mutation (VMM)***, simply tries to merge requests from two randomly selected vehicles. The idea is to try to reduce the number of vehicles by distributing the requests among already existing vehicles, or possibly combining two vehicles into one.

Figure 8.4 demonstrates the steps of the VMM. Figure 8.4(a) shows that vehicles $v2$ and $v3$ were randomly selected for merging from the current solution. The requests belonging to them will then be placed in a relocation pool in a random order. The remaining requests in the solution, i.e., vehicles $v1$ and $v4$ will be copied to the new solution to form a partial solution. Figure 8.4(b) shows that the requests in the relocation pool are then re-inserted into the partial solution using the SEQ construction, i.e., Algorithm 7.2, and the final

mutated solution is constructed. In our mutation operator, the new solution replaces the old one only if it is better in quality, i.e., if it has a lower objective function value.



(a) VMM - two vehicles selected for merging, and their request placed in the relocation pool. Remaining routes copied to the new solution without change



(b) VMM - requests in the relocation pool re-inserted into the partial solution using the SEQ algorithm

Figure 8.4: Vehicle Merge Mutation (VMM).

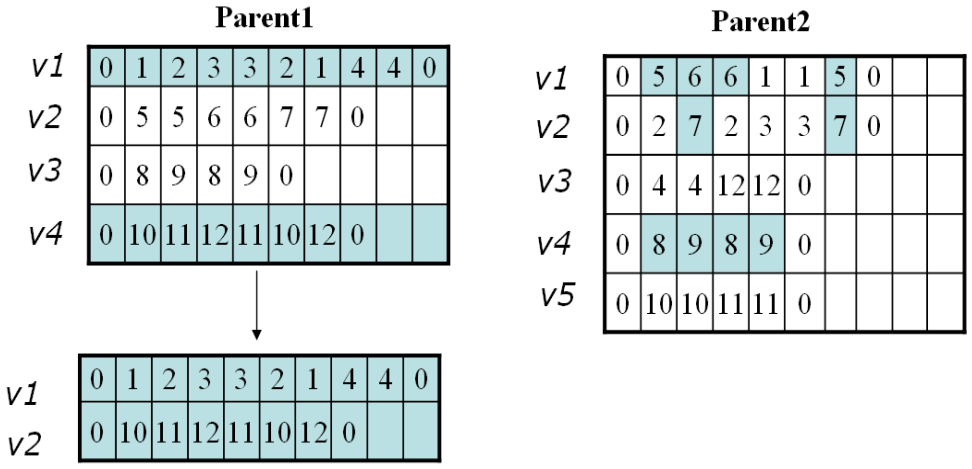
Crossover

Two crossover operators have been designed in our research. The first crossover operator, which we will call the ***Vehicle Merge Crossover (VMX)***, is similar to the mutation operator described above. However, instead of merging two vehicles from the same solution, the VMX tries to merge two vehicles selected at random, one from each parent solution.

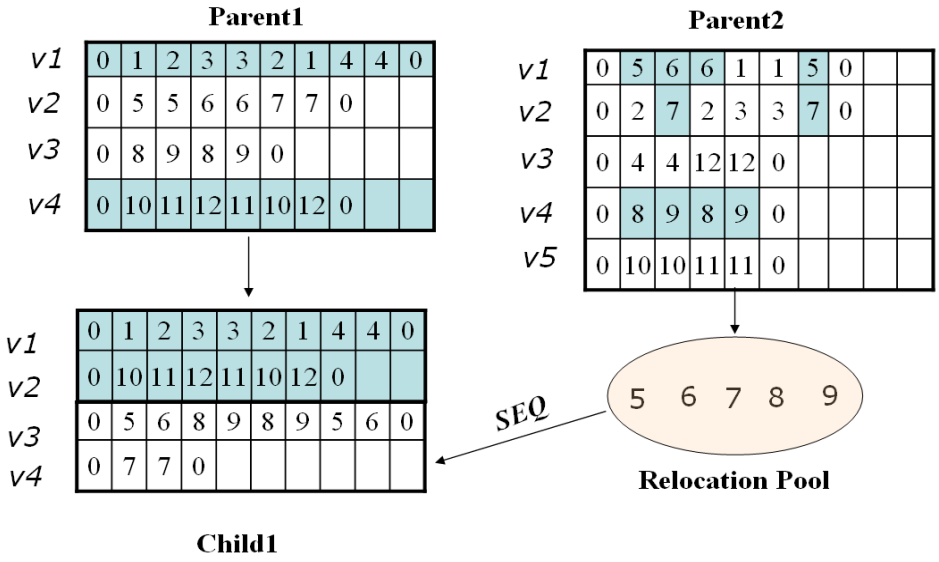
The second crossover operator, which will call the ***Vehicle Copy Crossover (VCX)***, tries to copy complete routes from the parent to the child. The number of routes to be copied is a random number between $1/4$ to $1/2$ the number of routes in the first parent. To select routes for inheritance, the VCX tries to select the 'good' routes. It is generally desirable to copy routes that serve a large number of requests, since our main objective is to reduce the number of vehicles. Accordingly, the VCX first ranks routes based to the number of nodes served in each route. The larger the number of nodes served the higher the rank of the route. Routes with the same number of nodes are ranked according to the total distance traveled, in which case routes with a shorter distance are more favourable than the longer ones.

Figure 8.5 demonstrate the steps of the VCX. Figure 8.5(a) shows that vehicles $v1$ and $v4$ were selected from Parent1, depending on the ranking criterion described above. These two vehicles are then copied to the first child. Figure 8.5(b) shows that the remaining requests that have not been included in Child1 (those highlighted in Parent2), will be copied in the same order of their appearance in Parent2 and placed in a relocation pool. The requests in the relocation pool are then sent to the SEQ construction algorithm and used to form a set of new routes. These new routes will afterwards be appended to the routes already existing in Child1, which were inherited from Parent1.

Child2 is created similarly by reversing the roles of parents. Our early experimentation indicated that the presence of both crossover operators was necessary for improving the results and satisfying the objective function.



(a) VCX - selected routes copied from parent1 to the child



(b) VCX - remaining requests copied from Parent2 to the relocation pool, and used to create new child routes

Figure 8.5: Vehicle Copy Crossover (VCX).

8.3.4 The Complete GA

Algorithm 8.1 shows the outline of the complete evolutionary algorithm that we used to solve the MV-PDPTW.

Algorithm 8.1: The Complete Genetic Algorithm.

1: Initialize a population POP of candidate solutions to the MV-PDPTW, using the SEQ construction algorithm (Algorithm 7.2)
2: **for** (a pre-specified number of generations) **do**
3: **for** (i=0; i<NumCrossovers;i++) **do**
4: Select parents P_1 and P_2 from POP, using roulette wheel selection
5: Randomly select crossover type (VCX or VMX)
6: Apply the selected crossover to parents (P_1, P_2) to produce child C_1
7: Apply the selected crossover to parents (P_2, P_1) to produce child C_2
8: With some probability, apply mutation (VMM) to C_1 and C_2
9: Update POP by integrating the new generation and eliminating some worst individuals {i.e., steady state GA with overlapping populations}

8.3.5 Attempts to Improve the Results

Before we report the experimental findings of this part of our research, we describe in this section some other operators and modifications to the solution approach that we experimented with, in an attempt to reach the best possible results. These attempts are briefly explained below, in addition to some concluding remarks about their performance. However, since the benefits realized from adopting these approaches did not meet our anticipations, they were abandoned in favour of the simpler approach described in Sections 8.3.1 to 8.3.4.

Vehicle Removal Mutation

This mutation operator is based on the GGA of [116]. One vehicle is removed and its requests are placed in the relocation pool in a random order, where the SEQ algorithm tries to re-insert them into the solution. This may result in reducing the number of vehicles, if all the requests from the pool were accommodated into other vehicles. The approach in [116] selected a random vehicle for removal, while in our research we selected the vehicle with the minimum number of nodes.

Large Neighbourhood Search (LNS) Mutation

This mutation operator is based on the Large Neighbourhood Search (LNS) move used in the work by [131], which is an adaptation of the same operator used by Shaw in [139] for the VRPTW. The idea is to remove a number of requests from the solution and then try to re-insert these requests back. Usually a large number of nodes (30% to 40%) are removed. However, since in our research this operator is used as a mutation operator, it would be more appropriate to reduce the number of requests removed. In our experimentation we found that 5% to 10% of the total number of nodes produce reasonable results.

Two variations of this operator were tried in our research, as advised by [131]. One version of this mutation selects the nodes to be removed randomly (random removal), while the other version selects the nodes that seem to be located in the wrong positions in the solution (worst removal). To find these requests, we calculate the cost of the solution (the objective function), with and without each request in turn, i.e., the request is temporarily removed from the solution to calculate its effect on the overall cost. The requests are then sorted in decreasing order of the change in cost. The requests to be removed are taken, in order, starting from the top of the list and then placed in a relocation pool in a random order. The removed requests are then re-inserted in the solution using the SEQ algorithm. The 'worst removal' version of the LNS algorithm gave slightly better results than the 'random removal' version.

A Memetic Algorithm

We also tried to improve the performance of the GA by introducing local search at various stages of the evolutionary process. This idea is based on the *memetic algorithm* described in [94]. We tried both the vehicle removal mutation and the LNS mutation (explained above), as neighborhood moves to create a new solution within the local search. We also used an SA acceptance for solution replacement.

More specifically, in this memetic algorithm, local search is performed on some individuals in the population (according to a certain probability), for example at the beginning of each generation. The local search repeatedly applies a selected 'mutation' operator on the individual for a pre-specified number of trials. Within the local search, a new solution replaces the current solution using an SA acceptance criterion, i.e., a better solution always replaces the current solution, while a worse solution may replace the current solution with some small probability. The best solution found during the local search is finally the one that replaces the initial solution (the individual that we started with).

Our experimentation indicated that the more suitable operator to be used within the local search seems to be the LNS operator. It should also be noted that a small population size (e.g. 30 individuals) would be most appropriate for this memetic algorithm, due to the large increase in processing time that result from the repeated application of the local search.

Allowing Infeasible Solutions

In our attempt to improve the results, we also tried to add some infeasible solutions to the set of allowable solutions. The idea is that sometimes good solutions lie in the vicinity of infeasible solutions, so by allowing infeasibility we may be able to climb out of local optima. In this version of our algorithm, whenever a solution construction is needed during the initialization phase of the population, or during the course of the evolutionary operators, there was a 50% chance that the construction process will allow infeasibility.

In order to restrict the amount of infeasibility and reduce the search space, we only allowed violations in the time window but not in the capacity constraint. To do that, several modifications to the algorithm were needed.

Firstly, we had to restrict the number of nodes allowed in each individual vehicle. Otherwise, we could end up with a solution in which all requests are served by one vehicle. This is due to the fact that even if violating the capacity is not permitted, the algorithm would still be able to find a solution in which all requests are served, without any capacity violations, using only one vehicle. For example, a solution in which each pickup is followed by its delivery will always satisfy the capacity constraint. Specifically, the allowed number of nodes, for each vehicle individually, was set to a random number ranging from 1 to m, where m was arbitrarily chosen to be 20% of the total number of nodes in the problem instance. Thus, nodes are added to the current vehicle until the predetermined limit on the number of nodes for this vehicle is reached, at which stage a new vehicle has to be allocated.

Secondly, the objective function has to be changed in order to penalize infeasibility in the solution. The following equation was used

$$O(S) = w_1 \times N(S)^2 + w_2 \times TotDist(S) + w_3 \times TWV(S). \tag{8.3}$$

Where $N(S)$ is the number of vehicles in the solution, $TotDist(S)$ is the total distance traveled by all vehicles in the solution, and $TWV(S)$ is the number of time window violations in the solution. w_1, w_2 and w_3 are weights in the range [0,1] and $w_1+w_2+w_3 =$

1.0. As with all other similar objective functions in our research, the penalty on the time window violations has to be large in order to get rid of infeasible solutions as the search progresses.

Thirdly, the genetic operators had to be modified. We modified the VCX crossover operator by ranking vehicles according to the percentage of infeasibility in each vehicle, rather than the number of nodes visited and the total distance, as done previously when only feasible solutions were allowed. The VMX crossover was used without modification.

We also used the LNS mutation, after modifying it to remove the requests that have a time window violation and try to re-insert them again in the solution, i.e., the worst requests are now the requests that violate the time window. The vehicle removal mutation was also used without modification.

Results of the Improvement Attempts

Based on our experimentation, we can rank the effectiveness of the above improvement attempts (from best to worst) as follows: the memetic algorithm, the LNS operator, the vehicle removal mutation and finally allowing infeasible solutions.

The improvement introduced by these variants, or any combination of them, though, did not warrant, in general, the increase in the complication of the overall algorithm. These variants seem to cause an improvement in some test cases, but may cause a degradation in the results of other test cases. Accordingly, to keep the overall algorithm as simple as possible, which is the main philosophy in our research, we decided not to include any of these variants in the final testing version of the proposed GA. Thus, only the operators described in Section 8.3.3 and the final GA described in Section 8.3.4, were adopted in the final version. The experimental results of this final version are detailed in the following subsection.

8.3.6 GA Experimental Results

To test our algorithm, we implemented a steady state GA with a 95% replacement. The following parameters were used: population size= 500, crossover probability= 0.6, mutation probability= 0.05, and the number of generations= 300. In cases where crossover is performed either VCX or VMX is selected at random. We used the standard 56 (100-customers) benchmark instances, created by Li & Lim in [100]. For the different types of problem instances in the data set, the reader is referred to Section 7.6.1. The algorithm was run 10 times on each problem instance.

We compared our algorithm, which we will call the **Grouping-Routing GA *(GRGA)***, with the GA in [33], denoted by ***CKKL***[5], and the grouping GA in [116], denoted by ***GGA***. To the best of our knowledge, they are the only GA approaches that have been attempted in the literature for the MV-PDPTW and applied to the published benchmark data of [100]. The results in [116] are also close to the best known results, so we found that a comparison with their results will be sufficient for the purpose of this part of our research[6]. Before we report our experimental findings, though, we present in Table 8.1 a comparison of the most distinctive features of the three algorithms under consideration.

Figures 8.6 and 8.7 show the best results achieved by the three algorithms in terms of the number of vehicles and the total distance traveled. The two figures show that our GA clearly achieves better results than the CKKL algorithm in almost all test cases. There are only 5 cases in which our algorithm produced one more vehicle than the number of vehicles produced by the CKKL. Moreover, all our total distance results were better than the results of the CKKL. On average, the improvement of our results compared to the results of the CKKL in the number of vehicles is approximately 16%, while the average improvement in the total travel distance is approximately 36%. Also, our results are also close to the results of the GGA in the number of vehicles produced, with only few exceptions. Nevertheless, the resulting total distance is larger than the resulting total distance of the GGA in most test cases. This is even more noticeable in instances with a long time window width, i.e., instances of category '2'.

[5]We thank the authors of [33] for providing us with the data files containing their detailed results.

[6]The detailed results of the GRGA approach and a comparison with best known results will be presented in Section 8.4

Table 8.1: Comparison between the GRGA, GGA and CKKL Algorithms

GA	GRGA	GGA	CKKL
General Approach	- A GA handles both the grouping and the routing aspects of the problem. - All problem information is explicity monitored and manipulated by the GA.	- A GA only handles the grouping aspect of the problem. - The routing information is hidden from the GA and created when the chromosome is decoded.	- A GA handles both the grouping and the routing aspects of the problem. - All problem information is explicity monitored and manipulated by the GA.
Encoding	- A Chromosome has a variable number of genes. - Each gene is a vehicle route (a sequence of visited nodes). - Same code for pickup and delivery (P&D), and a parser to traverse the route and identify each.	- A Chromosome has a variable number of genes. - Each gene is a cluster of requests assigned to one vehicle. - A sperate data structure and an insertion heuristic are used to create individual routes.	- A Chromosome has a variable number of genes. - Each gene is a vehicle route (a sequence of visited nodes).
Routing or Insertion Heuristic	- Insert P&D pair at end of route, and improve the route using an HC algorithm. - The routing decisions of requests in the route may be changed by the genetic operators. - New routes are copied back to the chromosomes, during recombination and mutation.	- Examine all feasible insertions for the P&D pair in all routes, and select the insertion that causes minimal additional cost. - The routing decisions of nodes already existing in the route are static and are not changed by the genetic operators.	- The P&D pair is inserted in a feasible route position. - Position of insertion could be modified later using a local search mutation.

Continued on Next Page...

Table 8.1 . . . Continued from Previous Page

GA	GRGA	GGA	CKKL
Crossover	- Vehicle Copy Crossover (VCX) & Vehicle Merge Crossover (VMX). - Crossover operators are aware of, and make use of, the routing information of each gene. - Offspring is always feasible, and no repair method needed.	- Adaptation of the general GGA crossover, where crossover is not aware of the routing information of each gene. - Consecutive set of clusters are selected from the first parent and inserted in the second parent. - Chromosome cleanup is needed to correct infeasibility of offspring.	- Sequence Based Crossover (SBX): fragments of two routes are selected from each parent and joined together to form a new route. - Route Based Crossover (RBX): two selected routes are exchanged between the two parents. - If possible, infeasibility of offspring is repaired. Otherwise, offspring is discarded.
Mutation	- Vehicle Merge Mutation (VMM). - Mutation is performed on the offspring created by crossover with a certain probability.	- Remove one vehicle and reassign its requests. - Mutation is performed on the offspring created by crossover with a certain probability.	- One-Level exchange Mutation (1M): removes one vehicle and reassigns its requests. - Local Search Mutation (LSM): tries to find better locations for requests in a randomly selected route. - Mutation is performed on a randomly chosen individual.
Objective Function	- Minimize the number of vehicles, followed by total distance and total duration.	- Minimize total travel distance, irrespective of the number of vehicles.	- Minimize a weighted sum of the number of vehicles, total distance and total duration (equal weights are assigned).

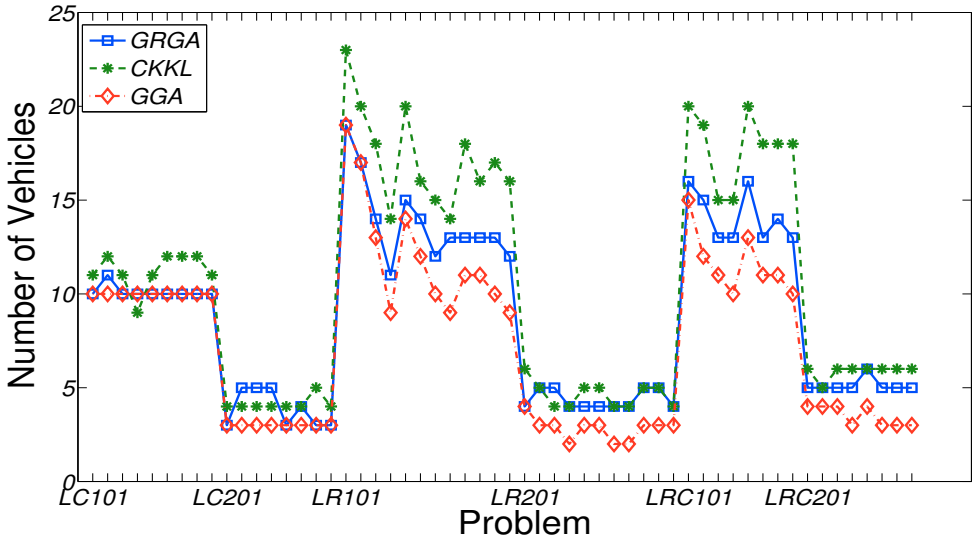


Figure 8.6: Total number of vehicles produced by all algorithms.

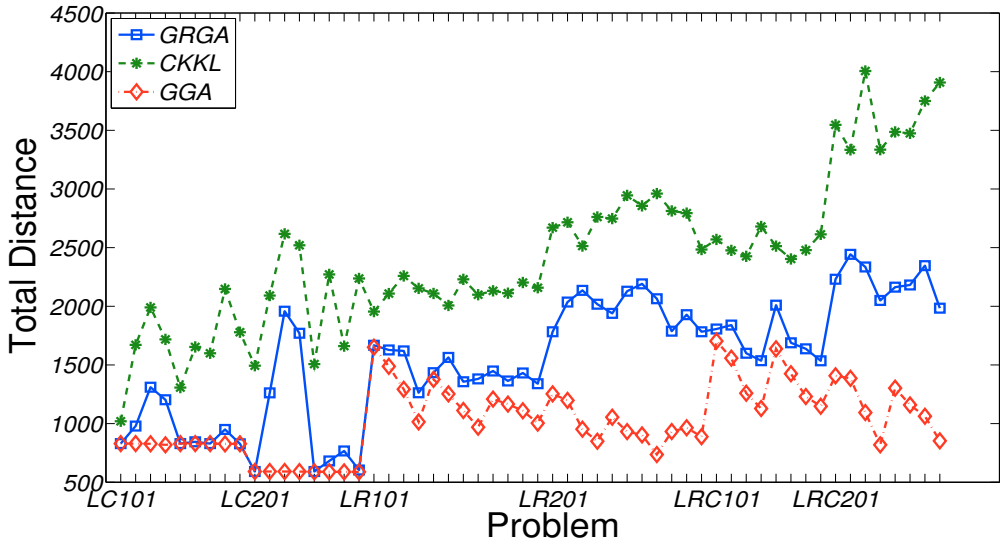


Figure 8.7: Total travel distance produced by all algorithms.

When we try to analyze the computational results in the light of the differences between the three algorithms summarized in Table 8.1, we will realize the following. First, since both the GRGA and the CKKL algorithms have the same chromosome structure and the same components of the objective function, then the obvious success of the GRGA compared to the CKKL algorithm must be due to the routing algorithm and/or the genetic operators used in the former. The success of the GRGA to produce solutions with less vehicles in most test cases could be attributed to the presence of two genetic operators the are specifically designed for this purpose and heavily applied during the search, namely the VMM mutation and the VMX crossover. On the other hand, the 1M mutation, used for reducing the number of vehicles in the CKKL, is only performed occasionally during the search. Also, the noticeable success of the GRGA in terms of reducing the total travel distance, could be attributed to the routing algorithm that is called whenever a route is created or modified to try to improve the quality of the route by reducing its overall cost. This again is in contrast to the LSM mutation of the CKKL that tries to improve the route, but is only called occasionally during the search. The VCX crossover, used in the GRGA, also seems to do a better job than the RBX crossover used in the CKKL. The RBX merely exchanges one route between parents, while the VCX tries to be selective when transferring routes from the parent to the child, by choosing routes that serve a large number of nodes with the smallest possible distance. It also seems that the SBX crossover, used in the CKKL, may not be suitable for the genetic representation used. Since the gene is actually a complete route, it would seem more appropriate to transfer a collection of routes rather than route fragments between parents.

We will now try to analyze the reasons behind the sub-optimal results achieved by the GRGA compared to the GGA, specially in terms of the total travel distance[7]. First of all, one of the major differences between the two algorithms is the objective function. The objective function of the GRGA includes the number of vehicles, the total distance, and the total duration, while the objective function of the GGA only includes the total distance. Thus, the best solution as far as the GRGA is concerned must balance all three components, i.e., it takes into consideration other parameters involved in the routing schedule like the total waiting time of the vehicle at each location and the total service time. The second main difference is the routing heuristic used in both algorithms. The routing (insertion) algorithm of the GGA tries to find the best insertion position for each newly inserted request in all available routes, while we use a simple and fast algorithm that accepts any feasible insertion. The attempt to improve the route, using the HC algorithm, is only local to each route and does not involve comparing the insertion cost with other

[7]It should also be noted that our algorithm was run only 10 times on each test case, while the GGA of [116] was run 30 times and the best result was selected.

routes. It seems that our routing algorithm could probably use the help of an additional local search operator (e.g. 2-Opt), to try to improve the routes, which should in turn help reduce the number of vehicles.

In our opinion, these two issues are the main reasons for the inability of our approach to reach the solution quality of the GGA. On the other hand, the genetic operators in both seem to be comparable, since they either try to reduce the number of vehicles, or copy complete routes from parents to children. However, we think that our VCX crossover may be more suitable than the crossover of the GGA for this problem type, because transferring a sequence of routes is not really meaningful, since the order of routes in the chromosome is irrelevant.

Also, as mentioned previously, our algorithm seems not able to cope with instances of type '2', as evident by the large gap between our results and the results of the GGA in the total travel distance. This could be explained if we recall that our routing algorithm had a neighbourhood move, which was guided by the TW. It seems that the routing algorithm was thus capable of dealing better with instances in which the TW constraint is hard to satisfy, i.e., those with a tight TW width. This neighbourhood move may not be sufficient to improve the route in problems with a large TW width, because of the availability of many different feasible orders of nodes. An alternative neighbourhood move may be needed in that case. For example, the neighbourhood move could take into account not only the bound(s) of the time windows of the swapped locations, but also whether the swapping can improve other route characteristics, such as the total waiting time, the total travel time, or the extra travel distance. The results obtained for instances of type '2' also sustain our previous observation in Section 7.6.3, regarding the large solution space for this specific type of problem instances.

Finally, the average processing time needed by our algorithm, over all 56 problem instances, was 176.9 seconds, which is comparable to the processing time of the grouping GA in [116] having an average of 167.1 seconds. Nevertheless, the separation of the data structure used for individual routes in [116] from the actual chromosome, seems to be favourable than our representation which includes all vehicle routes in the chromosome, as far as processing time is concerned, since the transfer of complete routes during the recombination and mutation operators is definitely time consuming. The authors in [33], on the other hand, do not report their processing time.

8.4 Simulated Annealing (SA) for the MV-PDPTW

As previously mentioned in Section 8.1, we will try in this part of our research to adopt some of the neighbourhood operators used in the GA approach for the MV-PDPTW within an SA approach. As expected, the solution representation used in the SA is similar to the chromosome representation used in the GA (explained in Section 8.3.1). The same objective function (Equation 8.1) is also used here.

The initial solution from which the SA will progress is created by generating a number of random solutions[8], in a manner similar to the creation of an initial genetic population (as explained in Section 8.3.2), and the best solution among them, in terms of the objective function value, is selected.

To allow for an adaptive calculation of the SA parameters for each problem instance individually, we again used the approach proposed by [40], as previously done for the SV-PDPTW (see Section 6.3 and Algorithm 6.1). Thus, the annealing parameters are calculated based on the average value of $\Delta cost$, where $\Delta cost$ is the difference in the objective function value between some randomly generated solutions for the current problem instance. In our approach, the same set of random solutions created for the selection of the starting solution, were also used for the purpose of calculating the annealing parameters.

A crucial part of any SA algorithm is the neighbourhood move that will be used to generate a new solution. In this part of the research, we experimented with different neighbourhood moves from the ones tried in our GA approach. Two neighbourhood moves were found to be the most appropriate for the SA approach:

1. **The Large Neighbourhood Search (LNS) Move:** the LNS move is similar to the LNS mutation operator explained in Section 8.3.5, which is inspired from the Adaptive Large Neighbourhood Search (ALNS) approach of [131]. As previously explained, the idea is to remove and then re-insert a large number of requests in each application of the move. The authors in [131] recommend that 30% to 40% of the total number of nodes is removed in each iteration. In our LNS move, we slightly increased the allowed range for the number of removed requests, in order to explore a wider area of the search space. The number of removed requests in our LNS move ranges between 20% to 50% of the total number of requests. In addition, we adopted here the "Worst Removal" variant from the three types of removal heuristics applied in [131] (see Section 8.2 for more details about the different removal heuristics of [131]). In our approach, the requests removed are those that are estimated to cause

[8] 100 random solutions were created in this experiment.

a large increase in the cost of their respective routes, i.e., they may be inserted in unfavourable positions in the solution. Thus, to determine the cost of each request, the total travel distance of the route to which the request belongs is calculated, with and without the request under consideration, and the difference in the route distance is used as a measure of the request cost. Requests having the highest costs are selected for removal from the current solution. After this, the removed requests are placed in a relocation pool in a random order, before they are inserted back in the solution using the SEQ construction algorithm, hoping to find better insertion positions for them in the new solution.

2. **The Vehicle Merge (VM) Move:** this move is identical to the Vehicle Merge Mutation (VMM), explained in Section 8.3.3. The vehicle merge move selects two vehicles at random from the current solution. Then, the requests belonging to them will be temporarily placed in a relocation pool in a random order, before they are re-inserted in the solution using the SEQ algorithm as shown in Figure 8.4.

Similar to the idea of the 3-stage SA for the SV-PDPTW (explained in Section 6.3), our SA approach to the MV-PDPTW operates in two stages. In the first stage, the LNS move is used to generate a new solution, while in the second stage the VM move is used instead. Changing the SA move in this manner allows the search process to slightly perturb the current solution, before trying to re-optimize it, which may help in escaping local optima. The second SA stage starts from the final solution obtained in the previous stage and from the final temperature reached by the end of the previous stage. During each stage, the best so far solution is saved and each SA stage terminates when no improvement is realized in the best solution for a consecutive number of iterations[9]. Also, during each stage, the current temperature value is reduced in each iteration of the SA algorithm.

For a more extensive searching, the two SA stages are repeated several times. Again, the repetition only stops when the best obtained solution reaches a stage of stagnation and does not improve for a number of consecutive attempts of applying the two stages[10]. We also found during our computational experimentation that there seems to be no significance to the order of application of the two moves. Thus, as long as the two stages are repeated, the SA may be started from either move. The overall 2-stage SA approach is shown in Algorithm 8.2.

[9] 100 consecutive attempts without improvement was used to terminate each SA stage.

[10] 10 consecutive attempts without improvement was used to stop the repetition of the 2-stage SA approach

Algorithm 8.2: The 2-Stage SA Algorithm.

1: Find an initial solution ($InitSol$) and calculate the annealing parameters
2: $BestSol \leftarrow InitSol$ {Initialize the best so far solution}
3: Initialize $MaxAttempts$ to a small number {We used 10}
4: $NoImprovement \leftarrow 0$
5: **repeat**
6: $OldCost \leftarrow Objective(BestSol)$ {Calculate the cost of the solution using Equation 8.1}
7: $BestSol \leftarrow SA_{LNS}(BestSol)$ {Perform SA on the current best solution, using the LNS move, and return the best found solution}
8: $BestSol \leftarrow SA_{VM}(BestSol)$ {Perform SA on the current best solution, using the VM move, and return the best found solution}
9: $NewCost \leftarrow Objective(BestSol)$ {Calculate the cost of the new solution using Equation 8.1}
10: **if** ($NewCost$ is not better than $OldCost$) **then**
11: $NoImprovement++$
12: **else**
13: $NoImprovement \leftarrow 0$
14: **until** ($NoImprovement$ reaches $MaxAttempts$)

SA Experimental Results

Our SA algorithm was tested on the same 100-customers test cases used for testing our GA, as explained in Section 8.3.6. Similar to the GA, the SA algorithm was run 10 times on each test case. Table 8.2 shows the best result, in the 10 runs, achieved by both the GA and the SA for each test case. The best result is the one having the minimum number of vehicles, and for the same number of vehicles, the one having the minimum total travel distance. The better obtained result between the two algorithms is highlighted in boldface. In addition, the processing time in seconds is also shown in the table. The last two columns of the table show the current best known results for these problem instances, which are published in:

`http://www.sintef.no/Projectweb/TOP/Problems/PDPTW/Li--Lim-benchmark/`

The best known results are attributed to [100], [12], and [135].

The last two rows of the table show, respectively, the overall average results of the corresponding column, and the percent difference (gap) between our average and the average of the best known results, in terms of both the number of vehicles and the total distance.

Table 8.2: GA & SA Best Results (100-Customers)

	GA Results			SA Results			Best Known	
Name	**Vehic**	**Dist**	**Time**	**Vehic**	**Dist**	**Time**	**Vehic**	**Dist**
LC101	**10**	**828.94**	34.8	**10**	**828.94**	45.72	10	828.94
LC102	11	978.48	109.97	**11**	**945.88**	57.99	10	828.94
LC103	10	1310.3	131.3	**10**	**1238.57**	226.67	9	1035.35
LC104	10	1203.1	145.28	**9**	**1328.29**	201.8	9	860.01
LC105	**10**	**828.94**	85.03	**10**	**828.94**	40.41	10	828.94
LC106	10	844.58	87.69	**10**	**828.94**	41.52	10	828.94
LC107	**10**	**828.94**	85.66	**10**	**828.94**	46.7	10	828.94
LC108	10	949.96	85.52	**10**	**828.04**	64.66	10	826.44
LC109	**10**	**827.82**	83.7	10	849.08	52.63	9	1000.6
LC201	**3**	**591.56**	261.17	**3**	**591.56**	3.47	3	591.56
LC202	5	1261.81	304.77	**4**	**1186.64**	58.34	3	591.56
LC203	5	1957.86	420.83	**5**	**1903.04**	75.36	3	585.56
LC204	5	1770.26	627.95	**4**	**2194.2**	119.03	3	590.6
LC205	**3**	**591.56**	283.27	**3**	**591.56**	8.02	3	588.88
LC206	4	681.35	216.52	**4**	**626.89**	11.61	3	588.49
LC207	3	766.62	293.88	**3**	**701.72**	24.77	3	588.29
LC208	**3**	**604.51**	320.33	3	604.7	15.19	3	588.32
LR101	**19**	**1667.68**	80.67	**19**	**1667.68**	10.22	19	1650.8
LR102	**17**	**1627.73**	90.61	17	1627.91	48.41	17	1487.57
LR103	14	1619.68	103.23	**14**	**1525.99**	86.06	13	1292.68
LR104	**11**	**1262.3**	231.22	11	1335.91	146.14	9	1013.39
LR105	**15**	**1433.79**	76.48	15	1450.98	49.74	14	1377.11
LR106	14	1564.06	95.33	**13**	**1458.71**	114.17	12	1252.62
LR107	12	1356.72	99.78	**12**	**1353.44**	195.7	10	1111.31
LR108	13	1380.93	108.16	**12**	**1353.05**	235.58	9	968.97
LR109	**13**	**1448.14**	85.14	13	1449.38	71.34	11	1208.96
LR110	13	1362.74	94.81	**12**	**1323.12**	151.78	10	1159.35
LR111	13	1431.07	103.59	**12**	**1299.28**	278.8	10	1108.9
LR112	12	1339.66	95.56	**11**	**1237.92**	100.78	9	1003.77
LR201	**4**	**1783.1**	194	4	1841.72	65.25	4	1253.23
LR202	5	2035.85	335.24	**4**	**2083.77**	177.84	3	1197.67
LR203	5	2135.47	500.69	**4**	**2312.46**	409.73	3	949.4

Continued on Next Page...

Table 8.2 . . . Continued from Previous Page

	GA Results			SA Results			Best Known	
Name	**Vehic**	**Dist**	**Time**	**Vehic**	**Dist**	**Time**	**Vehic**	**Dist**
LR204	**4**	**2017.64**	541.06	4	2059.11	181.25	2	849.05
LR205	**4**	**1939.61**	289.36	4	2123.62	78.56	3	1054.02
LR206	**4**	**2128.15**	554.16	4	2134.15	206.45	3	931.63
LR207	**4**	**2191.28**	615.16	4	2360.2	185.77	2	903.06
LR208	4	2064.23	723.53	**4**	**2059.02**	276.97	2	734.85
LR209	5	1787.82	275.5	**4**	**2058.16**	106.94	3	930.59
LR210	5	1928.42	313.3	**4**	**2175.39**	164.81	3	964.22
LR211	**4**	**1782.99**	401.3	4	1886.82	196.34	2	911.52
LRC101	**16**	**1806.27**	81.13	**16**	**1806.27**	26.11	14	1708.8
LRC102	15	1840.05	94.08	**14**	**1776.15**	80.47	12	1558.07
LRC103	13	1599.7	118.67	**13**	**1558.38**	122.19	11	1258.74
LRC104	13	1535.82	110.97	**12**	**1464.11**	279.77	10	1128.4
LRC105	16	2009.51	91.36	**16**	**1862.36**	88.17	13	1637.62
LRC106	**13**	**1688.86**	87.11	14	1672.87	53.06	11	1424.73
LRC107	14	1637.27	93.05	**12**	**1452.42**	153.31	11	1230.15
LRC108	13	1535.48	101.88	**12**	**1379.59**	148.47	10	1147.43
LRC201	**5**	**2230.74**	144.81	5	2327.42	89.98	4	1406.94
LRC202	**5**	**2442.78**	286.58	5	2525.27	171.39	3	1374.27
LRC203	**5**	**2335.04**	395.59	5	2413.63	400.44	3	1089.07
LRC204	5	2049.8	628.95	**4**	**2323.05**	289.55	3	818.66
LRC205	6	2162.85	162.3	**5**	**2605.27**	148.02	4	1302.2
LRC206	**5**	**2181.3**	191.58	5	2400.12	40.58	3	1159.03
LRC207	5	2346.32	245.63	**5**	**2319.27**	172.53	3	1062.05
LRC208	**5**	**1983.58**	247.73	5	2195.22	68.34	3	852.76
AVG	**8.75**	**1562.52**	**226.2**	**8.43**	**1592.23**	**124.37**	**7.18**	**1036.68**
GAP%	**22%**	**51%**	-	**17%**	**54%**	-	-	-

Table 8.2 shows that SA achieved better results than the GA in 31 out of the 56 problem instances. In terms of the overall average results, SA achieved a slightly better average in the number of vehicles and a slightly worse average in the total distance traveled, which can also be seen from the relative difference to best known results, indicated in the last row of the table. In terms of the average processing time, though, the result was in favour

of SA, with approximately 45% less average processing time than the GA.

Figure 8.8 shows the average objective function value (as calculated by Equation 8.1) of the 10 runs, for each problem instance, obtained by both the GA and the SA.

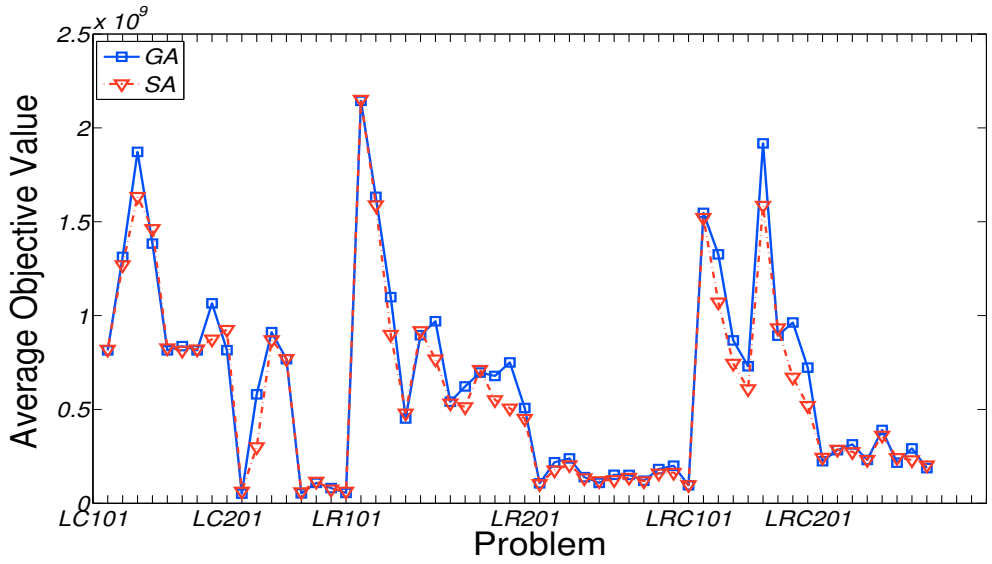


Figure 8.8: Average objective value for the GA and the SA algorithms.

In general the results in Table 8.2 and Figure 8.8 indicate that both the SA and GA obtained comparable results when tested on the 100-customers instances for the MV-PDPTW. This could be explained by realizing that both algorithms adopted similar neighbourhood moves. The VM move used in SA is exactly the same as the VMM mutation used in the GA. In addition, the LNS move used in SA is also comparable to the crossover operators used in the GA. In the GA, the crossover operators generally try to remove requests belonging to some selected routes from parent solutions and insert them back into the child solution. This process is similar to what the LNS move performs in SA, when requests are removed and then re-inserted into the solution. Moreover, selecting 'good' routes to be transferred to the child in the VCX crossover is also analogous to removing 'bad' requests in the LNS move adopted in SA.

The results of both algorithms, though, were inferior to the best known results, especially in terms of the total travel distance. This indicates that both algorithm could probably make use of an additional local search method, such as 2-Opt edge exchange, to improve the quality of the routes and reduce the travel distance at some stage during the run.

8.5 Summary and Future Work

In this chapter we investigated both a GA and an SA for solving the MV-PDPTW. Our GA approach tried to face the challenge of allowing the GA and its operators to be aware of and manipulate both the grouping and the routing aspects of the problem. A challenge that most previous GA research on this problem has achieved little success with, or has been avoided completely by allowing the GA to tackle only one problem aspect. We first tried a simple representation and an intelligent neighbourhood move to handle the routing part of the problem. For the grouping part, on the other hand, we designed new genetic operators that try to exploit problem-specific information and create new solutions from existing ones, while maintaining the feasibility of solutions throughout the search. Our operators overcome the difficult problem constraints, and avoid at the same time the need for a repair method to fix infeasible solutions, a technique that previous GAs and most other heuristic and meta-heuristic techniques have been relying on to maintain feasibility. Overall, though, our algorithm is a simple and straightforward GA technique, and the genetic operators developed here are applicable to other related routing problems.

We compared our results with two previous GA attempts to solve the problem, the CKKL algorithm of [33] and the GGA algorithm of [116]. The experimental results show that our algorithm was able to greatly improve upon the results of the CKKL algorithm, using just a few simple modifications in the routing algorithm and the genetic operators, and applying problem-specific information. The improvement was evident in both main objectives, the number of vehicles and the total travel distance. However, our results are still behind the results of the GGA in most test cases. The three algorithms have been thoroughly analyzed and possible reasons behind the differences in performance have been identified.

The GA was also compared with a 2-stage SA approach that uses two different neighbourhood moves repeatedly. The neighbourhood moves used within this SA approach are analogous to the genetic operators used previously in the GA, since they rely on removing and then re-inserting requests or merging selected vehicles.

Both the GA and the SA obtained comparable results when tested on published benchmark instances, which is mostly due to their reliance on similar neighbourhood operators. However, neither algorithm was capable of competing favourably with best known results. In general, it appears that our representation and neighbourhood operators in both algorithms are doing a fair job in guiding the search towards better solutions. However, to cope with the difficulty of the problem and the different types of problem instances, both approaches still need further improvement. For this purpose, different neighbourhood moves could be

attempted in the route improvement heuristic, for example by taking some route characteristics into consideration when swapping locations, such as the resulting total waiting time or the total route duration. Moreover, a local search method, such as 2-Opt or 3-Opt, could be added to improve the quality of the routes and reduce the total travel distance. In terms of processing time, though, the GA was clearly much slower than the SA, which is expected, due to the overhead of maintaining a large population of solutions.

Having concluded our investigation of the PDPTW both its single and multiple vehicles variants, in the following chapter we move on to another interesting and important variant of pickup and delivery problems, the **One-Commodity Pickup and Delivery Problem (1-PDP)**.

Chapter 9

The One-Commodity Pickup and Delivery Problem: Introduction and an Evolutionary Perturbation Scheme

The One-Commodity Pickup and Delivery Problem (1-PDP) is another important problem in transportation and logistics systems. The 1-PDP deals with supplying and collecting one type of commodity from a number of customers, some of them are designated as pickup customers and the others as delivery customers. Each pickup customer provides a certain amount of the commodity, while each delivery customer consumes a certain amount of the same commodity, i.e., goods collected from pickup customers can be delivered to any delivery customer. All customers are served by one vehicle with a limited capacity, and the journey of the vehicle should start and end at a central depot. The depot can supply or consume any additional amount of the commodity that is not supplied or consumed by the customers. Our goal is to find a feasible and minimum cost route for the vehicle, such that all customers are served without violating the vehicle capacity constraint.

This problem has attracted our interest for several reasons. First, there are many applications for this problem in practice. For example, the commodity could be milk that should be collected from farms and delivered to factories with no restriction on the origin and the destination of the product, or it could be money that should be distributed between the branches of a bank [74]. It can also model any logistic situation in which some warehouses have an extra supply of some commodity, while others are in short of the same commodity. A typical situation is when some hospitals need to transfer a certain medicine to other hospitals, who are in short of this medicine. For example, an H1N1 vaccination or treatment could be transferred in urgent epidemic circumstances [106].

Second, this problem has not been adequately explored in the literature. Since the introduction of the 1-PDP in [74], only very few papers seem to have handled it. Finally, there are other important problems in the literature that are closely related to the (1-PDP).

For example, the Traveling Salesman Problem with Pickup and Delivery (TSPPD) (see Section 9.3 for more details about this and other related problems).

Our initial attempt to handle the 1-PDP was based on a simple **Evolutionary Perturbation Scheme (EPS)** that has proved successful for other routing problems, such as the TSP and the Capacitated Vehicle Routing Problem (CVRP). We made some preliminary experimentation with this technique for the 1-PDP and tested it on published benchmark data. The details of the approach and its experimental results are reported in this chapter. The outcome of the approach, though, did not meet the anticipated standard. Some analysis of possible shortcomings of the algorithm and future directions are also discussed here. Nevertheless, A more successful heuristic for solving the 1-PDP, will be detailed in the next chapter.

The rest of this chapter is organized as follows. Section 9.1 formally defines the 1-PDP, while Section 9.2 is a brief summary of related research in this area. Some problems that are related to the 1-PDP are presented in Section 9.3. The EPS technique attempted for solving this problem is explained in Section 9.4, together with its experimental findings. Finally, Section 9.5 concludes with a summary and some thoughts for future work.

9.1 The 1-PDP

Based on the definition provided in [73], the 1-PDP is characterized by having a set of customers $i(i = 1, 2, ..., n)$, where customer 1 is the depot. $V = 1, 2...n$ is the vertex set, and E is the edge set. For each pair of locations (i, j) the travel cost c_{ij} is known in advance. For each customer there is an associated demand q_i, such that $q_i > 0$ for a pickup customer and $q_i < 0$ for a delivery customer. The depot is considered as a customer that supplies or consumes any amount of the product that is not supplied or consumed by the set of customers, and the demand of the depot will be calculated as $q_1 = -\sum_{i=2}^{n} q_i$. All customers are to be served by one vehicle with a limited capacity $Q > 0$, and the capacity of the vehicle should at least be equal to the maximum customer demand (whether a pickup or a delivery), i.e., $Q \geq \max_{i \in V}\{|q_i|\}$.

A feasible path for the vehicle is a path that travels from the first customer to the last customer, and visits each customer exactly once without exceeding its capacity. The authors in [73] more formally describe the feasibility of a path P through the sequence of customers $i_1, ..., i_k$, with $k \leq n$. If we assume that $l_j(P)$ is the load of the vehicle after visiting the j^{th} customer, and $l_0(P) = 0$, then P is feasible if and only if

$$\max_{j=0,...k}\{l_j(P)\} - \min_{j=0,...k}\{l_j(P)\} - Q \leq 0. \tag{9.1}$$

In [158], more explanation of Equation 9.1 is given by indicating that $\min_{j=0,\dots k}\{l_j(P)\}$ could be either 0 or negative, given the existence of some negative customers' demands. Accordingly, there are two possible conditions for a feasible route:
(i) if $\min_{j=0,\dots k}\{l_j(P)\} = 0$, Equation 9.1 will simply become $\max_{j=0,\dots k}\{l_j(P)\} \leq Q$.
(ii) if $\min_{j=0,\dots k}\{l_j(P)\} < 0$ (i.e., there is a shortage in the required delivery demands), we can assume that $|\min_{j=0,\dots k}\{l_j(P)\}|$ will be supplied by the depot when the vehicle starts its journey. Thus, $|\min_{j=0,\dots k}\{l_j(P)\}|$ will be added to the demands of all nodes, making $\min_{j=0,\dots k}\{l_j(P)\} = 0$ again, and we return to the first case, i.e., $\max_{j=0,\dots k}\{l_j(P)\} \leq Q$.

Although the 1-PDP is $\mathcal{NP}$-*hard* (indeed it coincides with the TSP when the vehicle capacity is large enough), checking the feasibility of a path can be done in a linear time. It is also important to note that the feasibility of the path is independent of its orientation, such that a path that is feasible/infeasible if traversed forward, will be also feasible/infeasible if traversed backward [75]. This feature allows a solution construction algorithm to start a tour from any customer node, visit each node, including the depot, exactly once and return to the first node. This closed circular tour will correspond to starting and ending at the depot (see Section 9.4.3 for an example of a construction algorithm).

9.2 Related Work

Since this problem is $\mathcal{NP}$-*hard*, exact algorithms are only suitable for small problem sizes. For example, [74] presented a branch-and-cut exact algorithm to solve instances of up to 60 customers. To deal with large size problems, the same authors tried heuristic approaches in [75]. Two heuristic approaches have been presented in their work. The first approach starts with a construction heuristic that is based on an adaptation of a TSP Nearest-Neighbour (NN) insertion heuristic. However, travel distances were modified using a special formula that intends to penalize the use of edges connecting customers of the same type, i.e., edges connecting pickup customers together or delivery customers together. It was thought that this approach would be more likely to lead the construction algorithm to find feasible solutions. An improvement phase then follows, using 2-Opt and 3-Opt edge exchanges, to try to improve the feasibility and/or the total travel distance. The process of optimization is repeated several times, each time a new initial solution is constructed using a different starting node from the node list. The second approach is an incomplete optimization procedure, based on the branch-and-cut approach presented in [74] to find the best solution in a restricted feasible region.

In [73] a heuristic approach, named hybrid GRASP/VND, is proposed. The approach is based on combining two optimization heuristics that have been successfully applied

to combinatorial optimization problems. The first is called GRASP (Greedy Randomized Adaptive Search Procedure), and is based on a repetition of a construction phase and a local search phase. The construction process works by selecting the next element for insertion from a Restricted Candidate List (RCL), which has been previously created according to the benefit of inserting each node, and depends on the current state of the solution. There is also a probabilistic element in the choice of the next node from the RCL. On the other hand, the Variable Neighbourhood Search (VNS) approach is based on systematically changing the neighbourhood move, during the optimization process, each time a local optimum is reached. The new neighbourhood move is usually of higher order than the previous move, and it is applied to the same starting solution. Variable Neighbourhood Descent (VND) is a variant of VNS, where the local optimum found acts as the new starting point for the local search (see Section 2.3.6 for more details about the VNS and the VND approaches).

For the 1-PDP, the hybrid GRASP/VND approach in [73] is basically a GRASP, where the local search is performed using a VND procedure. In this heuristic, two VND algorithms have been applied with different neighbourhood moves in each algorithm. The first VND (called VND1) is applied in the improvement phase after the construction of the initial solution. The neighbourhood move used in VND1 is a classical 2-Opt, which is then followed by a 3-Opt move whenever a local optimum is reached. After the basic GRASP/VND, a further post-optimization phase is performed, using a second VND (called VND2). This time, however, the neighbourhood moves applied are a *move forward* and a *move backward* operators respectively. In the move forward operator, a customer is moved from its current position to a further position in the route. So for example, if a customer in position i is moved to position j with $j > i$, this requires that all customers in positions $i + 1, \ldots, j$ must be shifted backwards one position. The locations of other customers in the route do not change. The move backward operator works similarly, but j is now a position that precedes i in the tour, and intermediate customers have to be shifted forward one position. The VND algorithm used in this heuristic is shown in Algorithm 9.1 and the hybrid GRASP/VND algorithm is shown in Algorithm 9.2, both adopted from [73].

Algorithm 9.1: VND(x) Procedure [73].

1: **for** $(k = 1; k < max; k++)$ **do**
2: $x' \leftarrow LocalSearch(x, N_k(x))$
3: **if** (x' is better than x) **then**
4: $x \leftarrow x'$
5: Return x

Algorithm 9.2: Hybrid GRASP/VND Procedure [73].

1: **while** stopping condition is not satisfied **do**
2: $x \leftarrow GreedyRandomizedInitSol()$ {construction phase}
{improvement phase}
3: $x \leftarrow VND1(x)$ {edge-exchange neighbourhoods}
4: **if** x is feasible and improves the best solution x' **then**
5: $x' \leftarrow x$
{post-optimization}
6: $x' \leftarrow VND2(x')$ {vertex-exchange neighbourhood}
7: Return x'

A GA approach was introduced in [158] to solve the problem. The algorithm first starts by creating a population of feasible solutions using a new nearest-neighbour construction heuristic. The initial population is then optimized using a 2-Opt neighbourhood move. The most distinguishing feature of the algorithm is a new *pheromone-based* crossover operator, inspired from the Ant Colony Optimization (ACO) technique, where pheromone trails are updated each generation. During crossover, the selection of the next node to be inserted in the child is based on a probabilistic rule that takes into account the pheromone trail of the edge connecting the last inserted node and the potential new node, such that edges that have proved successful in the past are favoured. The crossover operator also considers the distance between the two nodes, as well as the demand of the new node and the current maximum and minimum loads carried by the vehicle. The offspring is further optimized using a 2-Opt local search, in which only feasible solutions are accepted, and only the closest neighbours to the current node are considered for edge exchanges. The mutation operator is based on a 3-exchange procedure. Since there are 5 legal ways in which 3 nodes could be exchanged, the move that yields the best result is the one performed in the mutation operator. The algorithm was tested on the benchmark data that are created by [75], producing the best so far results in most test cases.

The work done in [106] presents a simulation environment to solve the problem. The algorithm starts with a Modified Simulated Annealing (MSA) algorithm, followed by a manual

improvement phase. The SA approach is called Iterative Modified Simulated Annealing (IMSA). The idea is to create multiple neighbourhood feasible solutions Y from the current feasible solution X by performing a number of changes on X. The changes are done iteratively on X, starting from 1 change to a maximum of $nChanges_{max}$. For each change value, a fixed number of Y solutions is created, and this is done at the same SA temperature. This whole process is then repeated in an outer loop for a number $nCounts_{max}$ of iterations. The initial solution X is created based on a Greedy Random Sequence (GRS) algorithm, which is a nearest-neighbour heuristic with some randomness in selecting the nearest customer or the second nearest customer. A final manual optimization is then performed to improve the solution. Using a Graphical User Interface (GUI), the user selects a subgraph (of less than 15 nodes) from the optimal result obtained, upon which an improvement algorithm is applied. This is achieved by performing an Exact Permutation Algorithm (EPA) for small size problems, or improving a selected sub-route for large size problems. The algorithm was tested on some instances created in [7], and was found to be useful for practical applications.

9.3 Related Problems

Hernández-Pérez and Salazar-González in [75] presented an extensive survey of routing problems that are closely connected to the 1-PDP. Based on this survey, some important problems that are related to the 1-PDP are:

- **The Capacitated Traveling Salesman Problem with Pickup and Delivery (CTSPPD):** which is a special case of the 1-PDP, in which the demand of each pickup or delivery customer is restricted to one unit (e.g. [5]).
- **The Capacitated Dial-a-Ride Problem (CDARP):** in which the vehicle should move one-unit of commodity (for example a person) between pairs of customers (e.g. [124]).
- **The Pickup and Delivery Traveling Salesman Problem (PDTSP):** which is the same as the CDARP, but there is no restriction on the vehicle capacity (e.g. [127]).
- **The Traveling Salesman Problem with Backhauls (TSPB):** in which all delivery customers must be visited before all pickup customers (e.g. [57]).
- **The Traveling Salesman Problem with Pickup and Delivery (TSPPD):** due to the importance of this problem and its close connection to the 1-PDP, the following discussion provides more details about this problem.

The Traveling Salesman Problem with Pickup and Delivery (TSPPD)[1]

This problem was introduced in [110], and it is assumed here the commodity collected from pickup customers is different from the commodity delivered to delivery customers. The depot supplies all the demand of the delivery customers and collects all the supplies from the pickup customers, and both products must be accommodated in the vehicle without exceeding its capacity. An example of an application of this problem is when empty soft drink bottles are to be collected from homes or shops and delivered to the depot, and at the same time full bottles are supplied by the depot to be delivered to some customers [75]. For a TSPPD to be feasible, the vehicle capacity should at least be the maximum among the sum of the pickup demands and the delivery demands. This condition is not required for the 1-PDP, though, since the vehicle capacity could even be as small as the largest customer demand [75].

In mathematical terms, if we let $K = \max\{\sum_{i \in V:q_i>0} q_i, -\sum_{i \in V:q_i<0} q_i\}$, then for a TSPPD to be feasible Q must be $\geq K$. However, we can always assume, as mentioned in [110], that the TSPPD is in 'standard form', meaning that $K = Q$. Accordingly, we can also assume that in a feasible TSPPD route, the vehicle starts at the depot with a full load, delivers all the demand needed by the delivery customers, and at the same time collects the goods provided by the pickup customers. Finally, the vehicle will arrive at the depot fully loaded again, having collected all the goods from the pickup customers. This condition does not occur in the 1-PDP, since the product can be collected from the pickup customers and delivered to the delivery customers, with the depot only supplying or absorbing any additional amount [75]. In [74] it is observed that a TSPPD can be solved using the same algorithm as the 1-PDP if the TSPPD instance is transformed by duplicating the depot into two dummy 1-PDP customers, one collecting all the quantity supplied by the pickup customers, and the other providing all the quantity needed by the delivery customers.

9.4 Solving the 1-PDP Using an Evolutionary Perturbation Scheme (EPS)

In this part of our research we are trying to handle the 1-PDP using a technique based on 'perturbation' of the problem instance, first introduced in Codenotti *et al.* [27] for solving large instances of the TSP. The idea is to introduce a small perturbation to the

[1] This problem was briefly mentioned in our literature survey of pickup and delivery problems in Section 4.1 under the name: The Traveling Salesman Problem with Mixed Linehauls and Backhauls (TSPMB).

original problem instance P to transform it to a new instance P', for example by making a small change in the coordinates of the cities to be visited in the TSP. The new coordinates, which in turn will result in new separating distances between the nodes, will then be used to construct a solution for the new instance P'. The new solution is then evaluated, possibly after applying some local search method to improve it, relative to the *original* problem instance P to potentially replace the initial solution[2]. Figure 9.1(a) shows a typical product of a nearest-neighbour construction algorithm to an original small TSP instance, while Figure 9.1(b) shows an improved solution obtained after perturbing the coordinates of the original instance.

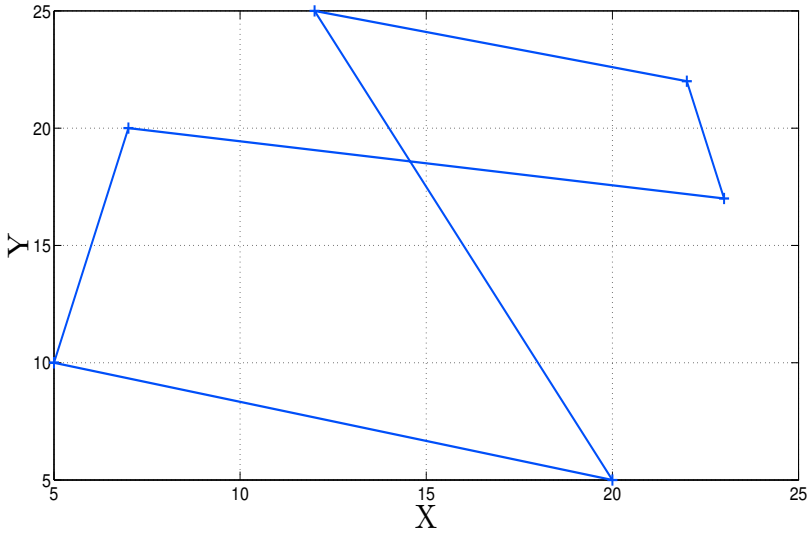
This idea was extended in [151], [152], [17] and [150], by applying a search algorithm to find a perturbation that will give a better solution. For example, a GA can be applied to optimize the set of perturbed coordinates. A simple greedy algorithm could then be used to create a solution for the TSP, using the perturbed coordinates rather than the original coordinates.

For the 1-PDP, we tried to apply the same idea of optimizing the set of perturbed coordinates, as done in [150] for the TSP, and in the PhD thesis by Matthew Morgan[3] for the Capacitated Vehicle Routing Problem (CVRP) [109]. We were motivated by the success of this simple algorithm on these routing problems, as evident by the experimental findings reported. For both problems, it was demonstrated that the technique was able to produce high quality solutions compared to simple solution construction heuristics without perturbation. The results obtained also compared favourably to published results from the literature in terms of both solution quality and processing time. Its applicability for the 1-PDP seemed to be viable, since a simple construction heuristic, such as the one described in [158], can be easily embedded within the evolutionary perturbation scheme, in which nodes' coordinates could be perturbed and optimized using a simple GA technique.

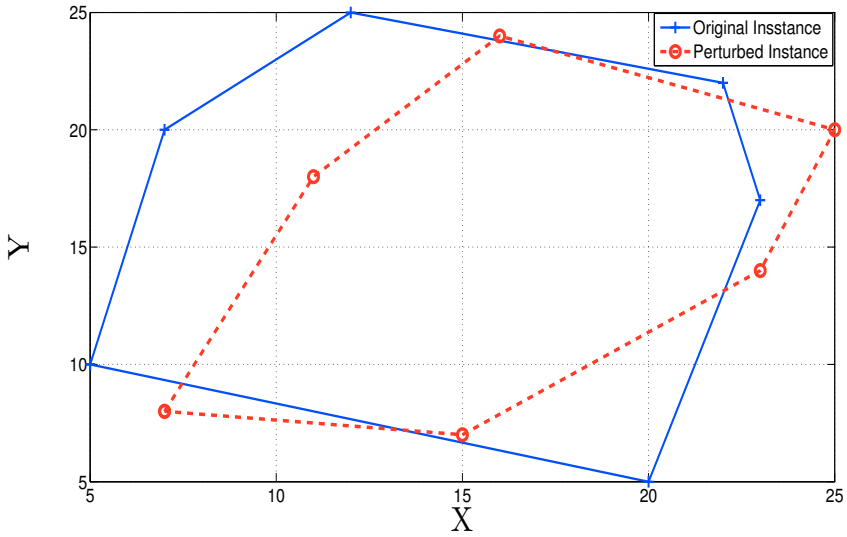
More specifically, in our perturbation scheme, we use a straightforward GA technique with a chromosome representing a list of customer locations, with each gene consisting of the x and y coordinates of the location. Traditional crossover and mutation operators are then applied to optimize the set of perturbed coordinates. The perturbation of each coordinate is done by applying a small change to the coordinate within a pre-specified radius, as suggested in [109]. As mentioned above, the underlying construction heuristic,

[2] A local search heuristic, applied to a solution for the new instance P' could be 'fooled' into finding better solutions for the original instance, because a local optimum for the original instance is not necessarily a local optimum for the new transformed instance.

[3] Matthew Morgan is a member of our research group (Scientific Computing and Optimization) in Cardiff School of Computer Science & Informatics, UK.



(a) TSP solution - original instance



(b) TSP solution after perturbation

Figure 9.1: TSP solution before and after perturbation.

used to create a solution using the perturbed coordinates, is a simple nearest-neighbour insertion heuristic, similar to the heuristic introduced in [158]. The created solution is then evaluated relative to the *original* coordinate set, and the cost of the solution is used as an objective function for the perturbed coordinates that were used to create the solution. Algorithm 9.3 describes the basic steps of our Evolutionary Perturbation Scheme (EPS).

Algorithm 9.3: The EPS Algorithm.

1: Initialize a population POP of perturbed coordinates from the original coordinates of problem instance P
2: **while** (stopping condition is not satisfied) **do**
3: **for** (i=0; i<NumCrossovers;i++) **do**
4: Select parents P_1 and P_2 from POP, using roulette wheel selection
5: Apply crossover to parents (P_1, P_2) to produce child C_1
6: Apply crossover to parents (P_2, P_1) to produce child C_2
7: with some probability, apply mutation to C_1 and C_2
8: Generate a solution s_1 from child C_1 and a solution s_2 from child C_2, using a simple problem-specific construction heuristic
9: decode solutions s_1 and s_2 to produce two real solutions r_1 and r_2 for the original problem instance P
10: Possibly apply an improvement heuristic to r_1 and r_2 {improvement heuristic is only applied to small problem sizes, due to time limitations}
11: Evaluate r_1 and r_2 and assign fitness values to C_1 and C_2 accordingly
12: Update POP by integrating the new generation and eliminating some worst individuals {i.e., steady state GA with overlapping populations}

Although the technique in general is simple, there are some parts that need careful consideration during the implementation of this algorithm. A critical issue is how to deal with infeasible solutions created during the search, and how to assign a suitable objective value for these solutions, since the total travel distance will no longer reflect the true cost of an infeasible solution. In the following subsections, we explain the basic components of the EPS.

9.4.1 The Encoding

The chromosome in our GA is an ordered list of customer locations starting from customer 1 to n. Each gene in the chromosome consists of the x and y coordinates of the customer location. Figure 9.2 shows the chromosome representation for both the original and the perturbed coordinates, depicted in Figure 9.1.

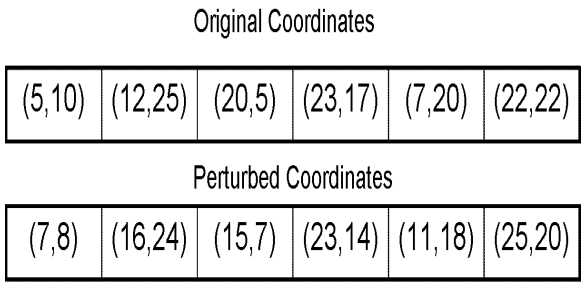


Figure 9.2: EPS chromosome representation.

9.4.2 The Initial Population

A chromosome is created by applying a small perturbation on the coordinates of each customer location, and storing the perturbed coordinates in the corresponding gene. The process is repeated for all individuals in the population. The perturbation scheme follows the recommendation of [109], such that each coordinate is shifted a small distance within a pre-specified radius. This radius is calculated as a function of the distance between the current node and its nearest neighbour. More specifically the new coordinates are given by the following equation:

$$new_x = old_x + (D \times S) \times r,\ new_y = old_y + (D \times S) \times r, \tag{9.2}$$

where r is a random number between -1 and 1, D is the distance between the current node and its nearest neighbour, and S is a scaling factor used to control the permitted shift.

9.4.3 The Nearest Neighbour Construction (NN-Construction)

The construction algorithm is called whenever the objective function of the chromosome has to be calculated. The chromosome is first used to create a distance matrix reflecting the current perturbed coordinates that are stored in the chromosome. This distance matrix is then passed to a simple greedy nearest-neighbour construction (NN-Construction) algorithm to create a corresponding solution. The construction algorithm we adopted in our research is the one described in [158] to create their initial GA population, with a slight modification. This algorithm works as follows.

1. Given CL [4], a user defined parameter, generate the list of CL closest neighbours to each node in the customer list, based on the current distance matrix;

2. Let $m = 1$, insert the depot node as the first customer $T(m)$ in the tour, and initialize both $MaxLoad$ and $MinLoad$ of the vehicle to be equal the demand of the depot;

3. If $m = n$, where n is the total number of customers including the depot, then stop; otherwise go to Step 4;

4. Search the CL closet neighbours of $T(m)$ for feasible candidates that have not been inserted before and can be added to the end of the tour without violating the vehicle capacity. If there are such customers, select the customer i having the largest demand (in absolute value) among them, and insert it as $T(m + 1)$, and update $MaxLoad$ and the $MinLoad$ of the vehicle, go to Step 5; otherwise go to Step 6;

5. $m = m + 1$, go to Step 3;

6. Search all customers that have not appeared in T for feasible candidates for insertion. If there are such customers, select the closest customer to the last inserted customer and insert it as $T(m + 1)$ and update $MaxLoad$ and the $MinLoad$ of the vehicle, go to Step 5; otherwise stop and declare an infeasible solution;

7. Return $m \leq n$, the number of nodes that have been successfully inserted in the solution before infeasibility (if any) is encountered.

The difference between our algorithm and the construction algorithm described in [158], and used to create a genetic population of random solutions to the 1-PDP, is that they start their tour from a randomly selected customer, while we always start the tour from the depot. We chose to fix the starting node to try to make the quality of the constructed solution only a function of the perturbed coordinates used to create the solution. Any randomness in the creation of the solution will interfere in the quality of the solution and will not reflect the true fitness of the perturbed coordinates. Thus, the learning process of the GA from generation to another may be obstructed. Similarly, some degree of randomness in Step 6 of the algorithm that was adopted in [158] has been eliminated in our algorithm. This is in fact essential in our scheme, because the solution constructed from the perturbed coordinates is not saved during the evolutionary process. Making the construction algorithm deterministic, by eliminating all randomness in the selection of nodes, will enable our algorithm to use the best perturbed coordinates (individual) to re-construct the best solution, when the evolutionary process terminates. In addition to

[4] CL stands for a constant that determines a selected number of *closet* neighbours to a certain node.

removing randomness, our algorithm stops once the solution becomes infeasible, while in [158], they repeat the construction algorithm starting from a new node, if the previous attempt resulted in an infeasible solution.

The selection of the customer with the largest demand in Step 4 of the construction algorithm is justified in [158] by giving priority in the insertion order to nodes that may be difficult to insert, because of their large demand. Also, as mentioned in Steps 4 and 6, the minimum load and the maximum load carried by the vehicle are updated each time a new node is inserted. This step is needed for the feasibility check of each candidate node. More specifically, according to Equation 9.1, to check the feasibility of insertion, the difference between $MaxLoad$ and $MinLoad$ after serving the candidate node, should not exceed its capacity, otherwise the potential candidate cannot be feasibly inserted at the end of the tour.

9.4.4 The Objective Function

To calculate the fitness of each chromosome in the population, the following steps are performed. First, a new distance matrix is calculated based on the set of perturbed coordinates stored in the chromosome. Then, this new distance matrix is passed to the construction algorithm, described in Section 9.4.3, to create a solution S that takes into account the new distances between the nodes. Finally, the objective function of a chromosome (C) is calculated using the following equation

$$F(C) = ((n+1) - m) \times Dist(S) \tag{9.3}$$

where n is the total number of nodes, m is the number of nodes that have been successfully inserted in the solution before infeasibility (if any) is encountered, and $Dist(S)$ is the 'true' total distance traveled when the solution S is decoded relative to the *original* coordinate set.

This objective function combines both the total travel distance and the degree of infeasibility in the generated solution. If the created solution is feasible, all nodes would have been inserted successfully, i.e. $m = n$. Accordingly, the fitness of this solution will be $Dist(S)$. On the other hand, if the solution is infeasible, $Dist(S)$ will be multiplied by a factor that increases with the degree of infeasibility. The smaller the number of nodes that have been successfully inserted, the larger the multiplication factor. Note also that in case of an infeasible solution, the distance value will be smaller than the distance of a feasible solution, because not all nodes have been successfully inserted.

9.4.5 The Operators

Mutation

The mutation operator first selects the number of nodes (genes) to be mutated. This number was chosen to be a random number between 1 and 20% of the total number of nodes in the chromosome. Then, the mutation operator applies Equation 9.2 to each gene selected at random.

Crossover

Different traditional crossover operators have been tried in our research. The most effective crossover operator seems to be the *EvenOdd* crossover operator, which works by selecting alternative genes from each parent and inserting them in the corresponding locations in the child. The roles of the parents are reversed for the second child.

9.4.6 Computational Experimentation

We implemented a steady state GA, with an 80% replacement, and the following parameters: population size= 200, number of generations=1000 for large problems (more than 60 customers) and 200 for small problems (from 20 to 60 customers), probability of mutation= 0.3, probability of crossover=0.8, and the perturbation scaling factor S in Equation 9.2 was set to 0.6, after some initial experimentation with different values. The neighbourhood size CL was chosen to be 2, i.e., only the closest two neighbours are checked for feasibility of insertion before all other nodes are checked. We found, experimentally, that this 'tight' neighbourhood size helped to reduce the runtime, since the neighbourhood list, which has to be created for all nodes whenever the construction algorithm is called, will be very small in size. Also, it appeared from trying several other values that this small value gave better results even for large size problems.

The Data Set: The algorithm was tested on instances created by [75]. There are 2 types of problem instances. Small instances have a number of customers n in {20, 30, 40, 50, 60}. For these instances, the optimum is known and was obtained using the exact method proposed in [74]. There are also large instances with n in {100, 200, 300, 400, 500}. For each combination of n and a different vehicle capacity Q in {10, 15, 20, 25, 30, 35, 40, 45, 1000}, 10 problem instances have been created and given the letters {'A' to 'J'}. So for example, a problem instance named ***N100q20A***, means the first instance (A) in

the 100-customers category (N100) with vehicle capacity 20 (q20). The data set and the results obtained in [75] and [73] can be downloaded from the Pickup and Delivery Site of Hernández-Pérez [5]: `http://webpages.ull.es/users/hhperez/PDsite/index.html`

Experimental Results: We briefly summarize here the experimental results of the EPS on test cases with the tightest vehicle capacity. These are the problems with $Q = 10$, and considered to be the hardest problems in the data set. We summarize the results on all problem sizes from 20 to 100. Our perturbation heuristic was run 20 times on each of these instances and the best result was recorded. Larger problem size, however, were found to be very time consuming and their testing was dealt with differently as will be explained later.

It should also be noted that there is a slight difference between the final testing version of the heuristic between small instances and large instances. This was based on preliminary experimentations and was intended to give the best possible results. For small instances (from 20 to 60 customers), the solution obtained from the NN-construction heuristic, explained in Section 9.4.3, was optimized using a Hill Climbing (HC) approach that uses a simple node swap neighbourhood move. The objective function (Equation 9.3) then uses the quality of the optimized solution as a fitness of the perturbed coordinates from which the solution was created. This is the same approach applied in the perturbation scheme of the CVRP in [109]. For larger problems, however, this technique could not be used due to the great increase in processing time. Therefore, the objective function was only based on the quality of the constructed solution without optimization. An optimization phase, though, was performed only on the best individual after the termination of the whole GA process. Thus, the set of perturbed coordinates that are represented in the best individual in the final generation, were used to construct the best solution obtained, using the NN-construction algorithm. This solution is then further optimized using an HC approach that uses a simple *move forward* operator, which was suggested in [73] (see Section 9.2 for more details about the move forward operator) . The results summarized here are the final results obtained after this optimization phase.

Results on Problem Sizes 20-100 Customers: Experimental results on problem sizes ranging from 20-100 customers indicated that the perturbation heuristic was able to achieve the optimal result in only 16 out of the 50 test instances, where the optimum is known (i.e., for problem sizes 20-60). Also, the optimum was achieved only for the smallest size pro-

[5] New best results were obtained by the GA in [158], but they do not appear in the pickup and delivery site yet.

blems of 20 and 30 customers. The heuristic, on the other hand, was unable to compete with previous heuristics from the literature for all larger size problems. The average relative difference between our results and the optimal or best known results ranged from 2% for 40 customers to 12% for 100 customers. The heuristic was also slower than the heuristic in [158]. For example the average run time for 100-customers problems was 43.8 seconds compared to 21.12 seconds reported by [158] for the same problem types.

Thus, the experimental results clearly indicate that the EPS achieved a limited success in dealing with the 1-PDP problem instances, especially as the problem size increases. Therefore, testing the algorithm on even larger size problems did not seem worthwhile. The perturbation scheme in all problem instances, though, always produced feasible solutions. This indicated to us that the heuristic may be used in cases where a quick and feasible solution is needed irrespective of its quality.

Results on Problem Sizes 200-500 Customers: For these problem sizes, we shifted our attention to testing the potential of the EPS in transforming infeasibility to feasibility, rather than competing with best known results. Hence, we only ran the algorithm for one generation and compared the result obtained, in terms of the quality of the solution constructed from the best individual in the population, with the result obtained when the NN-construction heuristic was run only once to produce an initial solution (using the original problem coordinates without perturbation). Remember that the NN-construction heuristic is deterministic in our scheme, since we removed all the randomness involved in the construction of such solution. Accordingly, there is only one initial solution that can be constructed using the original coordinates. If the NN-construction managed to insert all nodes in the solution whilst maintaining feasibility, the created solution is feasible. Otherwise, the quality of the created solution is determined by the number of nodes that have been successfully inserted before infeasibility is encountered. In the current experiment, the algorithm was again run 20 times on each problem instance, and we counted the number of times the EPS heuristic was able to generate a feasible solution after one generation.

The results of this experiment showed that the NN-construction heuristic failed to produce feasible solutions in all cases except one, which is the instance N400q10F. On the other hand, the EPS heuristic was able to find feasible solutions for all problem instances after one generation. In most test cases, the EPS was able to find a feasible solution in all 20 runs. In 9 out of the 40 problem instances the EPS heuristic did not produce a feasible solution in all runs. The minimum number of feasible solutions obtained were 3 out of the 20 runs.

These results, in general, seem to indicate that the EPS heuristic was in fact successful in transforming the frequent infeasibility problem of the NN-construction heuristic to feasibility. The processing time needed was also reasonable with an average of 1.13 seconds for 200-customers problems to 7.18 seconds for 500-customers problems.

9.5 Summary and Future Work

In this part of the research we investigated a new heuristic for the 1-PDP. The heuristic is a perturbation scheme that transforms the problem instance into a new one by performing a small shift in the coordinates of the nodes in the problem instance. A simple construction heuristic is used to construct a solution to the problem from the perturbed coordinates rather then the original coordinates, hoping to produce a better quality solution. The perturbed coordinates were also optimized using a simple GA technique.

The experimental results on a large number of test cases of different sizes indicated that the heuristic was in most cases able to improve the quality of the initial constructed solution to a large extent, removing infeasibility in all test cases. However, the final results obtained were in general of lesser quality than the best known or optimal results. The algorithm was also relatively slow, which is mostly due to the nature of the GA involved.

It should be noted that the EPS algorithm is highly sensitive to several underlying factors. The most important element seems to be the embedded construction heuristic. Previous attempts in the literature that used a similar technique, for example [109], made a considerable effort in testing several construction methods for the CVRP, to identify the best construction heuristic that can be used within the EPS. Nevertheless, the literature of the CVRP is relatively well-established and very rich with such heuristics. The literature on the 1-PDP, on the other hand, is scarce and only a small number of construction heuristics have been previously attempted. We chose from them the construction algorithm that seems to be relatively simple and fast, and also gave reasonable results within the context of other meta-heuristic techniques. This construction heuristic, however, apparently produce frequent infeasible solutions, as illustrated by the results obtained on large problem sizes. Having a large number of infeasible solutions will lower the quality of the overall population of perturbed coordinates, and make identifying the set of perturbed coordinates that lead to feasible and good quality solutions a difficult task for the EPS.

In addition, the EPS is also highly sensitive to the perturbation parameters and the selected perturbation neighbourhood shape and size around the coordinates. Fine tuning of these parameters was also a considerable part of the research done in [109]. In our research, we

simply relied on the final recommendations of [109] for their CVRP. However, obtaining the best perturbation parameters for the 1-PDP could allow the heuristic to achieve better results, but will, nevertheless, be very time consuming.

Another possibility for improvement is to modify the EPS, such that only feasible solution are allowed in the evolutionary process. Alternatively, the construction algorithm could be slightly adjusted, such that it will continue inserting nodes in the solution even after infeasibility is encountered. The objective function, then, should take into account the amount of infeasibility in the solution. This approach could allow the perturbation scheme to more easily transform the infeasibility to feasibility and improve the overall quality of the generated solutions. The viability of this 'modified' construction technique was demonstrated within the context of our VNS heuristic that was applied to the 1-PDP, as will be detailed in the next chapter.

Chapter 10

Solving the One-Commodity Pickup and Delivery Problem Using an Adaptive Hybrid VNS/SA Heuristic

In our attempt to investigate possible heuristics to solve the 1-PDP, and perhaps achieve better results than those achieved by the EPS heuristic, Variable Neighbourhood Search (VNS) seemed a reasonable choice that may have some potential. First, the idea is simple and can be easily adapted to the problem in hand. Second, it has been successfully applied to many routing and scheduling problems, for example to the TSP in [70] and the VRPTW in [19] and [119], and recently to the Periodic Vehicle Routing Problem with Time Windows (PVRPTW) in [118]. In this chapter, we explain our investigation of VNS for solving the 1-PDP. The VNS algorithm introduced here is hybridized with Simulated Annealing (SA) to escape local optima. We also employ adaptation of some search parameters for more efficient searching.

Section 10.1 briefly describes the VNS meta-heuristic in general, and highlights previous 1-PDP research that uses this approach. Section 10.2 explains in detail our proposed heuristic, which we will call an **Adaptive Hybrid VNS/SA (AVNS-SA)** technique for solving the 1-PDP. An outline of the complete algorithm is presented in Section 10.3. Experimental results of the algorithm are presented in Section 10.4, and a summary and some future directions will be presented in Section 10.5.

10.1 Variable Neighbourhood Search (VNS) and its Application to the 1-PDP

As previously mentioned in Section 2.3.6, VNS is a relatively new meta-heuristic that has been introduced by Hansen and Mladenović in [69] and [70]. The idea is to generate

new solutions that are distant from the incumbent solution, by systematically increasing the neighbourhood size within which the search is performed. The new solution replaces the current solution if it is better in quality. This way, many favourable characteristics of the incumbent solution will be preserved in the new generated solution. In addition, a local search is performed on the new solution to reach a local optimum within the current neighbourhood. For convenience, we repeat here the basic steps of the VNS algorithm as described in [70]:

- *Initialization*: Select the set of neighbourhood structures N_k, $(k = 1, ...k_{max})$, that will be used in the search; find an initial solution x; choose a stoping condition;
- *Repeat* the following until the stopping condition is met:
  1. Set $k \leftarrow 1$;
  2. Repeat the following steps until $k = k_{max}$:
     (a) *Shaking*: Generate a point x' at random from the k^{th} neighbourhood of x $(x' \in N_k(x))$;
     (b) *Local Search*: Apply some local search method with x' as initial solution; denote with x'' the so obtained local optimum;
     (c) *Move or not*: if the local optimum x'' is better than the incumbent, move there $(x \leftarrow x'')$, and continue the search with N_1 $(k \leftarrow 1)$; otherwise set $k \leftarrow k + 1$.

A variant of VNS, called Variable Neighbourhood Descent (VND)[1], has been tried for the 1-PDP as part of the heuristic proposed by Hernández-Pérez and Salazar-González in [73]. In their approach, the VND did not include a shaking phase, but only a local search that changes the neighbourhood move from 2-Opt to 3-Opt, whenever a local optimum is reached. The VND was embedded within another heuristic called GRASP (Greedy Randomized Adaptive Search Procedure) that repeatedly generates an initial solution upon which the VND is applied. Their approach is described in Algorithms 9.1 and 9.2 in the previous chapter. The heuristic in [73] achieved promising results that were better than the results obtained by the same authors using their previous heuristics suggested in [75]. However, their algorithm apparently was not fully capable of escaping the trap of local optima. This is evident by the fact that they had to use a post-optimization phase to improve the final result. According to the authors, this post-optimization often made the difference between beating the results obtained by their previous heuristic in [75] or not.

[1] See section 2.3.6 for more details about the VND algorithm.

The hybrid GRASP/VND heuristic was also outperformed by the GA of [158], in most test cases.

A possible shortcoming of the hybrid GRASP/VND heuristic is the absence of a shaking phase, which should help the diversification of the incumbent solution and allow the succeeding local search method to escape local optima. In our proposed approach, we try to apply the basic VNS, with both the shaking and the local search, hoping to overcome the limitation of this previous VND attempt on the 1-PDP.

10.2 The AVNS-SA Heuristic

To apply VNS to the 1-PDP, several choices should be made. These are outlined and explained below:

1. **The construction heuristic used to create the initial solution:** The construction algorithm we used is the same as the algorithm proposed by [158], which is the nearest-neighbour construction (NN-construction) heuristic, explained in Section 9.4.3.

2. **The set of neighbourhood structures used for shaking:** This is in fact the core of the VNS, and is the tool for *diversification* of the search. It is essential that the shaking procedure provides a balance between enough perturbation of the current solution, and also maintaining the most favourable characteristics of this solution. The shaking procedure should also allow the systematic change of the neighbourhood size. In our algorithm, we chose a simple move that displaces a sequence of nodes with or without inversion[2], where the number of nodes to be displaced could act as the variable neighbourhood size parameter that changes during the search.

3. **The local search procedure used to optimize the current solution:** This procedure is the tool that the VNS uses for *intensification* of the search. In our algorithm, we chose as a local search a simple 2-Opt edge exchange algorithm, based on the famous procedure proposed by Lin in [103]. We adopted the implementation described in [157] for the 2-Opt edge exchanges, having a computational complexity of $O(n^2)$, where n is the number of nodes in the problem instance. In addition, our local search is based on exhaustively testing all possible edge exchanges, and using *best improvement* as a replacement strategy.

[2]Inversion means to reverse the sequence of nodes in a manner similar to the inversion mutation operator shown in Figure 2.7(b)

4. **The acceptance criterion used by the VNS to accept a new solution as a replacement of the incumbent solution:** A *descent* only criterion will only accept a better solution, while a *descent-ascent* criterion will also accept a worse solution with some probability. Both techniques can be used within the VNS procedure as explained in [70]. In our approach we chose the second criterion, such that the acceptance is based on an SA acceptance probability, given a certain current temperature value.

In what follows we describe in detail how our proposed AVNS-SA approach was planned and developed.

10.2.1 The Initial Solution

To choose the initial solution that will be subject to the VNS, a number of solutions were created using the NN-construction heuristic (adopted from the algorithm in [158] and explained in Section 9.4.3), and the best solution was selected. However, in order to create different solutions, we re-introduced the randomness that has been removed in the EPS heuristic, for reasons previously explained. Specifically, the starting node was chosen at random, and also there was a small probability of choosing a random customer, rather than the nearest customer, in Step 6 of this algorithm (refer to the steps of the NN-construction algorithm in Section 9.4.3).

Also, rather than stopping the construction algorithm once infeasibility is encountered, we changed the algorithm so that if all remaining nodes are infeasible for insertion, the closest one to the last inserted node is selected, even if it is infeasible, hoping to fix the infeasibility later during the search. Thus, all nodes will be inserted in the solution and the degree of infeasibility is taken into consideration when calculating the cost of the solution. The value of CL, which is the neighbourhood size used to create the matrix of nearest neighbours to each node, was chosen to be 15 in the construction algorithm of [158]. In our approach, though, we tried all values of CL from 2 to 15 on selected test cases from both the small and the large size instances[3]. The value of CL that worked best for all problem instances tried was 7. Also, unlike the EPS heuristic, where a new neighbourhood matrix was created with each set of perturbed coordinates, the current approach requires only one static matrix to be created at the beginning of the run. Accordingly, a very 'tight' neighbourhood size was no longer needed to improve the run time[4].

For convenience, we show in detail below the steps of our construction algorithm:

[3]We selected test cases: N20q10A, N60q10A, N100q10A, and N500q10A.

[4]Recall that the value chosen for CL in the construction algorithm of our EPS heuristic, explained in Chapter 9, was 2.

1. Given CL, a user defined parameter, generate the list of CL closest neighbours to each node in the customer list;

2. Let $m = 1$, insert a random node as the first customer $T(m)$ in the tour, and initialize both $MaxLoad$ and $MinLoad$ of the vehicle to be equal to the demand of this customer;

3. If $m = n$, where n is the total number of customers including the depot, then stop; otherwise go to Step 4;

4. Search the CL closet neighbours of $T(m)$ for feasible candidates that have not been inserted before and can be added to the end of the tour without violating the vehicle capacity. If there are such customers, select the customer having the largest demand (in absolute value) among them, insert it as $T(m + 1)$, and update $MaxLoad$ and the $MinLoad$ of the vehicle, then go to Step 5; otherwise go to Step 6;

5. $m = m + 1$, go to Step 3;

6. Search all customers that have not appeared in T for feasible candidates for insertion. If there are such customers, select the closest feasible customer to the last inserted customer with probability 0.7, or select a random feasible customer with probability 0.3; if there are no remaining feasible customers, select the closest infeasible customer to the last inserted node; insert the selected node as $T(m+1)$ and update $MaxLoad$ and the $MinLoad$ of the vehicle; go to Step 5.

10.2.2 The Objective Function

The objective function used to estimate the solution quality was set to

$$F(S) = (NCV(S) + 1) \times Dist(S), \tag{10.1}$$

where $NCV(S)$ is the number of capacity violations along the route, i.e., the number of nodes at which the feasibility check, described by Equation 9.1, is not satisfied. At these locations, the vehicle could be 'carrying' an excess of its allowed capacity, or could be in shortage of the necessary quantity to be delivered. The larger the number of capacity violations, the larger the cost of the solution. $Dist(S)$ is the total distance of the solution, given the current visiting order of nodes. If there are no capacity violations in the route, i.e., the route is feasible, the total distance will be the sole measure of the solution quality. Thus, this multiplicative objective function will penalize the number of capacity violations in the route, by significantly increasing the cost of solutions that have a large number of

capacity violations, compared to feasible solutions, whose cost will only be measured by the total distance traveled. In addition, no arbitrary fine tuning of penalty weights is needed for this objective function.

10.2.3 The Initial SA Temperature

As previously mentioned, we used an SA acceptance criterion within the VNS approach. A critical part of any SA algorithm is the SA schedule, especially the starting temperature and the temperature reduction factor. These parameters significantly affect the performance of the SA algorithm, and their adjustment to fit a wide range of problem instances and sizes is a daunting task and very time consuming. To calculate the SA starting temperature for each problem instance individually, we again adopted the approach proposed by [40]. Recall that this approach was used in both our SA algorithms that handled the SV-PDPTW and the MV-PDPTW, as explained in Chapters 5 and 8. The procedure for creating the initial solution and calculating the SA temperature is shown in Algorithm 10.1.

Algorithm 10.1: Find Initial Solution & Calculate Starting Temperature.

Create a starting solution s using the NN-construction heuristic
$s^* \leftarrow s$ {$s*$ is the best so far solution}
{Initialize $Pstart$, the starting acceptance probability}
Let $Pstart$ = a large value {We used 0.9}
$\Delta_{avg} \leftarrow 0$
{Generate n solutions using the NN-construction (We used $n = 1000$)}
for $(i = 0; i < n; i++)$ **do**
    generate a new solution s' using the NN-construction heuristic
    if $(Objective(s') < Objective(s^*))$ **then**
        $s^* \leftarrow s'$
    $\Delta \leftarrow |Objective(s') - Objective(s)|$
    $\Delta_{avg} \leftarrow \Delta_{avg} + \Delta$
$\Delta_{avg} \leftarrow \Delta_{avg}/n$
$T_0 \leftarrow -\Delta_{avg}/\log(Pstart)$
Return s^*, the initial solution from which the AVNS-SA algorithm will progress, and the initial SA temperature T_0

10.2.4 The Shaking Procedure

As mentioned previously, we chose as a shaking procedure a displacement of a sequence of nodes with some probability of inverting this sequence (a 50% chance of inversion was adopted in our algorithm). This move is popular in the VRP literature, and especially for solving the TSP, but to the best of our knowledge it has not been tried before for the 1-PDP.

Our VNS algorithm passes the current neighbourhood size ($NhSize$) as a parameter to the shaking procedure, which will in turn use this parameter as the *maximum* possible number of nodes that will be displaced. Specifically, the number of nodes to be displaced is a random number between 1 and $NhSize$. So even for large values of $NhSize$, small sequences of nodes could still be displaced, and in fact there is a 'bias' toward such small moves, since they have a chance of being executed in all neighbourhood sizes. This is intended to prevent a large disruption of the current solution, and is recommended by some VNS implementations, as in [119] for the VRPTW. Both the starting position of the selected sequence and its new position within the route are chosen at random.

10.2.5 The Maximum Neighbourhood Size

Our VNS algorithm repeats the shaking followed by the local search for the current solution for all $NhSize = 1, 2, 3...NhSize_{max}$, where $NhSize_{max}$ is the maximum sequence of nodes that could be displaced. $NhSize_{max}$ should be chosen in a way that allows enough perturbation of the solution without disturbing its favourable characteristics. Naturally, the value for $NhSize_{max}$ in our shaking procedure must be smaller than n, where n is the total number of nodes in the current problem instance. Also, to make $NhSize_{max}$ adaptable for any problem instance, it must be calculated relative to n, and not fixed for all instances. However, large values of $NhSize_{max}$ will increase the computational cost and slow down the optimization process. So, in order to reduce the computational cost, we tried different fractions of n ($n/2$, $n/3$, $n/4$), on selected problem instances from both the small and large problem instances[5]. These values of $NhSize_{max}$, though, still resulted in a very long processing time for large problem instances. We then tried the value $2 \times \sqrt{n}$, which was recommended by [106], for the number of changes ($nChanges_{max}$) in their Iterated Modified Simulated Annealing (IMSA) approach to the 1-PDP, as previously explained in Section 9.2. $NhSize_{max} = 2 \times \sqrt{n}$ gave the best results among all values tested, in terms of the balance between processing time and solution quality, for all instances tried in this experiment.

[5] Again we used test cases: N20q10A, N60q10A, N100q10A, and N500q10A.

10.2.6 A Sequence of VNS Runs

During our early experimentation with the VNS algorithm, we found that to reach good solutions for the 1-PDP, several runs of the VNS procedure should be performed. Each run starts from the final solution obtained in the previous run. The VNS could be repeated for a fixed number of iterations, or until no improvement is realized in the current solution for a number of attempted iterations. We chose the second approach, and stopped the repetition of the VNS when no improvement happens in 5 consecutive iterations.

However, we realized that during the first VNS run, improvement happens quickly for most neighbourhood sizes, even for the large ones among them. Subsequent VNS runs, though, usually respond only to smaller changes in the solution. In other words, smaller neighbourhood sizes seem to be more beneficial in subsequent VNS runs, since larger changes may cause a disturbance of the current solution and reduce its quality. Accordingly, after each VNS run, $NhSize_{max}$ was reduced by a fraction of its value. To choose the reduction scheme, we tried the fractions 1/2, 1/3, 1/4 and 1/5 of $NhSize_{max}$, on the same selected test instances previously indicated in Sections 10.2.1 and 10.2.5. We finally chose the reduction factor 1/4, based on the quality of the results obtained.

As previously mentioned, the VNS procedure is repeated several times, and only stops when no improvement happens in the current solution for a number of consecutive iterations. Thus, reducing $NhSize_{max}$ cannot continue indefinitely, because otherwise later runs may not perform any shaking at all. In order to maintain a reasonable number of nodes to be displaced in the shaking procedure during later runs, the reduction of $NhSize_{max}$ is repeated until it reaches a certain minimum value, at which stage no further reduction is performed, and the VNS procedure uses the current $NhSize_{max}$ for all remaining runs. The minimum value of $NhSize_{max}$ was chosen to be the same as the reduction factor, i.e., $NhSize_{max}/4$, in order to reduce the number of parameters that need to be adjusted in the algorithm.

10.2.7 Stopping and Replacement Criteria for Individual VNS Runs

The VNS procedure repeats the shaking and the local search for all values of $NhSize = 1, 2, 3...NhSize_{max}$. However, in some cases, the current solution may not respond to changes in the neighbourhood size and reach a stage of stagnation. Therefore, rather than indiscriminately increasing $NhSize$ up to the pre-specified maximum, we chose to also end each VNS run when the solution has not changed for a certain number of consecutive attempts of increasing $NhSize$. For more flexibility and robustness, the

number of attempts should be chosen relative to the current maximum neighbourhood size ($NhSize_{max}$), and not fixed for all problem instances. Similar to how the reduction scheme was chosen (as explained in Section 10.2.6), the number of attempts was again set to $NhSize_{max}/4$. Thus, the VNS will be *adaptive* in the sense that it will stop the shaking and the local search cycle, when no benefit seems to be realized from increasing $NhSize$. It is also adaptive from another perspective, since $NhSize_{max}$ passed to the VNS is not fixed and depends on the current stage of the run, as previously discussed in Section 10.2.6

Also, as indicated before, we chose an SA acceptance criterion to replace the current solution within each VNS run. Initial experimentation showed that accepting worse solutions with some probability improved the final result obtained, although it is more time consuming. Thus, the SA component of the algorithm works by allowing the new solution resulting after the shaking and the local search to replace the current solution, even if it is worse in quality. The acceptance criterion is the same as the usual SA acceptance and depends on the difference between the objective value of the new solution and the current solution, and the present temperature value.

In our algorithm, the VNS procedure repeats the shaking and the local search for the ***same*** neighbourhood size ($NhSize$) for a number of trials. The number of trials is incremented only when a solution worse than the incumbent appears in the current $NhSize$. When the number of trials reaches a certain pre-defined limit, the shaking and the local search cycle stops for the current $NhSize$, and the VNS moves on to the next $NhSize$. Also for each $NhSize$, the current temperature is decremented in the current iteration each time the new solution is worse than the current solution.

10.2.8 Updating the SA Starting Temperature

Normally, by the end of each complete VNS run, the SA temperature would have reached a small value that should not permit the acceptance of any worse solutions. If we started the new VNS run with such a small value, there would be no benefit to the SA acceptance, since all worse solutions would be rejected. On the other hand, starting a new VNS run with the initial temperature too high is also not beneficial, since many worse solutions would be accepted, possibly causing the destruction of the current solution. To achieve a balance between these two situations, the final temperature value reached in the current VNS run was *doubled* before the beginning of the next VNS run.

10.3 The Complete AVNS-SA Algorithm

To put it all together, Algorithm 10.2 shows the main Adaptive VNS-SA (AVNS-SA) heuristic, which will invoke the VNSSA procedure, described in Algorithm 10.3.

Algorithm 10.2: Adaptive VNS-SA (AVNS-SA) Algorithm.

1: Find an initial solution ($InitSol$) and the starting temperature ($StartTemp$) using Algorithm 10.1.
2: $NhSize_{max} \leftarrow 2 \times \sqrt{n}$, where n is the number of nodes
3: $Decrement \leftarrow NhSize_{max}/m_1$ {We used $m_1 = 4$}
4: $MaxStagnation \leftarrow NhSize_{max}/m_2$ {We used $m_2 = 4$}
5: Initialize $MaxAttempts$ to a small number {We used 5}
6: $NoImprovement \leftarrow 0$
7: **repeat**
8: $NewSol = VNSSA(InitSol, NhSize_{max}, StartTemp, MaxStagnation)$
9: **if** ($NhSize_{max} > Decrement$) **then**
10: $NhSize_{max} \leftarrow NhSize_{max} - Decrement$
11: **else**
12: $NhSize_{max} \leftarrow Decrement$
13: **if** ($NewSol$ is not better than $InitSol$) **then**
14: $NoImprovement++$
15: **else**
16: $NoImprovement \leftarrow 0$
17: $InitSol \leftarrow NewSol$
18: $StartTemp \leftarrow StartTemp \times 2$
19: **until** ($NoImprovement$ reaches $MaxAttempts$)

Algorithm 10.3: The VNSSA Algorithm.

Input: $InitSol$, $NhSize_{max}$, $StartTemp$, $MaxStagnation$
Output: a new, possibly improved, solution X
$k \leftarrow 0$ { Initialize the current neighbourhood size k }
$Stagnation \leftarrow 0$
$NumTrials \leftarrow LIMIT$ {$LIMIT$ is the maximum allowed number of trials for the current neighbourhood size (we used 30 trials)}
$X \leftarrow InitSol$
repeat
  $k++$ { Increment the current neighbourhood size}
  $Trials \leftarrow 0$
  while ($Trials < NumTrials$) **do**
    $Shaking(X, XI, k)$ {displacing a sequence of nodes in X up to a maximum of k, with or without inversion. The result is stored in XI}
    $LocalSearch(XI, XII)$ {local search is done on XI using 2-Opt. The result is stored in XII}
    if ($Objective(XII) < Objective(X)$) **then**
      $X \leftarrow XII$
    else
      Accept XII using SA acceptance probability
      $StartTemp \leftarrow StartTemp \times \alpha$ {Decrement current temperature (we used $\alpha = 0.99$)}
      $Trials++$
    end while
  if (X did not change in the last iteration (i.e., for the current neighbourhood size k)) **then**
    $Stagnation++$
  else
    $Stagnation = 0$
until ($Stagnation = MaxStagnation$) or ($k = NhSize_{max}$)
Return X

10.4 Experimental Results

To test the performance of our AVNS-SA algorithm, we used the same test cases described in Section 9.4.6. We ran the algorithm 5 times on each test case from 20-300 customers. On the other hand, only one run was performed on test cases of 400 and 500 customers, due to time limitation. In this experiment a number of computers with different specifications were used to run the algorithm. For this reason, the run times we quote in this section will vary according to the platform. Nevertheless, our timings give useful estimates of the time requirements of the algorithm.

Table 10.1 shows the results achieved by the AVNS-SA algorithm for small size problems of 20-60 customers. As previously explained in Section 9.4.6, a problem instance named **N20q10A**, for example, means the first instance (A) in the 20-customers category (N20), with vehicle capacity 10 (q10). Table 10.1 shows the best result obtained in the 5 runs, the number of times the best result appeared in the 5 runs (Num Seen), and the average result of the 5 runs. Finally the table also shows the optimum result found by the exact algorithm in [74].

Table 10.1: AVNS-SA Results (20-60 Customers)

Name	Best	Num Seen	Average	Optimum
N20q10A	**4963**	4	4974.6	4963
N20q10B	**4976**	5	4976	4976
N20q10C	**6333**	3	6390.4	6333
N20q10D	**6280**	4	6341	6280
N20q10E	**6415**	5	6415	6415
N20q10F	**4805**	3	4808.6	4805
N20q10G	**5119**	5	5119	5119
N20q10H	**5594**	5	5594	5594
N20q10I	5157	2	5195.2	5130
N20q10J	**4410**	5	4410	4410
N30q10A	**6403**	1	6455	6403
N30q10B	**6603**	5	6603	6603
N30q10C	**6486**	1	6576.2	6486
N30q10D	**6652**	2	6746	6652
N30q10E	**6070**	5	6070	6070
N30q10F	**5737**	4	5817.4	5737
N30q10G	**9371**	1	9388.2	9371

Continued on Next Page...

Table 10.1 . . . Continued from Previous Page

Name	Best	Num Seen	Average	Optimum
N30q10H	**6431**	1	6451.4	6431
N30q10I	**5821**	3	5909	5821
N30q10J	6271	3	6344.2	6187
N40q10A	**7173**	2	7246.2	7173
N40q10B	**6557**	1	6621.2	6557
N40q10C	**7528**	3	7548.2	7528
N40q10D	8073	1	8206.2	8059
N40q10E	**6928**	4	6941.4	6928
N40q10F	**7506**	1	7577.4	7506
N40q10G	7669	2	7707.4	7624
N40q10H	**6791**	2	6870.6	6791
N40q10I	**7215**	1	7267.8	7215
N40q10J	**6512**	4	6516.4	6512
N50q10A	**6987**	3	7031.2	6987
N50q10B	**9488**	1	9603.6	9488
N50q10C	**9110**	1	9178.4	9110
N50q10D	10464	1	10678	10260
N50q10E	**9492**	1	9644	9492
N50q10F	**8684**	2	8747.2	8684
N50q10G	**7126**	1	7240.8	7126
N50q10H	**8885**	1	8982.6	8885
N50q10I	8404	1	8486.8	8329
N50q10J	**8456**	3	8638.6	8456
N60q10A	**8602**	2	8646.8	8602
N60q10B	**8514**	2	8571.8	8514
N60q10C	9483	1	9553.8	9453
N60q10D	11061	1	11324.2	11059
N60q10E	**9487**	1	9638	9487
N60q10F	**9063**	1	9250	9063
N60q10G	8998	2	9101.8	8912
N60q10H	**8424**	1	8473.4	8424
N60q10I	9524	1	9577.2	9394
N60q10J	8844	1	8982.2	8750

Table 10.1 shows that the algorithm was able to achieve the optimum results at least once in the 5 runs for 39 out of the 50 test cases. These are shown in boldface in the table. The maximum relative difference to the optimum was less than 2% among the 11 cases where the optimum was not found, which was for test case N50q10D. The processing time was reasonable with an average ranging from 0.66 seconds for 20 customers problems to 47.79 seconds for 60 customers problems.

Table 10.2 shows the results of the AVNS-SA algorithm on large size problems, from 100 to 500 customers. The table shows the best result achieved and the average result of the 5 runs. Note that this average is replaced by the best result for problems of size 400 and 500, since the algorithm was run only once on these problems. Finally, the previous best known results are also shown in the last column. Only 6 out of the previous best known results are attributed to [73], which are the results for test cases (N300Q10C, N400Q10A, N500Q10A, N500Q10D, N500Q10E, N500Q10H). All remaining previous best results were found by the GA in [158]. The best result achieved by the AVNS-SA is shown in boldface if it was better than the previous best known result.

Table 10.2: AVNS-SA Results (100-500 customers)

Name	**Best**	**Average**	**Previous Best**
N100q10A	**11741**	12173.8	11828
N100q10B	**13066**	13410.6	13114
N100q10C	**13893**	14073.8	13977
N100q10D	14328	14567.2	**14253**
N100q10E	11430	11823.6	**11411**
N100q10F	11813	11947	**11644**
N100q10G	**12025**	12118	12038
N100q10H	12821	12844	**12818**
N100q10I	**14025**	14278.6	14032
N100q10J	13476	13642.8	**13297**
N200q10A	17690	17849.2	**17686**
N200q10B	**17618**	17887.8	17798
N200q10C	16535	16626.6	**16466**
N200q10D	**21228**	21594.2	21306
N200q10E	**19220**	19485.2	19299
N200q10F	**21627**	21677.4	21910
N200q10G	**17361**	17634	17712
N200q10H	**20953**	21191.4	21276

Continued on Next Page...

Table 10.2 . . . Continued from Previous Page

Name	**Best**	**Average**	**Previous Best**
N200q10I	**18020**	18328.2	18380
N200q10J	19016	19240.4	**18970**
N300q10A	**22940**	23163	23242
N300q10B	**22473**	22920.4	22934
N300q10C	**21183**	21454	21800
N300q10D	**25220**	25500.6	25883
N300q10E	**26636**	26934	27367
N300q10F	**24042**	24290.6	24826
N300q10G	**23683**	23945	23868
N300q10H	**21555**	21824.6	21625
N300q10I	**23871**	24110.2	24513
N300q10J	**22503**	22688.8	22810
N400q10A	**30657**	30657	31486
N400q10B	**24248**	24248	24262
N400q10C	**27853**	27853	28741
N400q10D	**23750**	23750	24508
N400q10E	**24798**	24798	25071
N400q10F	**26625**	26625	26681
N400q10G	23925	23925	**23891**
N400q10H	25628	25628	**25348**
N400q10I	**28262**	28262	28714
N400q10J	**24847**	24847	26010
N500q10A	**27904**	2790	28742
N500q10B	**26612**	26612	26648
N500q10C	**30247**	30247	30701
N500q10D	**29875**	29875	30794
N500q10E	**29978**	29978	30674
N500q10F	**28527**	28527	28882
N500q10G	**26171**	26171	27107
N500q10H	**35805**	35805	36857
N500q10I	**30247**	30247	30796
N500q10J	**30428**	30428	31255

Table 10.2 shows that the AVNS-SA algorithm was able to improve the previous best known results for 50% of the 100 test cases, 70% of the 200 test cases, 100% of the 300 test cases, 80% of the 400 test cases, and finally 100% of the 500 test cases.

The overall average of the results in the 5 runs for all test cases of size 100 is 13087.94, which is only 1% worse than the average result of the GA in [158], having a value of 12954.16. Moreover, our overall average for the 200 test cases is 19151.44, while the overall average of the heuristic in [158] for the same test cases in 10 runs is 19339.48, i.e., our results account for an improvement of approximately 1%. On the other hand, the overall average of our results for the 300 test cases was 23683.12, with an improvement of more than 2% compared the overall average of their results for the same test cases, which is 24224.28.

Also, our average for the results of the 10 test cases of size 400 was 26059.3, which accounts for an approximately 2% improvement over the average of the best results of [158], having the value 26490.4. In addition, our average result of the 10 test cases of size 500 was 29579.4. This is an improvement of approximately 3% over the average of the best results of [158], having the value 30377.1 for the same instances. These results also indicate that our algorithm performs even better on larger size problems. The average processing time in this experiment ranged from approximately 542.22 seconds for 100 customers instances to 151103.04 seconds for 500 customers instances.

To further test the robustness of the AVNS-SA algorithm, we performed an additional experiment by running the algorithm on 100-customers problems with a vehicle capacity of 20 and 40. The algorithm was run 10 times on each test case. The results of this experiment is shown in table 10.3. The table shows the best result achieved among the 10 runs, the average result of the 10 runs, and the best known results published in [73]. Results of the algorithm that are the same or better than the best known results are shown in boldface.

The results in the table show that the algorithm was able to achieve better than the previous best known results in 4 out of 10 test cases, for vehicle capacity $Q = 20$. It was also able to achieve the same result as the best known result for 6 out of 10 cases, for vehicle capacity $Q = 40$. The later results might as well be the optimum results, since they have also appeared as best results in the heuristic of [75], and each one appeared more than once among our 10 runs experiment. The average processing time for $Q = 20$ instances was 155.76 seconds, and for $Q = 40$ instances was 146.17 seconds.

Table 10.3: AVNS-SA Results (100 Customers with Q=20 and Q=40).

Name	Best	Average	Best Known
N100q20A	8650	8768.6	8616
N100q20B	**9533**	9628.8	9536
N100q20C	**9954**	10093	9993
N100q20D	**10015**	10351	10064
N100q20E	8864	9710.35	8838
N100q20F	**9004**	9127.9	9029
N100q20G	8986	9138.7	8865
N100q20H	9561	9692.3	9495
N100q20I	10017	10411	10005
N100q20J	9769	9873.5	9742
N100q40A	**7938**	8005.8	7938
N100q40B	**8124**	8234.1	8124
N100q40C	**8441**	8508.9	8441
N100q40D	8336	8402.9	8264
N100q40E	**7960**	8037	7960
N100q40F	**8074**	8131.9	8074
N100q40G	8181	8237.9	8168
N100q40H	**7992**	8010.5	7992
N100q40I	8478	8551.3	8440
N100q40J	8261	8311.2	8255

Finally, to test the effect of the SA acceptance on the performance of the algorithm, we ran the heuristic twice on test case N100q10A. In the first run the SA acceptance was used, and the result of each VNS iteration was recorded. The final result produced was 11751. In the second run, we removed the SA acceptance, and used an HC criterion, where only solutions of lower cost are accepted in each VNS iteration. The final result produced was 12313. Figure 10.1 shows the performance of both variants. It is clear from the figure that the SA version explored a wider area of the search space, before finally reaching a better result than the one achieved by the HC variant, which converged to a suboptimal solution. The processing time, though, was in favour of the HC variant, with 218.33 seconds compared to 552.11 seconds for the SA variant.

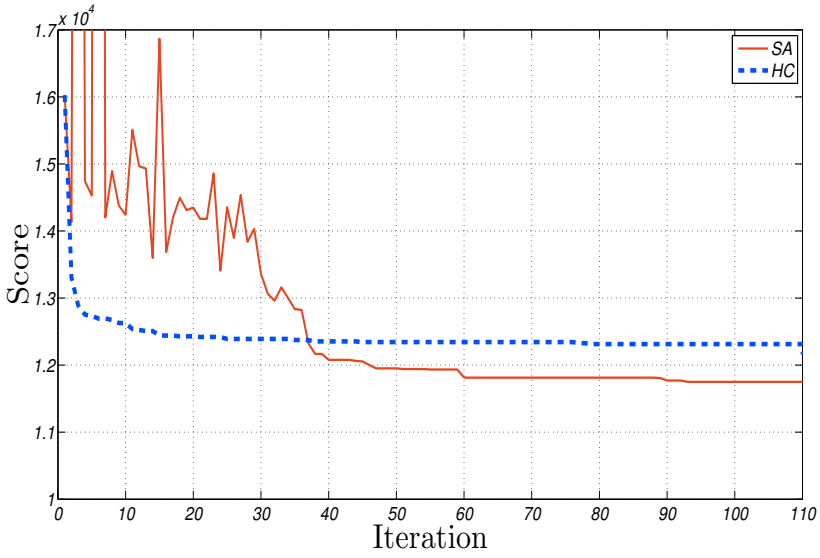


Figure 10.1: SA against HC for N100q10A.

Contrary to the exceptional results achieved by our AVNS-SA algorithm, its processing time in general was to a large extent disappointing. For example, the average processing time for 100 customers problems was 542.22 seconds, while the processing time reported by [158] was 21.12 seconds for the same problem category.

10.5 Summary and Future Work

In this part of the research, we investigated a VNS approach to the 1-PDP problem. The algorithm is distinguished by performing the VNS repeatedly, each time starting from the final solution obtained in the previous run. Also, the algorithm is *adaptive*, in the sense that the maximum neighbourhood size allowed in each VNS run is not fixed and depends on the current stage of the run. Early runs are allowed to perform wider jumps in the solution space from the current solution, using large neighbourhood sizes. Later runs, on the other hand, are only allowed smaller explorations of the search space, in the vicinity of the current solution, to maintain the solution quality.

The stopping criterion for each VNS run is also adaptive and depends on the improvement realized in the current solution. The VNS is terminated when a further increase the neighbourhood size seems unhelpful. During each VNS run, an SA acceptance criterion is used to allow the algorithm to escape local optima, and explore a wider area of the search space. In addition, we used a new neighbourhood move in the shaking part of

the algorithm, which has not been tried before for the 1-PDP. This move, which is based on a displacement of a sequence of nodes with or without inversion, seems to work perfectly well within the context of the VNS for this problem, as evident by the final results obtained for the overall heuristic.

Experimental results on a large number of problem instances indicate that our algorithm outperforms previous heuristics in most hard test cases, where the vehicle capacity is smallest. This is especially noticeable for large problem sizes. The algorithm was able to achieve the optimum results for all but few test cases in the small size problems, and was able to improve the previous best known results for 90% of the large test cases. The algorithm is also robust enough, since it performs equally well on a wide range of problem instances, e.g. instances with a different vehicle capacity, without the need for any parameter adjustment.

These distinguished results, though, come at the expense of computation time. Although we cannot provide an accurate analysis at this stage, because of the use of different processors to run the experiments, we recognize that the running time of the algorithm is rather too long. Of course, this is more noticeable for very large problem sizes, i.e., more than 200 customers.

In the future, we will continue investigating possible techniques to reduce the run time. Some attempts include reducing the number of VNS runs, and changing the stopping criteria for each individual run. For example, our computational experimentation indicated that some problem instances needed fewer than 5 consecutive attempts (without improvement) to reach the best results. However, for other instances, reducing the maximum number of attempts to less than 5 may cause the algorithm to prematurely stop and produce lower quality result. More investigation of the best termination criterion is therefore needed to reduce the overall processing time. Other possible improvement attempts, with respect to the run time, should be oriented towards the local search procedure, since it is the most time consuming part of the algorithm. For example, we can try to reduce the number of calls to this algorithm, or make it optimize only part of the solution rather than whole solution. This could possibly be done by changing the exhaustive search component of the 2-Opt algorithm to some random or selective search that does not process all nodes in the solution. A better analysis of the neighbourhood structure could also be beneficial in enhancing the execution time, for example by eliminating solutions that may not seem promising from further processing.

Having completed our investigation of several heuristics and meta-heuristics on selected pickup and delivery problems, we present in the next chapter a brief discussion of how this research relates to commercial transportation software and industrial applications.

Chapter 11

The Research and Real-Life Applications

Efficient transportation has become vital for today's dynamic society. In the European Union, the transportation sector constitutes more than 10% of Gross Domestic Product (GDP) and employs more than 10 million people [1]. Transportation volume continues to increase rapidly every day, as a result of economic growth and globalization, compared to a limited expansion in roads and networks capacities.

Transportation demand is not always geographically balanced. In addition, the lack of coordination between manufacturers, shippers and carriers in supply chains can lead to inefficient usage of natural and human resources. In many cases, transportation planning is done manually. However, the advent of today's technology- including high speed computers, digital cellular phones, Geographic Information Systems (GIS), Geographic Positioning Systems (GPS), navigation and tracking technologies, wireless data communication, digital mapping and web-based services- has increased the demand for more efficient commercial software for route planning. If applied on a large scale, commercial software can lead to enormous savings, both economically and environmentally. Vendors of software tools claim that the reduction in cost may range from 5% - 30% [76]. Given the huge volume of today's transportation, such cost reduction is in fact very significant. Besides cost reduction, efficient routing can greatly reduce the environmental impact of transportation. For example, in a recent survey of UK brewery, Paragon Software Systems, Inc.[1] identified savings of more than 2.5 million miles- corresponding to 3,700 tons of CO_2- a year, as a result of more efficient routing [68].

To meet this demand, research in vehicle routing and scheduling has grown substantially in the last few decades [48]. As previously discussed in Chapter 3, a huge number of problem variants, different problem constraints, and operating scenarios have been investigated. In fact, research in this field is central to the development of efficient decision

[1] http://www.paragonrouting.com/

support tools that can be adopted in the transportation industry. Despite this, research in vehicle routing is often accused of being too idealistic. The majority of published research tackles simplified problems, based on, for example, Euclidean distances, homogeneous fleets, hard constraints, fixed service times... etc. Unfortunately, most of these assumptions do not hold in reality. Industrial aspects of vehicle routing have recently started to gain the attention of researchers, and are increasingly being incorporated into models and solution methods that address these problems. Thus, the current trend is towards 'holistic' approaches that are capable of solving richer and more realistic VRP models.

It is important from a realistic perspective to understand the relationship between theoretical research and commercial applicability in every day business requirements. To this end, we discuss in this chapter some industrial aspects that may be considered when addressing the VRP and its related variants in Section 11.1. Section 11.2 describes the basic components of commercial software that are adopted in the transportation industry, and explains how a theoretical research, like ours, may be integrated within its framework. Examples of commercial software products and applications in the transportation sector are provided in Section 11.3. Section 11.4 summarizes some future trends in vehicle routing research. Finally, Section 11.5 concludes this discussion with a summary and some brief remarks. Most of the information presented in this chapter is based on the research by SINTEF research organization in Norway[2], published in [72] and [76], unless otherwise indicated.

11.1 Industrial Aspects of Vehicle Routing

As mentioned above, research in vehicle routing is now shifting towards solving non-standard and rich VRP models that will facilitate decision making in real-life situations. The advancement in computational power in the last few decades has encouraged researchers to consider industrial aspects of vehicle routing, in order to meet the demands of transportation service providers and fleet management companies. Rich VRP models allow general and more realistic features to be incorporated, as opposed to conventional OR models which are simplistic in nature. Some industrial aspects of vehicle routing are summarized below:

- **Heterogeneous Fleet:** most of the VRP research makes the assumption that the operating fleet is homogeneous, with identical characteristics and operating costs. In reality, though, this is often not the case, since companies usually benefit from

[2] http://www.sintef.no/Home/

versatility in their fleet. A rich VRP model should allow non-homogeneous vehicles, such that not only the optimum number of vehicles is determined, but also the optimum number of each vehicle type and the optimum cost for vehicle acquisition/depreciation.

- **Drivers' Working Time:** in real life applications, drivers' working hours are governed by certain legislation rules. Hence, a rich VRP model should create working plans that conform to these regulations, such that the allocation and exchange of drivers is also taken into consideration while determining the optimum routing decision.

- **Depots and Service Locations:** basic VRP models usually assume that there is only one central depot, such that each vehicle's journey should start and end at that depot. Nevertheless, this assumption does not hold in many practical situations, since there might be multiple depots, or arbitrary starts and ends for vehicles. In addition, customer locations are sometimes not fixed, with alternative service locations being permitted in the routing plan.

- **Order Types:** in the basic VRP variants, order types are either pickups or deliveries. In addition, split deliveries are usually not allowed, and each customer can only be visited once. In real life applications, a customer order may be both a pickup and a delivery. Also, some orders may not require the transportation of goods, but only a certain service type (e.g. maintenance). Allowing such variants adds to the complexity of the problem but makes it more realistic from a business point of view.

- **Distances and Times:** the assumption that all distances under consideration are Euclidean distances is not adequate in real-life scenarios. Network characteristics, traffic, vehicle speed, and travel costs should also be taken into consideration. Moreover, service times for clients are not fixed in practical applications. Variable service times, depending on order types and volumes, should be allowed in rich VRP models.

- **Time Windows and Capacity Constraints:** most VRP models deal with the time window constraint as a *hard* constraint, with no violation permitted in the underlying routing plan. Time windows in reality are not always that rigid. They are often defined by preferred visiting times, with some cost penalty for visiting outside the specified period. Another extension to the basic model is multiple time windows, where different alternative visiting periods are given. Also, some applications require certain vehicle capacity and loading restrictions. For example, a

specific loading sequence may be enforced in order to facilitate un-loading, or to protect fragile items.

- **Uncertainty and Dynamic Situations:** dynamic vehicle routing refers to the situation where routing decisions are affected by input data arriving in real time. Information about orders, travel times, service times, vehicle breakdown... etc, arriving while the routing plan is being executed should be taken into consideration in most real-life applications, and necessitates an immediate response time. Uncertainty and stochastic variables, for example while planning a certain emergency situation, also add to the complexity of rich VRP models.

11.2 Commercial Transportation Software

A commercial software package for decision support in the transportation industry usually integrates an underlying algorithm with an efficient user friendly interface for optimum usage. A survey of commercial vehicle routing software [67] identifies the basic software capabilities as:

1. Geocoding addresses using a digital map database, i.e., determining the coordinates of a location using its address or postal code;
2. Determining the best driving route between pairs of geocoded pints;
3. Solving the VRP, i.e., assigning stops to vehicles and routing vehicles between stops, and
4. Displaying the results in both graphical and tabular forms, such that the dispatcher can communicate the solution to the drivers, and edit these solutions, with a 'drag-and-drop' feature if necessary.

Step 3 in the above list is where theoretical research, such that presented in this thesis, takes its part. It is in fact the 'core' or the 'engine' of the software tool, sometimes referred to as a**VRP solver**. Hasle and Kloster in [72] define a VRP solver as:

> A software component with functionality for modeling instances of targeted variants of the VRP and finding feasible, optimized solutions to a given instance. The effect of a given routing tool is highly dependent on its VRP solver.

The efficiency of a VRP solver can be generally attributed to the richness of the VRP model (problem) it is trying to solve, in addition to its algorithmic capabilities measured in terms of the quality of the objective and the processing time needed. To the best of our knowledge, most VRP solvers of today's commercial software do not adequately handle pickup and delivery problems. Improvements in this area are obviously in demand.

11.3 Examples of Commercial Vehicle Routing Tools

Some examples of commercial vehicle routing software are: ILOG Dispatcher, Paragon Routing and Scheduling System, Direct Route, DISC, and JOpt.SDK. Notable installations of these and other commercial software include companies like: Sainsbury's, Argos, Tesco, Royal Mail, the Home Depot, Samsung, Kraft, Dunkin Donuts, Coca-Cola, BP, TNT, Fujitsu, and many others [68].

One example of a rich generic VRP solver is SPIDER [72], which was developed by SINTEF Applied Mathematics research institution in Norway[3]. SPIDER VRP solver is capable of solving a number of vehicle routing problems and its variants, such as the VRPTW, the PDPTW and the multiple-depot variants of these problems. In addition, it takes into consideration many industrial aspects, such as the ones described above. For example, it allows a heterogeneous fleet, multiple time windows, alternative service locations, variable service times, and travel times that vary according to road network topology and information available from Geographic Information Systems (GIS).

The algorithmic approach in SPIDER is a *unified approach* to all problem types and instances. This technique has advantages in terms of simplifying the code and its maintenance, but may sometimes suffer in terms of computation time, since some operations will still be performed for problem instances that may not need them.

The SPIDER VRP solver is basically a meta-heuristic approach that integrates several features from successful academic research in the VRP field. The algorithm consists of three main components: *Construction of Initial Solutions*, *Tour Depletion* and *Iterative Improvement*. These are briefly explained below:

- **Construction of Initial Solutions:** the construction phase is based on extensions of classical construction heuristics like the Clark and Wright savings heuristic [26], Solomon's I1 insertion [141], and the regret-based insertion [121]. In addition, an

[3] `http://www.spidersolutions.no/`

instance analysis is performed in order to determine if the instance has a heterogeneous fleet, in which case a special construction heuristic called *SPIDER constructor* is used to build the initial solution.

- **Tour Depletion:** this phase is intended for reducing the number of routes in the initial solution, but is also used as a local search operator during the iterative improvement phase. Each route is depleted in turn, and an attempt is made to insert all its requests into other routes. If the attempt was successful, the new solution is accepted. The routes that have been changed by this operation may also be optimized using 2-Opt or 3-Opt improvement heuristics of [103].

- **Iterative Improvement:** the iterative improvement phase is based on Variable Neighbourhood Descend (VND) [70], using a number of well-known operators within each route and in-between routes. For example, 2-Opt, 3-Opt, Or-Opt, EXCHANGE, and CROSS (see Section 3.4 for more details about some of these operators). However, these heuristics have been extended to accommodate the SPIDER rich VRP model which allows heterogeneous fleets and multiple time windows. In addition, several neighbourhood filters have been applied to accelerate the optimization, for example by analyzing the current solution and exploring promising moves only. When the VND reaches a local minimum, a diversification mechanism is applied using Large Neighbourhood Search (LNS) [131]. The overall process is a hybrid of VND and Iterated Local Search (ILS).

To evaluate its performance, SPIDER was tested on published benchmark data available from the literature and compared favourably with state-of-the-art solution methods (see [72] for some experimental results of SPIDER on published benchmark instances of the PDPTW).

11.4 Future Trends

Scientific research in transportation optimization is an indispensable part of any commercial software tool, and there is, and will continue to be, a great demand for innovations in research methodologies. Researchers in this field, however, should also be aware of new demands in the industrial sector and try to develop richer models in their research. Issues like heterogeneous fleets, multiple tours, split deliveries, variable service times, soft time windows, special vehicle equipment or driver certificates, dynamic route planning, and many other real-life requirements and constraints should motivate researchers to invest more effort in developing efficient solution techniques that comply with today's

ever changing domain of industrial vehicle routing. The complexity of these models, nevertheless, means that solving the problem to optimality is not an option in most practical situations. In fact, heuristics, meta-heuristics and hybrid approaches dominate scientific research in this field.

There is a trend in today's research towards optimization tools that integrate the whole supply chain rather than individual components. In addition, current concerns over global warming has increased the demand towards rewarding lower carbon emissions and green logistics. More robust planning in dynamic and stochastic situations is also gaining more attention every day. Decision support tools that enable the service provider to choose between more than one good solution is also gaining more popularity in the current research environment.

Another important factor that needs urgent attention is the development of better benchmark test cases. Most of the benchmark cases available to the research community for the VRP and its related variants are created randomly. In addition, they are often overly simplified and do not reflect real-world cases. Using such test cases increases the risk of over fitting, i.e., a great effort may be invested in developing solution methods that produce good results on published benchmark instances only. Applying these methods on a larger scale, though, may reveal their shortcomings. The research in [76] identifies several features that test cases for the vehicle routing research should have. These are: 1) they should be based on real-world data, 2) they should be as rich as possible, i.e., contain sufficient details, 3) they should have a common format (e.g. XML), 4) solutions (and not just their objective functions) should be published, and 5) test cases and their solutions should be published in the Web.

11.5 Summary and Conclusions

This chapter reviewed some industrial aspects of vehicle routing that are currently under consideration by the research community to meet the increase in demand for more efficient transportation. The current trend is towards rich VRP models that can be used in commercial decision support tools, in order to achieve better customer service, cost reduction, and efficient resource management in transportation systems.

Yet, although scientific research is a major and important step towards a complete real-life applicable solution to routing and scheduling problems, there is often a considerable distance between theoretical research and practical applicability. Integrating theoretical scientific research within a commercially applicable tool is usually done by consulting

companies, routing software vendors, contract research organizations, and large research laboratories or institutions. In addition, it often requires the cooperation of a number of experts in different fields and possibly several years of development effort. These institutions, however, do not work in isolation from academic research. They monitor research carried out by the scientific community and incorporate state-of-the-art techniques in their products. Assessing the quality of the different scientific approaches is mostly done by comparing their performance against published test cases available for researchers in the academic field. Hence, the focus of researchers in Computer Science and Operations Research should be on developing competitive and robust solution methodologies that can be later integrated within a larger framework for applicability in real-life situations.

We have hereby completed a thorough explanation and analysis of the research carried out in this thesis, together with some necessary background information and a literature survey of state-of-the-art techniques in this field. The next chapter will conclude this thesis with a summary of the whole research and its major contributions, in addition to some future research directions.

Chapter 12

Conclusions and Future Directions

In this research we investigated some heuristic and meta-heuristic algorithms for solving selected pickup and delivery problems, namely the PDPTW and the 1-PDP. Innovations in solution techniques that handle these and similar vehicle routing problems are in continuous demand, since they can be used in decision support tools and may help reduce transport costs and optimize resource consumption.

Having explained in detail our research and its findings, in addition to reviewing important related work in this field, we present in this chapter a summary of the research and its main achievements in Section 12.1. Section 12.2 emphasizes parts of this research where further work may be carried out and summarizes some future research suggestions. Section 12.3 afterwards concludes this chapter and the whole thesis with some brief final remarks.

12.1 Research Summary and Contribution

Advances in computational power in the last few decades has contributed to the emergence of a trend among researchers towards powerful algorithms for solving optimization problems. A common phenomenon that existed as a result of this trend, though, is that solution algorithms have tended to became increasingly complex, often with many sophisticated and intertwined components. As a consequence, recently published results can be difficult to replicate, and some algorithms are indeed challenging to implement. Additionally, it may be difficult to assess the contribution of the different algorithmic components to the overall performance. In our view, it is good practice to provide a thorough analysis of all components of an optimization algorithm, to ensure that all are making a valuable contribution. Those that are not, should be removed and the algorithm simplified.

Another weakness that we perceive in much of the VRP literature is an overemphasis on beating best published results for benchmark data, at the expense of algorithm robustness. If solution algorithms are finely tuned for special benchmark instances, they may not work

at all well on unseen data or real world instances. Robustness is a key requirement, if an approach is to be eventually useful in practice.

The main philosophy in our research is the development of simple heuristic or meta-heuristic frameworks that can be easily understood and implemented. We concentrate here on some very challenging variants of vehicle routing problems involving pickup and delivery. The difficulty in dealing with pickup and delivery problems stems mainly from the existence of several problem constraints that must be dealt with during the construction and improvement phases of a solution process. Keeping this in mind, we directed our attention towards establishing effective ways to deal adequately with problem constraints. Two main aspects have been focused on to achieve this goal: the *solution representation* and the *neighbourhood moves*.

Our solution representation for the PDPTW tried to overcome the precedence constraint, between the pickup and the delivery, by assigning the same code to both locations, and always considering the first occurrence as the pickup. This simple technique removed the burden of having to check and correct the precedence infeasibility at each step of the solution process. This representation was used for both the single and multiple vehicle cases of the problem. In addition to handling the precedence issue, our representation also overcomes the coupling constraint that must be enforced when multiple vehicles are used, since both the pickup and its delivery should be served by the same vehicle. Hence, this simple approach further reduced the number of hard constraints that the solution algorithm has to deal with during the search.

In addition, neighbourhood moves played a central role in our research. For the PDPTW, a simple neighbourhood move that is guided by the time window proved successful on several occasions. It was used as a mutation operator and as a solution improvement tool within the different heuristic and meta-heuristics techniques applied. In addition, a neighbourhood move used for the first time within the VNS approach to the 1-PDP, namely the displacement and inversion move, has demonstrated its effectiveness in helping the search escape local optima. We also introduced some adaptive moves. For example, different bounds within the time window interval were used to direct the neighbourhood move at different stages during the search, when solving the PDPTW. Similarly, the increase in neighbourhood size within the VNS approach for the 1-PDP was controlled by the current search progress, such that a larger neighbourhood size is only attempted when such increase seems fruitful to the search process.

Our research methodology has made it possible for us to accomplish a number of achievements that we believe are significant to scientific research in this area. The main achievements of this study are:

1. We used our solution representation and neighbourhood moves to develop new simple routing heuristics for the SV-PDPTW. The results obtained in this part of the research were impressive, both in terms of the solution quality and processing time. One particular approach appears to have the best potential for the SV-PDPTW, the 3-stage SA routing heuristic. This approach was able to obtain results better than those previously published in all test cases and also performed significantly better than all other algorithms implemented in this part of our research. The experimental results and comparison with other heuristics indicate that the success of this approach was mostly due to the guided neighbourhood moves that were adopted to overcome the difficult time window constraint. Another heuristic which also showed potential, especially in terms of processing time, is the Hill Climbing (HC) routing heuristic. The HC algorithm also employed the same time window guided neighbourhood moves during the search. The routing heuristics developed for the SV-PDPTW can be easily integrated within real-world optimization tools that deal with this problem. Two publications were made out of the results obtained, one late breaking conference paper [79] and another journal paper [84].

2. Since there are no standard benchmark instances for the SV-PDPTW, we were able in this research to create test cases that can be used as benchmark data and used by researchers for testing their algorithms.

3. Based on the simple routing heuristics developed for the SV-PDPTW, we designed and compared several solution construction methods for the MV-PDPTW. These construction methods, especially the sequential construction approach (SEQ), can be used in any heuristic or meta-heuristic that deals with the PDPTW and in the related dial-a-ride problem as well. The construction heuristics are distinguished by their simplicity and ease in coding and application, compared to classical construction methods from the literature. This part of the research was published in [81].

4. We developed new problem-specific genetic operators and neighbourhood moves for the MV-PDPTW. These operators use techniques developed in the first parts of our research, i.e., for the SV-PDPTW and for the solution construction of the MV-PDPTW. Our operators are characterized by the ability to create feasible solutions throughout the search. The operators developed here can be easily adopted by other GA approaches for different vehicle routing problems. For example, a crossover that ranks routes to guide inheritance is applicable to any routing problem. These operators can also be adapted and employed within different heuristics and meta-heuristics for solving the problem, as done in this research by using similar operators in both the GA and the SA for solving the MV-PDPTW. The research

dealing with the GA approach in this part of the thesis was published in [80].

5. We developed an interesting adaptive hybrid VNS-SA approach for the 1-PDP. The algorithm is distinguished by adopting, for the first time, the traditional displacement and inversion move from previous vehicle routing research as a 'shaking' procedure within the VNS. In addition, adaptation is introduced in a novel way within the VNS meta-heuristic. Traditional VNS approaches usually increase the neighbourhood size up to a certain pre-defined limit. In our approach, we adapted this requirement, such that a further increase in the neighbourhood size is only applied when it seems beneficial from a search perspective. The algorithm introduced here proved its potential by beating previously best published results for 90% of the large problem instances solved. Two conference papers covering this part of the research have now been accepted for publication in *GECCO2010* conference (as a late breaking abstract) [82], and the *PPSN2010* conference [85]. In addition, a third journal paper has been submitted to the *Journal of Heuristics* and is currently under review [83].

The above achievements indicate that the techniques we developed to guide our heuristic and meta-heuristic approaches were successful to a large extent in accomplishing the objectives of the research. Nevertheless, we can also identify a few parts of the research that did not meet the anticipated standards. Three main areas have shown some shortcomings and need further investigation. These are:

1. The improvement heuristics of the MV-PDPTW (explained in Chapter 8), since the results obtained by both the GA and the SA algorithms were in general of lesser quality than the best known results.

2. The evolutionary perturbation heuristic of the 1-PDP (explained in Chapter 9). Despite its success on other VRPs, this technique did not obtain high quality solutions for the 1-PDP. However, the approach did show promise in removing infeasibility.

3. The processing time needed by the VNS approach to the 1-PDP (explained in Chapter 10), since the final solution obtained, albeit having a very good quality, needed quite a long processing time.

The possible reasons behind these shortcomings have been addressed in their respective chapters. Nevertheless, we will further elaborate on some future research directions that may be pursued to remedy these shortcomings. To this end, Section 12.2 includes a brief critical analysis of some parts of the current research and suggestions of additional work that may be performed to complement the research carried out in this thesis.

12.2 Critical Analysis and Future Work

Some aspects of this thesis where further work can be done are summarized below.

12.2.1 The SV-PDPTW

To complement the research done so far for this problem, more problem instances of different sizes and different characteristics need to be created. For example, similar to the benchmark instances of the VRPTW and the MV-PDPTW, a certain distribution of nodes may be enforced, such as having some instances with clustered or partially clustered locations. Also, problem instances may differ according to the width of the time window, by having short or long schedule horizons.

In addition, a more thorough investigation of the approaches developed in this part of the research may be undertaken by testing the algorithms on a larger number of test cases and performing an in-depth statistical analysis of the results.

12.2.2 Solution Construction for the MV-PDPTW

Further investigation of the solution construction heuristics developed in this part of the research may be carried out by implementing one or more 'traditional' construction methods, such as Solomon's I1 insertion heuristic [141], after adapting it for the PDPTW. This should then be followed by comparing the performance of the traditional heuristic(s) with the new construction algorithms developed in this research, on benchmark instances.

12.2.3 Solution Improvement for the MV-PDPTW

Our selection of GAs as a candidate solution improvement method for the MV-PDPTW was based on the general attractive features that GAs possess, such as simplicity and robustness. Moreover, we were also encouraged by the success of GAs in solving closely related vehicle routing problems, like the VRPTW. As previously discussed during the course of this thesis, though, GAs usually suffer when dealing the MV-PDPTW. This is mostly due to the number of underlying constraints that must be dealt with when designing problem-specific genetic operators. Another important factor is the difficulty in handling the grouping and the routing aspects of the problem *simultaneously*, in the solution encoding as well as the genetic operators. Similar difficulties also seem to apply to

our SA approach, which was attempted for solving this problem, since it employs similar operators to those adopted by our GA approach.

Nevertheless, we may still be able to improve the results obtained by both the GA and SA in the improvement phase of the MV-PDPTW by augmenting the solution approach with some local search method. For example, a 2-Opt or a 3-Opt heuristic may be used to improve individual routes at various stages during the search.

In retrospect, given the opportunity to repeat the project, a new meta-heuristic technique would have probably been chosen for the improvement phase of the MV-PDPTW. For example, VNS would be one option that would definitely be considered. In fact, it was notable that a significant number of papers presented on VRPs at the recent *MIC2009*[1] conference, used VNS, most of them achieving impressive results. This was indeed one of the reasons that encouraged us to apply VNS to the 1-PDP, after the 'disappointing' results of the EPS heuristic. Applying VNS to the PDPTW would also seem an attractive prospect, since many neighbourhood moves that have been previously used for the VRPTW and the PDPTW can easily be incorporated into the VNS framework. In addition, to the best of our knowledge, VNS has not been previously tried on the PDPTW. It would be interesting to test the routing and construction heuristics developed in this research within a VNS meta-heuristic for solving the MV-PDPTW.

12.2.4 The Evolutionary Perturbation Heuristic for the 1-PDP

As previously mentioned in Chapter 9, the EPS heuristic was not very successful in achieving the anticipated results. Both the quality of the results obtained and the processing time were inferior to previous heuristics in this field. The main suggestion in terms of the solution quality would be to modify the construction algorithm or experiment with a new construction technique. As previously discussed in Section 9.5, one possibility for improvement is to modify the construction algorithm such that rather than halting the insertion process when infeasibility is encountered, the algorithm would continue to insert nodes despite infeasibility. If this is done, correcting infeasibility would probably be easier for the perturbation heuristic, and the overall quality of generated solutions would improve. Moreover, this technique has demonstrated its potential within the AVNS-SA approach to the problem as explained in Chapter 10.

In terms of processing time, a possible option for improvement is to optimize the perturbed coordinates using a faster heuristic or meta-heuristic. For example, a simulated

[1] VIII Metaheuristic International conference, Hamburg, Germany - July 13-16, 2009
`http://www.smartframe.de/mic09/Home.html`

annealing or a hill climbing algorithm may be used instead of a GA to improve the perturbed problem coordinates. Together, with an appropriate and fast construction heuristic the overall scheme may be considerably enhanced.

12.2.5 The AVNS-SA Approach to the 1-PDP

The AVNS-SA approach was in fact very successful in dealing with the 1-PDP, especially for large problem sizes. As previously mentioned, the algorithm was able to obtain new best results for 90% of the large problem instances. The only drawback of the algorithm seems to be the long processing time. As previously discussed in Chapter 10, an improvement in this area can be achieved if we direct our attention to the local search component of the algorithm. The 2-Opt heuristic used in the local search phase is the most time consuming part of the algorithm. This heuristic adopts a best improvement strategy and is invoked to optimize every new generated solution. Several suggestions may help in this respect. One option is to reduce the number of calls to the 2-Opt algorithm based on some solution quality. For example, the 2-Opt procedure can be restricted only to solutions that look promising from a search perspective (depending on the objective function value or the number of constraint violations, for instance). In addition, the 2-Opt algorithm may be applied to selected edges of the solution and not to the whole solution. Some faster implementations of the 2-Opt algorithm may also be tried to reduce the time requirement of this component (e.g. [13]).

Finally, the stopping condition of the overall AVNS-SA algorithm also needs further investigation. To improve the processing time of the algorithm, it is necessary to find an appropriate criterion to stop repeating the AVNS-SA procedure, without sacrificing the quality of the final solution returned. However, obtaining a balance between solution quality and processing time may not be very straightforward for such hard problem.

12.3 Final Remarks

To sum up, we believe that the work done in this thesis contributes positively to scientific research on vehicle routing and scheduling. This study provided simple ideas on constraint handling mechanisms that can be used in designing effective and robust heuristic and meta-heuristic algorithms. The techniques developed in this research can be easily integrated within a larger framework and used in optimization tools to help improve transportation and logistic planing.

In this final chapter of the thesis we highlighted the most significant findings and the most promising areas of our research. We also tried to identify some research areas where further investigation and some elaboration are still needed. We aspire to carry out in the near future some of the outlined suggestions to complement the work done in this thesis and advance the research to its uttermost standards. Additional publications of the findings of this thesis are also underway.

Bibliography

[1] European transport policy for 2010: Time to decide. European Commission White Paper ISBN 92-894-0341-1, Office for Official Publications of the European Communities, Luxembourg, 2001. http://ec.europa.eu/transport/white_paper/documents/index_en.htm.

[2] Delivering a sustainable transport system: The logistics perspective. Technical report, Freight and Logistics Division, Department for Transport, U.K., December 2008. http://www.dft.gov.uk/pgr/freight/dastslogistics/.

[3] E. Aarts and J. K. Lenstra, editors. *Local Search in Combinatorial Optimization*. John Wiley & Sons, Inc., 1997.

[4] A. Alshamrani, K. Mathur, and R. H. Ballou. Reverse logistics: Simultaneous design of delivery routes and returns strategies. *Computers & Operations Research*, 34(2):595–619, 2007.

[5] S. Anily and J. Bramel. Approximation algorithms for the capacitated traveling salesman problem with pickups and deliveries. *Naval Research Logistics*, 46(6):654–670, 1999.

[6] C. Avanthay, A. Hertz, and N. Zufferey. A variable neighbourhood search for graph coloring. *European Journal of Operational Research*, 151(2):379–388, 2003.

[7] R. Baldacci, E. Hadjiconstantinou, and A. Mingozzi. An exact algorithm for the traveling salesman problem with deliveries and collections. *Networks*, 42(1):26–41, 2003.

[8] R. Battiti and G. Tecchiolli. The reactive tabu search. *ORSA Journal on Computing*, 6(2):126–140, 1994.

[9] J. Baugh, G. Kakivaya, and J. Stone. Intractability of the dial-a-ride problem and a multiobjective solution using simulated annealing. *Engineering Optimization*, 30(2):91–123, 1998.

[10] E. B. Baum. Iterated descent: A better algorithm for local search in combinatorial optimization problems. In D. Touretzky, editor, *Proc. Neural Information Processing Systems*, 1988.

[11] R. Bent and P. Van Hentenryck. A two-stage hybrid local search for the vehicle routing problem with time windows. *Transportation Science*, 38(4):515–530, 2004.

[12] R. Bent and P. Van Hentenryck. A two-stage hybrid algorithm for pickup and delivery vehicle routing problems with time windows. *Computers & Operations Research*, 33(4):875–893, 2006.

[13] J. J. Bentley. Fast algorithms for geometric traveling salesman problems. *INFORMS Journal on Computing*, 4(4):387–411, 1992.

[14] G. Berbeglia, J.-F. Cordeau, I. Gribkovskaia, and G. Laporte. Static pickup and delivery problems: A classification scheme and survey. *TOP*, 15(1):1–31, 2007.

[15] J. Berger, M. Barkaoui, and O. Bräysy. A route-directed hybrid genetic approach for the vehicle routing problem with time windows. *INFOR*, 41:179–194, 2003.

[16] J. Blanton and R. Wainwright. Multiple vehicle routing with time and capacity constraints using genetic algorithms. In *Proceedings of the 5th International Conference on Genetic Algorithms*, pages 452–459. Morgan Kaufmann Publishers, Inc., 1993.

[17] R. Bradwell, L. P. Williams, and C. L. Valenzuela. Breeding perturbed city coordinates and fooling travelling salesman heuristic algorithms. In *International Conference on Artificial Neural Networks and Genetic Algorithms (ICANNGA97)*, 1997.

[18] O. Bräysy. Fast local searches for the vehicle routing problem with time windows. *INFOR*, 40:319–330, 2002.

[19] O. Bräysy. A reactive variable neighbourhood search for the vehicle-routing problem with time windows. *INFORMS Journal On Computing*, 15(4):347–368, 2003.

[20] O. Bräysy and M. Gendreau. Vehicle routing problem with time windows, part I: Route construction and local search algorithms. *Transportation Science*, 39(1):104–118, 2005.

[21] O. Bräysy and M. Gendreau. Vehicle routing problem with time windows, part II: Metahueristics. *Transportation Science*, 39(1):119–139, 2005.

[22] O. Bräysy, P. P. Porkka, W. Dullaert, P. P. Repoussis, and C. D. Tarantilis. A well-scalable metaheuristic for the fleet size and mix vehicle routing problem with time windows. *Expert Systems with Applications*, 36(4):8460–8475, 2009.

[23] L. Bruggen, J. Lenstra, and P. Schuur. Variable-depth search for the single-vehicle pickup and delivery problem with time windows. *Transportation Science*, 27(3):298–311, 1993.

[24] A. M. Campbell and M. Savelsbergh. Efficient insertion heuristics for vehicle routing and scheduling problems. *Transportation Science*, 38(3):369–378, 2004.

[25] W.-C. Chiang and R. A. Russell. Simulated annealing metaheuristics for the vehicle routing problem with time windows. *Annals of Operations Research*, 63(1):3–27, 1996.

[26] G. Clarke and J. W. Wright. Scheduling of vehicles from a central depot to a number of delivery points. *Operations Research*, 12(4):568–581, 1964.

[27] B. Codenotti, G. Manzini, L. Margara, and G. Resta. Perturbation: An efficient technique for the solution of very large instances of the Euclidean TSP. *INFORMS Journal On Computing*, 8(2):125–133, 1996.

[28] J.-F. Cordeau, G. Desaulnier, J. Desrosiers, M. M. Solomon, and F. Soumis. *The Vehicle Routing Problem*, chapter VRP with Time Windows, pages 157–193. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2001.

[29] J.-F. Cordeau and G. Laporte. The dial-a-ride problem (DARP): Variants, modeling issues and algorithms. *4OR: A Quarterly Journal of Operations Research*, 1(2):89–101, 2003.

[30] J.-F. Cordeau and G. Laporte. A tabu search heuristic for the static multi-vehicle dial-a-ride problem. *Transportation Research Part B: Methodological*, 37(6):579–594, July 2003.

[31] J.-F. Cordeau, G. Laporte, and S. Ropke. *The Vehicle Routing Problem: Latest Advances and New Challenges*, volume 43, chapter Recent Models and Algorithms for One-to-One Pickup and Delivery Problems, pages 327–357. Springer US, 2008.

[32] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. MIT Press, second edition, 2001.

[33] J. Créput, A. Koukam, J. Kozlak, and J. Lukasik. *Computational Science- ICCS 2004*, volume 3038/2004, chapter An Evolutionary Approach to Pickup and Delivery Problem with Time Windows, pages 1102–1108. Springer, 2004.

[34] C. Cubillos, N. Rodriguez, and B. Crawford. *Bio-inspired Modeling of Cognitive Tasks*, chapter A Study on Genetic Algorithms for the DARP Problem, pages 498–507. 2007.

[35] G. B. Dantzig and D. R. Fulkerson. Minimizing the number of tankers to meet a fixed schedule. *Naval Research Logistics Quarterly*, 1:217–222, 1954.

[36] M. Dell'Amico and F. Maffioli. *Meta-Heuristics: Theory & Applications*, chapter A New Tabu Search Approach to the 0-1 Equicut Problem, pages 361–378. Kluwer Academic Publishers, 1996.

[37] U. Derigs and T. Döhmer. Indirect search for the vehicle routing problem with pickup and delivery and time windows. *OR Spectrum*, 30(1):149–165, 2008.

[38] J. Desrosiers, Y. Dumas, and F. Soumis. A dynamic programming solution of the large-scale single vehicle dial-a-ride problem with time windows. *American Journal of Mathematical and Management Sciences*, 6(3-4):301–325, 1986.

[39] K. Doerner, R. F. Hartl, and M. Reimann. *Operations Research Proceedings 2000*, chapter Ants Solve Pickup and Delivery Problems with Full Truck Loads, pages 395–400. Springer, Berlin, 2001.

[40] J. Dorband, C. L. Mumford, and P. Wang. Developing an ace solution for two-dimensional strip packing. In *18th International Parallel and Distributed Processing Symposium Workshop on Massively Parallel Processing*, 2004. Santa Fe, New Mexico.

[41] M. Dorigo, G. Di Caro, and L. M. Gambardella. Ant algorithms for discrete optimization. *Artificial Life*, 5(2):137–172, 1999.

[42] M. Dorigo, V. Maniezzo, and A. Colorni. The ant system: Ant autocatalytic optimizing process. Technical Report TR91-016, Politenico di Milano, 1991.

[43] M. Dorigo, V. Maniezzo, and A. Colorni. Optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man, and Cybernetics - Part B*, 26(1):29–41, 1996.

[44] M. Dorigo and T. Stützle. *Ant Colony Optimization*. MIT Press, Cambridge, MA, 2004.

[45] M. Dror, D. Fortin, and C. Roucairol. Redistribution of self-service electric cars: (a case of pickup and delivery). Technical Report 3543, INRIA, Unité de recherche de Rocquencourt, Rocquencourt, FRANCE, 1998.

[46] Y. Dumas, J. Desrosiers, and F. Soumis. The pickup and delivery problem with time windows. *European Journal of Operational Research*, 54:7–22, 1991.

[47] R. W. Eglese. Simulated annealing: A tool for operational research. *European Journal of Operational Research*, 46(3):271–281, 1990.

[48] B. Eksioglu, A. V. Vural, and A. Reisman. The vehicle routing problem: A taxonomic review. *Computers & Industrial Engineering*, 57(4):1472–1483, 2009.

[49] E. Falkenauer. *Genetic Algorithms and Grouping Problems*. John Wiley and Sons, 1998.

[50] A. Fraser. Simulation of genetic systems by automatic digital computers. I. introduction. *Australian Journal of Biological Science*, 10:484–491, 1957.

[51] A. Fraser. Simulation of genetic systems by automatic digital computers. II. effects of linkage on rates of advanced under-selection. *Australian Journal of Biological Science*, 10:492–499, 1957.

[52] L. M. Gambardella, E. Taillard, and G. Agazzi. *New Ideas in Optimization*, chapter MACS-VRPTW: A Multiple Colony System For Vehicle Routing Problems With Time Windows, pages 63–76. McGraw-Hill, 1999.

[53] B.-L. Garcia, J.-Y. Potvin, and J.-M. Rousseau. A parallel implementation of the tabu search heuristic for vehicle routing problems with time window constraints. *Computers & Operations Research*, 21(9):1025–1033, 1994.

[54] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1979.

[55] M. Gendreau. *Handbook of Metaheuristics*, volume 57, chapter An Introduction to Tabu Search, pages 37–54. Springer, NY, 2003.

[56] M. Gendreau, A. Hertz, and G. Laporte. New insertion and postoptimization procedures for the traveling salesman problem. *Operations Research*, 40(6):1086–1094, 1992.

[57] M. Gendreau, G. Laporte, and A. Hertz. An approximation algorithm for the traveling salesman problem with backhauls. *Operations Research*, 45(4):639–641, Jul. - Aug. 1997.

[58] B. E. Gillett and L. R. Miller. A heuristic algorithm for the vehicle-dispatch problem. *Operations Research*, 22(2):340–349, 1974.

[59] F. Glover. Heuristics for integer programming using surrogate constraints. *Decision Sciences*, 8(1):156–166, 1977.

[60] F. Glover. Multilevel tabu search and embedded search neighbourhoods for the traveling salesman problem. Working Paper, College of Business & Administration, University of Colorado, Boulder, 1991.

[61] F. Glover. *Computer Science and Operations Research: New Developments in Their Interfaces*, chapter New Ejection Chain and Alternating Path Methods for Traveling Salesman Problems, pages 449–509. Pergamon Press, Oxford, 1992.

[62] F. Glover. Tabu search and adaptive memory programming: Advances, applications and challenges. In *Interfaces in Computer Science and Operations Research*, pages 1–75. Kluwer, 1996.

[63] F. Glover and H. J. Greenberg. New approaches for heuristic search: A bilateral linkage with artificial intelligence. *European Journal of Operational Research*, 39(2):119–130, 1989.

[64] D. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1989.

[65] B. L. Golden and W. Stewart. *The Traveling Salesman Problem. A Guided Tour of Combinatorial Optimization*, chapter Empirical Analysis of Heuristics, pages 207–249. John Wiley and Sons, 1985.

[66] M. Guan. Graphic programming using odd or even points. *Chinese Mathematics*, 1:237–277, 1962.

[67] R. Hall. On the road to integration - 2006 survey of vehicle routing software spotlights critical supply chain management role. *ORMS TODAY*, June 2006. `http://www.lionhrtpub.com/orms/orms-6-06/frvehiclerouting.html`.

[68] R. Hall and J. Partyka. On the road to mobility - 2008 survey of vehicle routing software spotlights critical supply chain management role. *ORMS TODAY*, February 2008. `http://www.lionhrtpub.com/orms/surveys/Vehicle_Routing/vrss.html`.

[69] P. Hansen and N. Mladenović. An introduction to variable neighbourhood search. In S. Voss, S. Martello, I. H. Osman, and C. Roucairol, editors, *Meta-heuristics,*

Advances and trends in local search paradigms for optimization, pages 433–458. Kluwer Academic Publishers, 1998.

[70] P. Hansen and N. Mladenović. Variable neighbourhood search: Principles and applications. *European Journal of Operational Research*, 130(3):449–467, 2001.

[71] P. Hansen, N. Mladenović, and J. A. Moreno Pérez. Variable neighbourhood search: Methods and applications. *4OR: A Quarterly Journal of Operations Research*, 6(4):319–360, 2008.

[72] G. Hasle and O. Kloster. *Geometric Modeling, Numerical Simulation, and Optimization: Applied Mathematics at SINTEF*, chapter Industrial Vehicle Routing, pages 397–435. Springer, 2007.

[73] H. Hernández-Pérez, I. Rodríguez-Martín, and J.-J. Salazar-González. A hybrid GRASP/VND heuristic for the one-commodity pickup-and-delivery traveling salesman problem. *Computers & Operations Research*, 36(5):1639–1645, 2008.

[74] H. Hernández-Pérez and J.-J. Salazar-González. A branch-and-cut algorithm for a traveling salesman problem with pickup and delivery. *Discrete Applied Mathematics*, 145(1):126–139, 2004.

[75] H. Hernández-Pérez and J.-J. Salazar-González. Heuristics for the one-commodity pickup-and-delivery traveling salesman problem. *Transportation Science*, 38(2):245–255, 2004.

[76] A. Hoff, H. Andersson, M. Christiansen, G. Hasle, and A. Løkketangen. Industrial aspects and literature survey: Fleet composition and routing. Technical Report SINTEF A7029, SINTEF, 2008.

[77] A. Hoff and A. Løkketangen. Creating lasso-solutions for the traveling salesman problem with pickup and delivery by tabu search. *Central European Journal of Operations Research*, 14(2):125–140, 2006.

[78] J. H. Holland. *Adaptation in Natural and Artificial Systems*. MIT Press, 1975.

[79] M. I. Hosny and C. L. Mumford. Single vehicle pickup and delivery with time windows: Made to measure genetic encoding and operators. In *Proceedings of the 2007 GECCO Conference Companion on Genetic and Evolutionary Computation, SESSION: Late-breaking papers*, pages 2489–2496, 2007.

[80] M. I. Hosny and C. L. Mumford. Investigating genetic algorithms for solving the multiple vehicle pickup and delivery problem with time windows. In *MIC2009, Metaheuristic International Conference*, July 2009.

[81] M. I. Hosny and C. L. Mumford. New solution construction heuristics for the multiple vehicle pickup and delivery problem with time windows. In *MIC2009, Metaheuristic International Conference*, July 2009.

[82] M. I. Hosny and C. L. Mumford. An adaptive hybrid VNS/SA approach to the one-commodity pickup and delivery problem. In *Proceedings of the 2010 GECCO Conference Companion on Genetic and Evolutionary Computation, SESSION: Late-breaking abstracts (to appear)*, July 2010.

[83] M. I. Hosny and C. L. Mumford. A self adaptive hybridization of VNS and SA for solving the single-commodity pickup and delivery problem. *Journal of Heuristics, Special Issue on Intelligent Metaheuristics for Logistic (under review)*, 2010.

[84] M. I. Hosny and C. L. Mumford. The single vehicle pickup and delivery problem with time windows: Intelligent operators for heuristic and metaheuristic algorithms. *Journal of Heuristics, Special Issue on Advances in Metaheuristics*, 16(3):417–439, June 2010.

[85] M. I. Hosny and C. L. Mumford. Solving the one-commodity pickup and delivery problem using an adaptive hybrid VNS/SA approach. In *Proceedings of the 11th International Conference on Parallel Problem Solving From Nature (PPSN2010), LNCS, Springer, Verlag (to appear)*, September 2010.

[86] B. Hunsaker and M. Savelsbergh. Efficient feasibility testing for dial-a-ride problems. *Operations Research Letters*, 30(3):169–173, 2002.

[87] J.-J. Jaw, A. Odoni, H. Psaraftis, and N. Wilson. A heuristic algorithm for the multi-vehicle advance request dial-a-ride problem with time windows. *Transportation Research Part B: Methodological*, 20(3):243–257, June 1986.

[88] W. Jih and Y. Hsu. Dynamic vehicle routing using hybrid genetic algorithms. In *Proceedings of the 1999 IEEE International Conference on Robotics and Automation*, volume 1, pages 453–458, 1999. Detroit, Michigan.

[89] W. Jih and Y. Hsu. A family competition genetic algorithm for the pickup and delivery problems with time window. *Bulletin of the College of Engineering*, 90:89–98, February 2004.

[90] J. de Jong and M. A. Wiering. Multiple ant colony systems for the busstop allocation problem. In *Proceedings of the Thirteenth Belgium-Netherlands Conference on Artificial Intelligence*, pages 141–148, 2001.

[91] R. Jørgensen, J. Larsen, and K. Bergvinsdottir. Solving the dial-a-ride problem using genetic algorithms. *The Journal of the Operational Research Society*, 58:1321–1331, 2007.

[92] S. Jung and A. Haghani. Genetic algorithm for a pickup and delivery problem with time windows. *Transportation Research Record*, 1733(1):1–7, 2000.

[93] S. Kirkpatrick, C. Gelatt, and M. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[94] N. Krasnogor and J. Smith. A memetic algorithm with self-adaptive local search: TSP as a case study. In *Genetic and Evolutionary Computation Conference*, pages 987–994. Morgan Kaufmann, 2000.

[95] A. Landrieu, Y. Mati, and Z. Binder. A tabu search heuristic for the single vehicle pickup and delivery problem with time windows. *Journal of Intelligent Manufacturing*, 12(5):497–508, 2001.

[96] G. Laporte. The vehicle routing problem: An overview of exact and approximate algorithms. *European Journal of Operational Research*, 59(3):345–358, 1992.

[97] G. Laporte. Fifty years of vehicle routing. *Transportation Science*, 43(4):408–416, 2009.

[98] H. Lau and Z. Liang. Pickup and delivery with time windows: Algorithms and test case generation. In *Proceedings of the 13th International Conference on Tools with Artificial Intelligence*, pages 333–340, November 2001.

[99] H. C. Lau, M. Sim, and K. M. Teo. Vehicle routing problem with time windows and a limited number of vehicles. *European Journal of Operational Research*, 148(3):559–569, 2003.

[100] H. Li and A. Lim. A metaheuristic for the pickup and delivery problem with time windows. In *Proceedings of the 13th IEEE International Conference on Tools with Artificial Intelligence*, pages 160–167, November 2001. Dallas, TX, USA.

[101] H. Li and A. Lim. Local search with annealing-like restarts to solve the VRPTW. *European Journal of Operational Research*, 150(1):115–127, 2003.

[102] H. Lim, A. Lim, and B. Rodrigues. Solving the pickup and delivery problem with time windows using iSSQUEAKY WHEELi optimization with local search. In *AMCIS 2002 Proceedings*, 2002.

[103] S. Lin. Computer solutions of the traveling salesman problem. *Bell Systems Technical Journals*, 44:2245–2269, 1965.

[104] Q. Lu and M. Dessouky. A new insertion-based construction heuristic for solving the pickup and delivery problem with time windows. *European Journal of Operational Research*, 175(2):672–687, December 2006.

[105] K. Man, K. Tang, and S. Kwong. Genetic algorithms: Concepts and applications [in engineering design]. *IEEE Transactions on Industrial Electronics*, 43(5):519–534, October 1996.

[106] G. Martinović, I. Aleksi, and A. Baumgartner. Single-commodity vehicle routing problem with pickup and delivery service. *Mathematical Problems in Engineering*, 2008:1–17, 2008.

[107] N. Metropolis, A. Rosenbluth, M. Rosenbluth, A. Teller, and E. Teller. Equation of state calculations by fast computing machines. *Journal of Chemical Physics*, 21(6):1087–1092, 1953.

[108] N. Mladenović and P. Hansen. Variable neighbourhood search. *Computers & Operations Research*, 24(11):1097–1100, 1997.

[109] M. W. Morgan. *GAPS: A hybridised framework applied to vehicle routing problems*. PhD thesis, Cardiff University - School of Computer Science & Informatics, December 2008.

[110] G. Mosheiov. The travelling salesman problem with pick-up and delivery. *European Journal of Operational Research*, 79(2):299–310, December 1994.

[111] G. Nagy and S. Salhi. Heuristic algorithms for single and multiple depot vehicle routing problems with pickups and deliveries. *European Journal of Operational Research*, 162(1):126–141, 2005.

[112] W. Nanry and J. Barnes. Solving the pickup and delivery problem with time windows using reactive tabu search. *Transportation Research Part B: Methodological*, 34(2):107–121, February 2000.

[113] E. Nowicki and C. Smutnicki. The flow shop with parallel machine. a tabu search approach. Technical Report ICT PRE 30/95, Technical University of Wroclaw, 1995.

[114] I. Or. *Traveling Salesman-Type Combinatorial Problems and their Relation to the Logistics of Regional Blood Banking*. PhD thesis, Northwestern University, Evanston, Illinois, 1976.

[115] I. H. Osman. Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problems. *Annals of Operations Research*, 41(4):421–452, 1993.

[116] G. Pankratz. A grouping genetic algorithm for the pickup and delivery problem with time windows. *OR Spectrum*, 27:21–24, 2005.

[117] S. Parragh, K. Doerner, and R. Hartl. A survey on pickup and delivery problems. *Journal für Betriebswirtschaft*, 58(2):81–117, 2008.

[118] S. Pirkwieser and G. R. Raidl. Multiple variable neighbourhood search enriched with ILP techniques for the periodic vehicle routing problem with time windows. *Lecture Notes In Computer Science*, 5818:45–59, 2009.

[119] M. Placek, R. F. Hartl, and K. Doerner. A variable neighbourhood search for the multi depot vehicle routing problem with time windows. *Journal of Heuristics*, 10:613–627, 2004.

[120] J.-Y. Potvin and S. Bengio. The vehicle routing problem with time windows Part II: Genetic search. *INFORMS Journal On Computing*, 8(2):165–172, 1996.

[121] J.-Y. Potvin and J.-M. Rousseau. A parallel route building algorithm for the vehicle routing and scheduling problem with time windows. *European Journal of Operational Research*, 66(3):331–340, May 1993.

[122] J.-Y. Potvin and J.-M. Rousseau. An exchange heuristic for routing problems with time windows. *Journal of the Operational Research Society*, 46(12):1433–1446, 1995.

[123] P. Prosser and P. Shaw. Study of greedy search with multiple improvement heuristics for vehicle routing problems. Working Paper, University of Strathclyde, Glasgow, Scotland, 1996.

[124] H. N. Psaraftis. Analysis of an $O(n^2)$ heuristic for the single vehicle many-to-many Euclidean dial-a-ride problem. *Transportation Research Part B: Methodological*, 17(2):133–145, April 1983.

[125] C. R. Reeves, editor. *Modern heuristic techniques for combinatorial problems*. John Wiley & Sons, Inc., 1993.

[126] B. Rekiek, A. Delchambre, and H. A. Saleh. Handicapped person transportation: An application of the grouping genetic algorithm. *Engineering Applications of Artificial Intelligence*, 19:511–520, 2006.

[127] J. Renaud, F. Boctor, and J. Quenniche. A heuristic for the pickup and delivery traveling salesman problem. *Computers & Operations Research*, 27(9):905–916, August 2000.

[128] P. P. Repoussis, C. D. Tarantilis, and G. Ioannou. Arc-guided evolutionary algorithm for the vehicle routing problem with time windows. *IEEE Transactions On Evolutionary Computation*, 13(3):624–647, 2009.

[129] Y. Rochat and E. Taillard. Probabilistic diversification and intensification in local search for vehicle routing. *Journal of Heuristics*, 1(1):147–167, 1995.

[130] S. Ropke, J.-F. Cordeau, and G. Laporte. Models and branch-and-cut algorithms for pickup and delivery problems with time windows. *Networks*, 49(4):258–272, 2007.

[131] S. Ropke and D. Pisinger. An adaptive large neighbourhood search heuristic for the pickup and delivery problem with time windows. *Transportation Science*, 40(4):455–472, November 2006.

[132] S. Ropke and D. Pisinger. A unified heuristic for a large class of vehicle routing problems with backhauls. *European Journal of Operational Research*, 171(3):750–775, 2006.

[133] R. A. Russell. Hybrid heuristics for the vehicle routing problem with time windows. *Transportation Science*, 29(2):156–166, 1995.

[134] S. Salhi and G. Nagy. A cluster insertion heuristic for single and multiple depot vehicle routing problems with backhauling. *The Journal of the Operational Research Society*, 50(10):1034–1042, 1999.

[135] SAM::OPT SINTEF Applied Mathematics - Department of Optimisation. Technical report in progress- not yet published. `http://www.sintef.no/Projectweb/TOP/Problems/PDPTW/Li--Lim-benchmark/`.

[136] M. Savelsbergh. The vehicle routing problem with time windows: Minimizing route duration. *INFORMS Journal On Computing*, 4(2):146–154, 1992.

[137] M. Savelsbergh and M. Sol. The general pickup and delivery problem. *Transportation Science*, 29(1):17–29, 1995.

[138] G. Schrimpf, J. Schneider, and H. Stamm-Wilbrandt G. Dueck. Record breaking optimization results using the ruin and recreate principle. *Journal of Computational Physics*, 159(2):139–171, 2000.

[139] P. Shaw. Using constraint programming and local search methods to solve vehicle routing problems. In *Proceedings of the 4th International Conference on Principles and Practice of Constraint Programming*, pages 417–431, London, UK, 1998. Springer-Verlag.

[140] J. Skorin-Kapov. Extensions of a tabu search adaptation to the quadratic assignment problem. *Computers & Operations Research*, 21(8):855–865, 1994.

[141] M. M. Solomon. Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations Research*, 35(2):254–265, 1987.

[142] M. M. Solomon and J. Desrosiers. Survey paper–time window constrained routing and scheduling problems. *Transportation Science*, 22(1):1–13, 1988.

[143] E. Taillard, P. Badeau, M. Gendreau, and F. Guertin J.-Y. Potvin. A tabu search heuristic for the vehicle routing problem with soft time windows. *Transportation Science*, 31(2):170–186, 1997.

[144] S. R. Thangiah. *Application handbook of genetic algorithms: New frontiers*, volume II, chapter Vehicle Routing with Time Windows Using Genetic Algorithms, pages 253–277. CRC Press, 1995.

[145] S. R. Thangiah, J.-Y. Potvin, and T. Sun. Heuristic approaches to vehicle routing with backhauls and time windows. *Computers & Operations Research*, 23(11):1043–1057, 1996.

[146] P. M. Thompson and H. N. Psaraftis. Cyclic transfer algorithms for multivehicle routing and scheduling problems. *Operations Research*, 41(5):935–946, 1993.

[147] P. Toth and D. Vigo, editors. *The vehicle routing problem*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2001.

[148] C. A. Tovey. Simulated simulated annealing. *American Journal of Mathematical and Management Sciences*, 8(3-4):389–407, 1988.

[149] K.-P. Urban. A guided simulated annealing search for solving the pick-up and delivery problem with time windows and capacity constraints. *International Journal of Logistics: Research and Applications*, 9(4):369–381, December 2006.

[150] C. L. Valenzuela. Evolutionary divide and conquer (II): for the TSP. In *Genetic and Evolutionary Computation Conference*, 1999.

[151] C. L. Valenzuela and A. J. Jones. Evolutionary divide and conquer (I): A novel genetic approach to the TSP. *Evolutionary Computation*, 1(4):313–333, 1994.

[152] C. L. Valenzuela and L. P. Williams. Improving simple heuristic algorithms for the travelling salesman problem using a genetic algorithm. In *Proceedings of the Seventh International Conference on Genetic Algorithms (ICGA97)*, pages 458–464, 1997.

[153] V. V. Vazirani. *Approximation Algorithms*. Springer, 2001.

[154] C. Voudouris and E. Tsang. *Handbook of Metaheuristics*, chapter Guided Local Search, pages 185–218. 2003.

[155] M. Wall. Galib: A C++ library of genetic algorithm components, 1996. Mechanical Engineering Department, MIT `http://lancet.mit.edu/ga`.

[156] D. de Werra and A. Hertz. Tabu search techniques: A tutorial and an application to neural networks. *OR Spectrum*, 11(3):131–141, 1989.

[157] J. Yang, X. Shi, M. Marchese, and Y. Liang. An ant colony optimization method for generalized TSP problem. *Progress in Natural Science*, 18(11):1417–1422, 2008.

[158] F. Zhao, S. Li, J. Sun, and D. Mei. Genetic algorithm for the one-commodity pickup-and-delivery traveling salesman problem. *Computers & Industrial Engineering*, 56(4):1642–1648, 2008.