Heuristics for Dynamic Vehicle Routing Problems with Pickups and Deliveries and Time Windows

Penny Louise Holborn

School of Mathematics
Cardiff University

A thesis submitted for the degree of
Doctor of Philosophy

May 2013

Summary

The work presented in this thesis concerns the problem of dynamic vehicle routing. The motivation for this is the increasing demands on transportation services to deliver fast, efficient and reliable service.

Systems are now needed for dispatching transportation requests that arrive dynamically throughout the scheduling horizon. Therefore the focus of this research is the dynamic pickup and delivery problem with time windows, where requests are not completely known in advance but become available during the scheduling horizon. All requests have to be satisfied by a given fleet of vehicles and each request has a pickup and delivery location, along with a time window at which services can take place.

To solve the DPDPTW, our algorithm is embedded in a rolling horizon framework, thus allowing the problem to be viewed as a series of static sub-problems. This research begins by considering the static variant of the problem. Both heuristic and metaheuristic methods are applied and an analysis is performed across a range of well-known instances. Results competitive with the state of the art are obtained.

For the dynamic problem, investigations are performed to identify how requests arriving dynamically should be incorporated into the solution. Varying degrees of urgency and proportions of dynamic requests have been examined. Further investigations look at improving the solutions over time and identifying appropriate improvement heuristics. Again competitive results are achieved across a range of instances from the literature.

This continually increasing area of research covers many real-life problems such as a health courier service. Here, the problem consists of the pickup and delivery of mail, specimens and equipment between hospitals, GP surgeries and health centres. Final research applies our findings to a real-life example of this problem, both for static schedules and a real-time 24/7 service.

Declarations

This work has not previously been submitted in substance for any other degree or award at this or any other university or place of learning, nor is it being submitted concurrently in candidature for any other degree or other award.

Signed .. .. (candidate) Date .....

Statement 1

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

Signed .. .. (candidate) Date .....

Statement 2

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

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 .....

Acknowledgements

Firstly I would like to thank my supervisors Jonathan and Rhyd. I am certain that this process would not have been as enjoyable and rewarding without your continued support and encouragement. Thank you for believing in me and allowing me to develop my skills as a researcher.

To all the valued friends I have made within the School of Mathematics at Cardiff University, you have become a part of the most exciting journey. I hope that the friendships I have made continue for many years to come. I am also grateful for the help and support of many colleagues who have guided me along the way.

I would like to express my gratitude to EPSRC, in particular to the LANCS Initiative, and to those who initially awarded me the funding. Without which none of this would have been possible. Also to those at the WAST HCS for giving up their valuable time to meet and discuss the service with me and for proving me with the data used in this thesis.

To my Mum and Dad, you have always believed that I am capable of achieving anything I set my mind to and have always encouraged me to work hard to obtain any goal. Thank you for providing me with the motivation to succeed. To my Sister, my Niece and my closest friends, your welcome distractions and encouragement have kept me going through to the very end.

Finally, thank you to Leigh, for being there every step of the way and for never doubting that this would be possible.

Acronyms

ACO	Ant Colony Optimisation
ADW	Advanced Dynamic Waiting
CT	Computational Time
CVRP	Capacitated Vehicle Routing Problem
CV	Coefficient of Variation
DARP	Dial-A-Ride Problem
DCVRP	Distance Constrained Vehicle Routing Problem
DDARP	Dynamic Dial-A-Ride Problem
DF	Drive-First
DMPVRP	Dynamic Multi-Period Vehicle Routing Problem
DPDPTW	Dynamic Pickup and Delivery Problem with Time Windows
DVRP	Dynamic Vehicle Routing Problem
DVRPTW	Dynamic Vehicle Routing Problem with Time Windows
DW	Dynamic Waiting
EA	Evolutionary Algorithms
GA	Genetic Algorithm
GGA	Grouping Genetic Algorithm
GRASP	Greedy Randomised Adaptive Search Procedure
HCS	Health Courier Service
ILP	Integer Linear Programming
LC1	Li & Lim benchmark instances [100] - Clustered Locations - Short time window
LC2	Li & Lim benchmark instances [100] - Clustered Locations - Long time window
LNS	Large Neighbourhood Search
LR1	Li & Lim benchmark instances [100] - Random Locations - Short time window
LR2	Li & Lim benchmark instances [100] - Random Locations - Long time window
LRC1	Li & Lim benchmark instances [100] - Random & Clustered Locations - Short time window
LRC2	Li & Lim benchmark instances [100] - Random & Clustered Locations - Long time window
LS	Local Search
NFL	Non-Fixed Requests
NFR	Non-Fixed Locations

NHS	National Health Service
NP	Non Deterministic Polynomial Time P roblems
NV	Number of Vehicles Required
OR	Operations Research
PDP	Pickup and Delivery Problem
PDPTW	Pickup and Delivery Problem with Time Windows
PSO	Particle Swarm Optimisation
RCL	Restricted Candidate List
RND8	Mitrovic-Minic et al. (2004) DPDPTW instances
RND9	Mitrovic-Minic et al. (2004) DPDPTW instances
SA	Simulated Annealing
SD	Standard Deviation
SWO	Squeaky Wheel Optimisation
TD	Total Travel Distance
TSP	Travelling Salesman Problem
VNS	Variable Neighbourhood Search
VRP	Vehicle Routing Problem
VRPB	Vehicle Routing Problem with Backhauls
VRPBTW	Vehicle Routing Problem with Backhauls and Time Windows
VRPPD	Vehicle Routing Problem with Pickup and Deliveries
VRPSPD	Vehicle Routing Problem with Simultaneous Pickup and Delivery
VRPTW	Vehicle Routing Problem with Time Windows
WAST	Welsh Ambulance Service Trust
WF	Wait-First

List of Publications

P. L. Holborn, J.M. Thompson, and R. Lewis. Combining heuristic and exact methods to solve the vehicle routing problem with pickups, deliveries and time windows. In J.-K. Hao and M. Middendorf, editors, *Evolutionary Computation in Combinatorial Optimisation (Lecture Notes in Computer Science vol. 7245*, pages 63–74. Berlin: Springer-Verlag, 2012.

List of Presentations

P. L. Holborn, J.M. Thompson, and R. Lewis. An Introduction to dynamic vehicle routing problems with pickups, deliveries and time windows. SCOR Conference, April 2010a.

P. L. Holborn, J.M. Thompson, and R. Lewis. Vehicle routing problems with pickups, deliveries and time windows. OR52 Conference, September 2010b.

P. L. Holborn, J.M. Thompson, and R. Lewis. Methods for dynamic vehicle routing problems with pickups, deliveries and time windows. IFORS Conference, July 2011.

P. L. Holborn, J.M. Thompson, and R. Lewis. Dynamic vehicle routing problem with pickups, deliveries and time windows. SCOR Conference, April 2012a.

P. L. Holborn, J.M. Thompson, and R. Lewis. Investigating the dynamic pickup and delivery problem with time windows. OR54 Conference, September 2012b.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Since the explosion of the internet we now live in a 'real-time', 'customer-oriented' society. The increase in the availability of computational and communications capabilities to large segments of the population has resulted in 'us' as customers being accustomed to a dynamic market for services and goods. Therefore the distribution industry responsible for delivering such goods must also perform in this real-time, customer-oriented environment (Sussman [2008]).

The work presented in this thesis concerns the problem of dynamic vehicle routing and the motivation for this is the increasing demands on the transportation services to deliver a fast and efficient service. Systems are now needed for dispatching transportation requests that arrive dynamically throughout the scheduling horizon and hence much research is needed into how best to schedule these real-time demands.

The rest of this chapter is structured as follows: Section 1.1 introduces the problem to be investigated within this research. A taxonomy for the vehicle routing problem (VRP) and its variants is outlined in Section 1.2 which provides a methodology for classifying the abundance of literature for the VRP. Section 1.2 also gives an informal overview of the particular VRP to be considered in this research. The main contributions of this research are outlined in Section 1.3, followed by an overview of each chapter in Section 1.4. A note on implementation and computational experimentation is provided in Section 1.5.

1.1 Research Problem

The vehicle routing problem (VRP) plays a central role in the scientific research involving the distribution of goods and people. In its simplest form it can be described as the problem of routing a set of requests, from a central depot to a set of locations, using a fleet of vehicles. Each vehicle departs from and returns to a depot and each request is serviced exactly once. This form of the problem will hence be referred to as the 'classical' VRP. Figure 1.1 shows an example of a solution to a classical VRP with 2 vehicles and 6 requests. Requests 1, 2 and 3 are serviced by one vehicle and requests 4, 5 and 6 by another.

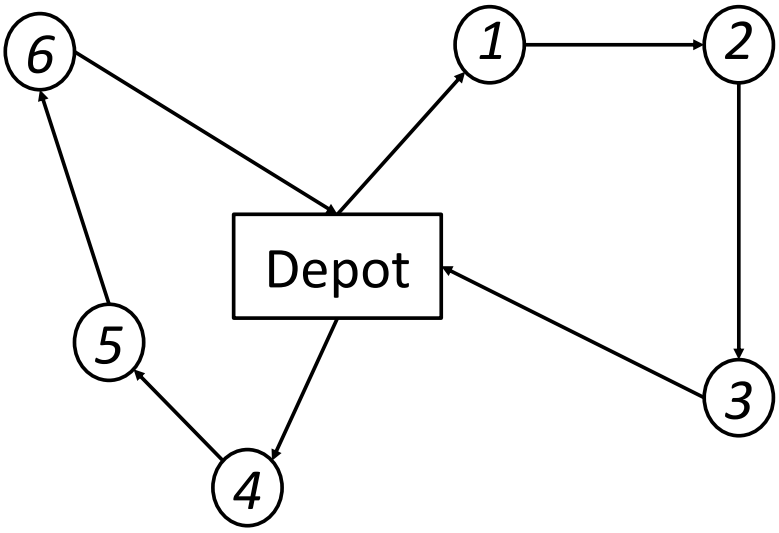


Figure 1.1: A solution to a simple VRP

Past research has mainly concentrated on the static variant of the problem where all requests are known in advance and no uncertainty exists. This results in the ability to schedule all requests prior to the beginning of service. In comparison, the dynamic problem is one where planning methods need to react to dynamically revealed information, e.g. the arrival of new requests during the scheduling horizon. In this case not all information is known in advance and so schedules need to be updated during service. Such problems are found in many real-life transportation domains, such as pickup and delivery courier services and dial-a-ride services (Gendreau and Potvin [1998] and Berbeglia et al. [2010]).

As outlined by Pillac et al. [2013] in a recent review of dynamic VRPs (DVRP), real-

world applications of the VRP often include two important dimensions: *evolution* and *quality* of information. The evolution of information relates to the fact that in some problems the information available may change during the scheduling horizon, for example with the arrival of new requests. The quality of information reflects possible uncertainty on the available data, for instance when requests have an unknown demand or where travel times are not constant. Based on these dimensions, Figure 1.2 identifies four categories of the DVRP.

		Information quality	
		Deterministic input	Stochastic input
Information evolution	Input known beforehand	Static and deterministic	Static and stochastic
	Input changes over time	Dynamic and deterministic	Dynamic and stochastic

Figure 1.2: Dimensions of DVRPs as outlined by Pillac et al. [2013]

This research will focus on both the static deterministic and the dynamic deterministic problem. In static deterministic problems, all the requests are known beforehand and vehicle routes do not change during service. In dynamic deterministic problems, part or all the information is unknown at the beginning of the scheduling horizon but revealed dynamically during the planning or execution of the routes.

Problems which are stochastic arise whenever some elements of the problem are random, e.g. demands or travel times. Sometimes, the set of requests to be visited is not known with certainty but with a given probability. Stochastic problems are not covered in this research but for an overview of stochastic DVRPs see Pavone et al. [2009].

This research will concentrate on a particular variant of the VRP known as the pickup and delivery problem with time windows (PDPTW). Applications of pickup and delivery problems are seen frequently in the area of transportation and logistics. Some applications include food and beverage distribution, currency collection and delivery between banks and ATM machines, internet-based pickup and delivery, and the transport of medical samples, to name just a few. The problem, in particular for internet-based pickup and delivery, is likely to become even more important in the future, due to the rapid growth in parcel transportation as a result of electronic commerce (e-commerce).

Online shopping or online retailing is a form of e-commerce whereby consumers directly buy goods or services from a seller over the internet without an intermediary service. In 2010, the UK had the biggest e-commerce market in the world when measured by the amount spent per capita, even higher than the USA. At the time, the internet economy in the UK was expected to grow by 10% between 2010 to 2015 (Kalapesi et al. [2010]). Hence the need for planning and schedules which can cope with the increasing demands.

1.2 VRP Taxonomy

A recent review of VRPs was carried out by Eksioglu et al. [2009] where a methodology for classifying the literature is defined quite broadly. It encompasses all of the managerial, physical, geographical and informational considerations as well as the theoretical disciplines affecting this ever-changing field.

Eksioglu et al. [2009] show that the majority of the research undertaken for the VRP studies the static variant of the problem. Our research will start with the static PDPTW to enable us to produce algorithms that can be adapted to the dynamic variant of the problem which has received significantly less attention in the literature. The taxonomy by Eksioglu et al. [2009] is shown in Figure 1.3.

This taxonomy gives an idea of the sophistication and diversity of the literature surrounding the VRP. It applies five major categories for classification:

1. **Type of Study :** This is based on the papers content and the nature of the study
2. **Scenario Characteristic :** This includes factors that are not a part of the constraints embedded into the solution
3. **Problem Physical Characteristics :** This includes the factors that directly affect the solution
4. **Information Characteristics :** This assesses the quality of the information from the solution
5. **Data Characteristics :** This classifies the type of data based on its origin

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 pickup and deliveries
     - 2.8.2. Nodes request either line haul or back haul 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
[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 1.3: Taxonomy of the VRP literature by Eksioglu et al. [2009]

By applying this classification scheme to the problem to be studied in this research the specific characteristics can be described more clearly. The definition of the dynamic PDPTW (DPDPTW) is based on that of Pankratz [2005b] and the PDPTW is based on that of Savelsbergh and Sol [1995]. The nature of the study will be centred on using mainly heuristic and metaheuristic approaches. The reasons for this will be outlined in the two literature reviews in this thesis found in Chapters 2 and 5.

The scenario characteristics of the static problem include a known number of requests with known service demands and time windows. For the dynamic variant not all requests will be known. Loads will not be split across vehicles and time windows will be strict in both variants of the problem. The time horizon will be a single period, such as a single day, when solving the PDPTW. However, as detailed in the review in Chapter 5, a multi period horizon will be adopted in the dynamic case. The precedence and coupling constraints for each request will hold, hence each request will have a pickup location (origin) and a delivery location (destination) associated with it. The precedence constraint ensures that the pickup location of a request is serviced before its corresponding delivery location and the coupling constraint ensures that the pickup and delivery locations of a single request are serviced by the same vehicle.

Physical characteristics of the problem include an undirected network for the transportation of requests between locations. Different geographical dispersion of locations will be considered and there will be a single depot as the sole point of origin.

An unlimited fleet of identical vehicles with a given capacity is chosen, due to ease of comparison with most previous approaches when referring to the standard PDPTW instances from the literature outlined in Section 3.3. Travel times will be treated as deterministic, with distance equal to time and the objective will be to minimise the total distance travelled again for ease of comparison.

The data used will be both synthetic and real and chosen to emulate, as much as possible, the difficulties that distribution companies face. For all requests to be satisfied, a given set of routes need to be planned, where each request is transported from its origin to its destination by exactly one vehicle. Each request has a size of load to be transported, and also a time window and loading times associated with the pickup and delivery location of each request. There is a maximum scheduling horizon and each vehicle must return to the depot before the end of the scheduling horizon. Requests cannot be refused and the arrival of requests is the only source of dynamics to be considered.

1.3 Summary of Contributions

This research aims to investigate methods to solve the PDPTW and in particular the dynamic variant of the problem, the DPDPTW. The majority of this thesis will be spent examining algorithms using heuristic and metaheuristic methods. Scientific investigations are performed using a range of well-known instances and results are compared with those from the literature. A real-world application of this research is then made. The following scientific contributions are made:

- The criteria for the neighbourhood operators in Chapter 3 are specifically designed to limit the computational time required enabling the algorithm to be adapted to a dynamic environment.
- The reconstruction heuristics developed in Chapter 3 are adapted from previous approaches in the literature where new criteria are introduced for combining multiple routes.
- The branch and bound heuristic introduced in Chapter 4 is based on other Large Neighbourhood Search (LNS) techniques applied to the PDPTW but has been further adapted to search all routes or subsets of routes to improve the ordering of locations.
- The tabu search heuristic introduced in Chapter 4 utilises a tabu attribute which has been applied to the PDPTW for the first time. Its applicability to a dynamic environment is highlighted, along with the different criteria the heuristic employs with regards to removing the aspiration criterion.
- In Chapter 4, we see that one of the main advantages of our final algorithm developed to solve the PDPDTW is the speed of constructing individual solutions. In this case it has allowed us to produce large samples of solutions in times that are consistent with other approaches. This advantage can be exploited when applying to the dynamic variant of the problem.
- Some of the methods applied in Chapter 4 generate results which are competitive with the state of the art results found in the literature. The results achieved obtain the best known solutions in 51 out of 56 instances with the algorithm appearing to perform consistently well over all types of instance. A new minimum total travel distance over all instances is also achieved.
- Chapter 5 provides an overview of the instances available in the literature for the DVRP and its variants. In particular the instances for the DPDPTW are

compared for the first time.

- The initial investigations performed into the DPDPTW in Chapter 6 provide insights into the characteristics of the problem which have not before been explored.
- The varied dynamic insertion and improvement criteria examined in Chapters 6 and 7 have not before been surveyed across varying instances. Insightful conclusions are made along with improving results for a range of instances sizes when compared to the best known results in the literature.
- For the first time in the literature methods for the DPDPTW are applied to a real-life example for a local health courier service (HCS)(Chapter 8). The specific constraints of this problem are responsible for its novelty and a range of initial investigations are performed to determine the current capacity and constraints on the service. This could lead to potential cost savings to the service by decreasing the distance travelled by the drivers and also by increasing the number of dynamic requests the service is able to accept.

1.4 Thesis Overview

The remainder of this thesis is structured as follows.

Chapter 2 overviews the wealth of literature for the VRP and its variants. It introduces the class of VRPs and in particular the evolution of the VRP to the PDPTW which is to be studied in this thesis. An overview of the solution methodologies applied to solve the VRP and its variants is provided, with a more detailed analysis into the specific approaches to be employed in this thesis.

Chapter 3 formulates the PDPTW as an integer linear program and introduces the standard instances for the problem. The remainder of the chapter is then concerned with investigating heuristic methods to solve the PDPTW. Initial insertion heuristics, neighbourhood operators and reconstruction heuristics from the literature are investigated and further adapted to the specifics of the problem.

Chapter 4 advances to investigate more sophisticated metaheuristics and evidence is provided on how applying a combination of these methods may be an effective way of tackling the problem. An investigation is performed to identify ways in which to reduce the computational time of our algorithm to enable it to be adapted to a dynamic

environment. Results are compared across a set of instances. The main outcomes from this chapter are also reported in Holborn et al. [2012].

Chapter 5 is dedicated to reviewing the literature for the DVRP and in particular the DPDPTW which is the overall focus of this research. A summary of the approaches taken to solve the DVRP are outlined and a detailed review of the methodology for the DPDPTW is provided. A review of the varied methods taken to adapt a static algorithm to a dynamic environment is included along with an overview of the relevant instances available in the literature for the problem.

Chapter 6 looks to adapt the algorithm developed in Chapter 4 to a dynamic environment. Different heuristic methods for incorporating the arrival of new requests are investigated, along with how the algorithm should be updated. The dynamic characteristics of the problem are examined, and in particular, investigations are performed on the effect of varying both the proportion of dynamic requests and also the degree of urgency of the requests. Comparisons are made with the results of Pankratz [2005b].

Chapter 7 concentrates on ways to improve the solutions of the DPDPTW and looks to validate the results achieved in Chapter 6. This is achieved by examining varying insertion and improvement criteria during the scheduling horizon. Comparisons in this case are made with the results of Mitrovic-Minic et al. [2004].

Chapter 8 examines a real-life example of a DPDPTW, specifically a HCS. A brief review of the research methods to solve transportation problems (i.e. VRPs) within a healthcare environment is provided. The complexities of healthcare related problems with regards to added real-life constraints are then incorporated into the algorithm. The aim of the chapter is to provide useful insights to improve future running of the service and to investigate the opportunities for expansion.

Chapter 9 summarises the main contributions of this research. It also provides ideas for further work in this area.

1.5 A Note on Implementation and Computational Experimentation

All algorithm implementations presented in this thesis are programmed in C++, using Microsoft Visual Studio 2010 and code optimisation set to maximise speed (/O2). The computational experiments were executed on a PC under Windows 7 with a 3.00GHz processor.

Chapter 2

Vehicle Routing Problems: A Literature Review

2.1 Introduction

The research into the 'classical' VRP and its extensions is both expansive and varied. Hence, this literature review will highlight only the main contributions and advancements in its history. The aim is to provide the reader with a relevant background to the particular variant of the VRP to be considered in this research, namely the VRP with pickup, delivery and time windows (PDPTW).

Section 2.2 introduces the literature for the 'classical' VRP before expanding this to show how it has evolved into the many complex variants which emulate the difficult real-life constraints faced today (see Section 2.3). Each VRP class will be outlined in more detail, with specific attention paid to the variants relevant to this research.

The first class to be described is the capacitated VRP (CVRP), which is the simplest and most studied member of the family and is discussed in Section 2.4. Next introduced is the VRP with time windows (VRPTW) followed by the VRP with pickup and delivery (VRPPD), these will be discussed in in Sections 2.5 and 2.6. The extension of these problems, the PDPTW, is the focus of this research and will be discussed in detail in Section 2.7.

A brief overview of the many approaches taken to tackle the VRP will be outlined in Section 2.8. This will provide a better understanding of how the research has progressed to the advanced metaheuristic approaches used today. A more detailed account of the methods specifically applied to solve the PDPTW, which aim to provide suitable

context for this research, are also provided including insertion heuristics, improvement heuristics and metaheuristics. The chapter is then concluded in Section 2.9.

2.2 The Vehicle Routing Problem

The vehicle routing problem (VRP) plays a central role in distribution management and has for a long time attracted attention in the field of Operational Research. The classic problem can simply be described as the problem of designing a set of routes from a depot to a set of locations.

The VRP was first introduced over 50 years ago by Dantzig and Ramser [1959] under the name: 'The Truck Dispatching Problem'. A real-world application concerning the delivery of gasoline to service stations was considered. Not only was the VRP first introduced to the research community, but the first mathematical programming formulation was proposed, and the first heuristic for solving the problem presented. To commemorate the 50th anniversary of this pioneering introduction, the main contributions in the history of the VRP are highlighted in the review by Laporte [2009].

A few years after the VRP was first introduced, Clarke and Wright [1964] improved on the approach of Dantzig and Ramser [1959] by proposing a savings heuristic, this is discussed in Section 2.8.2. The savings heuristic is both easy to implement and produced reasonably good solutions at that point in time. It has since become perhaps the most widely known heuristic for solving the VRP. Following the success of these two innovative papers, many models using exact and heuristic methods have been proposed to solve the VRP and its variants.

The early literature on the VRP concentrated on defining the problem and its complexities. Magnanti [1981] identified the extent and nature of the problem's complexities and in particular described several alternative models and new algorithms for the VRP which had not been considered at this point in time. It was shown that the prospects for applying exact methods, possibly in conjunction with heuristics, were far from fully realised by researchers. The complexity of the VRP was then fully summarised by Lenstra and Rinnooy Kan [1981], where it was shown that almost all routing problems are *NP*-hard and unlikely to be solvable in polynomial time, hence the need for heuristic methods. The VRP is an extension of the well-known travelling salesman problem (TSP) which is itself *NP*-hard, see Garey and Johnson [1979] for more information.

The early research into the classical VRP however still concentrated on exact methods. An extensive survey that is entirely devoted to exact algorithms for the VRP was

carried out by Laporte and Nobert [1987] where a complete and detailed analysis of the state of the art methods up until the late 1980s is provided. The survey showed that most types of VRPs at the point of writing remained virtually unsolved, as exact methods could only handle problems of relatively modest dimensions. The exact approaches at this time were able to address small VRPs with up to 50 requests and 8 vehicles.

Due to the limited success of exact methods, considerable attention and research effort since this has been devoted to the development of efficient approximate algorithms (or heuristics) which can provide near optimal solutions for large size problems. This is therefore the area we consider in this research.

The general principles of heuristic methods to solve practical VRPs was first studied in more detail by Christofides et al. [1979]. An early example in which a generalised assignment problem, with an objective function that approximates delivery cost, was applied is that of Fisher and Jaikumar [1981]. The heuristic has many attractive features as it always finds a feasible solution, if one exists, and it can be easily adapted to accommodate many additional problem complexities.

Neighbourhood search algorithms (see Section 2.8) up until the late 1980's mainly concentrated on the single-vehicle VRP. Cyclic transfer algorithms for multi-vehicle VRPs were first introduced by Thompson and Psaraftis [1993]. Cyclic transfers attempt to improve the cost of a set of routes by transferring small numbers of requests among routes in a cyclic manner. The results obtained revealed that this new class of neighbourhood search algorithms were either comparable to or better than the best published heuristic algorithms.

Due to the success of the early heuristic methods to solve the classical VRP the research in the early 1990's evolved to more complex heuristic algorithms and metaheuristic approaches, these are described in more detail in Section 2.8. Approximate methods based on descent, hybrid simulated annealing with tabu search, and tabu search algorithms were developed by Osman [1993]. The new methods improved significantly on both the number of vehicles required and the total distance travelled compared to a sample of test problems by Christofides et al. [1979]. The same set of instances are tested by Gendreau et al. [1994] using another tabu search heuristic. More information on tabu search can be found in Section 2.8.3.1.

There are several main survey papers on the 'classical' VRP including Bodin et al. [1983], Christofides [1985], Laporte [1992] and Fisher [1995]. A detailed bibliography by Laporte and Osman [1995] lists some of the main references on the subject of routing problems and concentrates on the most useful or significant publications, namely

references of general historic nature or classical articles. New developments, for example more robust algorithms, are considered in a survey by Bertsimas and Simchi-Levi [1996] where elements of uncertainty are also addressed.

A survey by Gendreau et al. [1997] shows the potential of applying local search (LS) algorithms to the VRP and highlights the developments in the areas of simulated annealing (SA), tabu search, genetic algorithms (GA) and neural networks. LS, in particular a Large Neighbourhood Search (LNS) is used in conjunction with constraint programming by Shaw [1998] and is shown to be much simpler than metaheuristic approaches and comparable in results. There are many research papers comparing the different methods for solving VRPs, for example the performance of descent heuristics is compared to metaheuristics by Van Breedam [2001].

Summarising the literature for the VRP, the early work by both Dantzig and Ramser [1959] and Clarke and Wright [1964] can be classified as the first generation of VRP research which relies on greedy methods and various local improvement heuristics. However, it is shown that the first generation methodology created in the 60's and 70's simply lacked the sophistication required to solve more complex, real problems faced by distribution companies. Success in the real world had to wait until the second generation of research started which began to apply mathematical programming techniques to solve the problem.

During the last decade the resources for achieving robustness have grown and hence rapidly decreasing computational costs are pushing the trade-off between computational time and solution quality in the direction of higher quality solutions. The base of fundamental research on which to draw has greatly expanded so to help interpret this, the following section sets out a classification scheme for the variants of the VRP.

2.3 The Class of Vehicle Routing Problems

A classification scheme is a hierarchical arrangement of classes. In terms of the vehicle routing class it is the set of all extensions to the classical VRP. Each class shares the fundamental characteristics of the classical problem but has developed in terms of complexity both with regards to design and the constraints. Examples of classification schemes for the VRP include an early approach by Bodin and Golden [1981] and Desrochers et al. [1990].

A formal definition of the basic problems of the vehicle routing class is given in Toth and Vigo [2002a]. Figure 2.1, taken from this book, illustrates the connections between

each class of problem. This problems included are the capacitated VRP (CVRP), the distance-constrained VRP (DCVRP), the VRP with backhauls (VRPB), the VRP with time windows (VRPTW), the VRP with pickup and delivery (VRPPD), the VRP with backhauls and time windows (VRPBTW) and the VRP with pickup, delivery and time windows (PDPTW). An arrow moving from problem A to problem B identifies that problem B is an extension of problem A. Hence the PDPTW is an extension of both the VRPTW and the VRPPD.

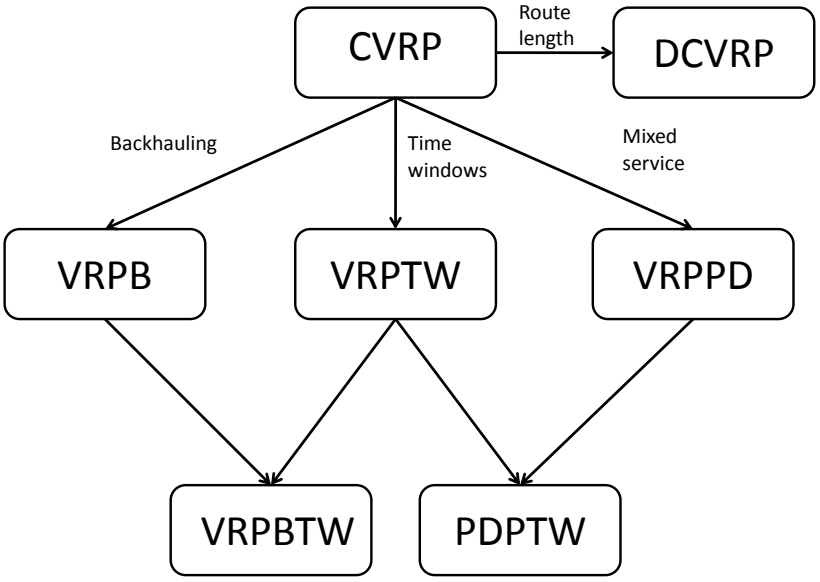


Figure 2.1: The VRP class and their interconnections by Toth and Vigo [2002a]

The extensions of the VRP relevant to the research presented in this thesis will be discussed in more detail in the following sections. These include the CVRP, the DCVRP, the VRPTW, the VRPPD, and finally, the PDPTW.

The variants of the VRP outside the scope of this research include the VRPB, also known as the line haul-back haul problem. For more information on this specific variant of the VRP see Toth and Vigo [2002c] and Parragh et al. [2008a]. The VRPB has also been extended to include time window constraints (VRPBTW), for recent literature on this see Tavakkoli-Moghaddama et al. [2006], Ropke and Pisinger [2006b] and Gajpal and Abad [2009].

2.4 The Capacitated VRP

The first extension to the VRP is the capacitated VRP (CVRP) in which a vehicle capacity constraint is imposed. This ensures that at any point in time the loads of all items present within a vehicle cannot exceed the capacity of that vehicle. Depending on the vehicle capacity and loads to be carried, this constraint could limit the number of requests that can be serviced by each vehicle. The CVRP, just like the VRP, is an extension of the TSP, therefore many approaches for solving it are inherited from the extensive research surrounding the TSP.

The branch and bound heuristic has been used extensively to solve the CVRP and its variants and this method will be discussed in more detail in Section 4.5. The survey by Laporte and Nobert [1987] discussed a complete analysis of branch and bound algorithms proposed until the late 1980's. In many cases these algorithms still represent the state of the art with regards to the exact solution methods for the CVRP; mainly due to the fact the algorithms have received relatively little interest in the literature since. This could be due to the fact that the algorithms may be very difficult to improve upon. More information on the CVRP can be found in Toth and Vigo [2002b].

The distance CVRP (DCVRP), which includes a maximum route length (time) requirement, was the first extension to the CVRP and was considered in Gaskell [1967] and again in Gillett and Miller [1974]. Christofides and Eilon [1969] consider three main solution methods for solving the DCVRP, (i) a branch and bound approach; (ii) the savings approach by Clarke and Wright [1964], and (iii) the 3-optimal tour method. The 3-optimal tour method was shown to achieve the best results and is discussed in more detail in Section 2.8.1.

There is limited literature on the VRP where only capacity (CVRP) and a maximum distance constraint (DCVRP) are taken into account. This is due to the fact that routing problems have evolved quickly to resemble more and more what is faced in real-life. It is difficult to find a real-life example of a VRP which does not have both capacity and distance time constraints imposed as a standard assumption. Hence, the research into the VRP quickly adopted both these constraints as standard features of the problem and began exploring the more difficult constraints faced. For more literature on the CVRP see Fischetti et al. [1994]. The following section will look at the next extension to the VRP, the addition of time windows.

2.5 The VRP with Time Windows

Time windows arise naturally in problems faced by business organisations that work on fixed time schedules and in the transportation of people and goods. As well as the vehicle capacity constraints (see Section 2.4), side constraints relating to time arise in almost every practical routing problem. The VRP with time windows (VRPTW) is an extension to the CVRP, where time dimension constraints have been incorporated. These constraints restrict the start of service at a location to begin no earlier than or at a pre-specified earliest time and earlier than or no later than a pre-specified deadline. For example, a parcel may need to be picked up from one location between 9:00am and 10:00am and will then need to be delivered on the same day between 4:00pm and 5:00pm.

Time windows can be considered as hard or soft, where in the case of hard time windows, if a vehicle arrives too early at a location; it is permitted to wait until service can begin. However, a vehicle is not permitted to arrive at a location after the latest time to begin service. For a solution to be feasible these hard time windows must be adhered to. In contrast, in the case of soft time windows, they can be violated at a cost. For more information on the VRPTW see Desrosiers et al. [1995].

Most of the research effort has been directed towards the hard time window variant, where specific examples of problems include bank deliveries, postal deliveries, industrial refuse collection and school bus routing and scheduling. A time window is often added to the depot in order to define a scheduling horizon and each route must then start and end within the bounds of this window.

The VRPTW is a generalisation of the VRP where the time windows are unbounded. Since the VRP is *NP*-hard, then the VRPTW is also *NP*-hard. In fact, even finding a feasible solution to the VRPTW when the number of vehicles is fixed, is itself a *NP*-complete problem (see Garey and Johnson [1979] for a formal definition). This is a corollary of the result derived by Savelsbergh [1985] for the case of a single-incapacitated vehicle. Consequently it may be difficult or impossible to construct a feasible solution, especially when time constraints are restrictive. On the other hand, an optimisation method may benefit from the presence of time constraints, since the solution space may be much smaller.

Perhaps due to the added computational challenges, the early work on the VRPTW was case study oriented. Examples of this can be found in, Pullen and Webb [1967], Knight and Hofer [1968] and Madsen [1976]. Pullen and Webb [1967] describes a system developed for duty scheduling of van drivers in a heavily time constrained environment.

Knight and Hofer [1968] presented a case study involving a contract transport company. Madsen [1976] developed a simple algorithm based on Monte Carlo simulation to solve a routing problem with tight time windows faced by a large newspaper and magazine distribution company.

After these initial studies the literature progressed once again to the early exact algorithms to solve the VRPTW. Desrosiers et al. [1984] attempted to solve a school bus transportation problem and Desrochers et al. [1992] successfully solved the linear programming relaxation of the set partitioning formulation for 100 requests.

Research after the early exact methods shifted focus once more to the development and analysis of heuristics able to solve larger problems. Heuristic algorithms for the VRPTW were first considered by Solomon [1987]. The initial approach extended on the savings heuristic proposed by Clarke and Wright [1964]. As well as checking time window constraints for violations, route orientation is now taken into account. Route orientation is the direction in which a vehicle services a route of locations assigned to it. This was negligible prior to the introduction of time window constraints, however in order to determine feasibility this needs to be declared. Efficient techniques for speeding up the process of rejecting infeasible solutions due to the violation of the time window constraints can be found in the extension to this, Solomon et al. [1988]. Extending on these results, Potvin and Rousseau [1995] describe and compare various iterative route improvement heuristics to solve the problem.

A summary of further heuristic methods to solve the VRPTW include a greedy randomised local search procedure (GRASP) by Kontoravdis and Bard [1995], a tabu search heuristic applied to the VRP with soft time windows by Taillard et al. [1997], a reactive tabu search heuristic developed by Chiang and Russell [1997], and an ant colony optimization (ACO) based approach by Gambardella et al. [1999]. Multiple heuristic methods for solving the VRPTW are investigated by Tan et al. [2001], namely SA, tabu search and GAs.

Survey papers for the VRPTW include Bräysy and Gendreau [2002] who surveyed the research on tabu search heuristics up until this time. Both traditional heuristic route construction methods and LS algorithms were examined in Bräysy and Gendreau [2005a] with the research on metaheuristic approaches in Bräysy and Gendreau [2005b]. A recent survey on solving large-scale VRPTWs is carried out by Gendreau and Tarantilis [2010]. The next section will discuss the VRP with pickup and delivery requirements.

2.6 The VRP with Pickup and Delivery

The VRP with pickup and delivery (VRPPD) is a variant of the VRP where each request is defined by a pickup location and a corresponding delivery location. This imposes both a coupling constraint and a precedence constraint on the original VRP. The coupling constraint is that each pickup and delivery location of a single request must be visited exactly once by the same vehicle. The precedence constraint is that the location of the request's pickup must be serviced before its corresponding delivery.

Figure 2.2 shows an example of a solution to a simple PDPTW with a single depot, 2 vehicles and 5 requests. Requests 1,2 and 3 are serviced by one vehicle and requests 4 and 5 by another. It is clear that the coupling constraints have been satisfied for all requests as the pickup location and delivery location of each request appear in the same vehicle. It can be seen from the orientation labeled on each route that the precedence constraints have also been satisfied as each pickup location appears before its corresponding delivery location in each route.

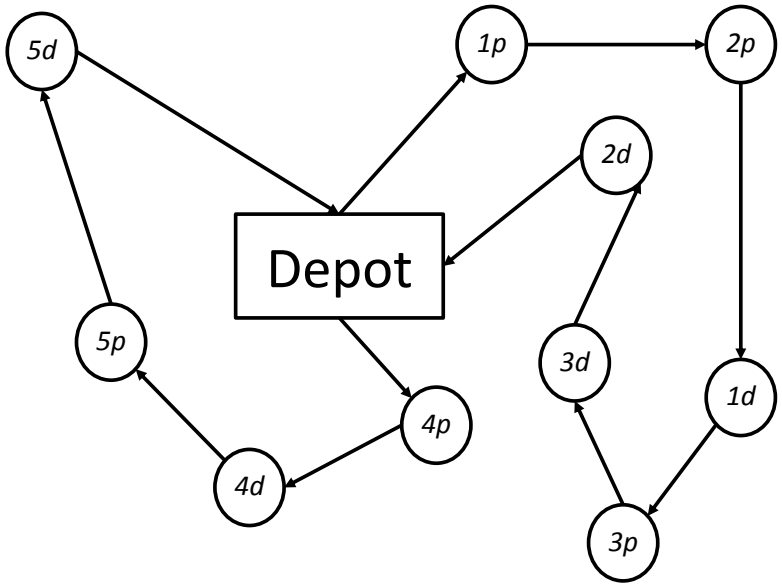


Figure 2.2: A solution to a simple PDPTW

When the requested transport involves people and not goods, this is known as the dial-a-ride problem (DARP). A common real-life example of a DARP includes door-to-door transportation for the elderly and the disabled (see Toth and Vigo [1996] and Toth and Vigo [1997]). The early work on the VRPPD was conducted for the DARP and first examined by Wilson et al. [1971], Wilson and Weissberg [1976] and Wilson and Colvin

[1977]. It was motivated by demand-responsive transportation systems and this work introduced the fundamental concepts of building tours through sequential insertion of requests and the general formulation of the problem that is widely used in the literature today. There are many differences when transporting people instead of goods, for one, the vehicle capacity constraints are usually much more constraining, also reduced user inconvenience must be balanced against minimising operational costs.

The single-vehicle DARP was first introduced by Psaraftis [1980]. Here, an optimisation technique is applied where every request requires service as soon as possible. The objective is to minimise a weighted sum of the time needed to service all requests and the total degree of 'dissatisfaction' based on lateness. An approximation technique for the single-vehicle DARP is considered again by Psaraftis [1983b] and the worst-case analysis of a simple two-phase approximation algorithm is described. A special case of the DARP, where there are no capacity constraints, is formulated as an integer program by Rutland and Rodin [1997] and a branch-and-cut algorithm is presented. For more information on the DARP see the review carried out by Cordeau and Laporte [2007].

The general pickup and delivery problem (PDP) was first introduced by Savelsbergh and Sol [1995] where an overview of the literature up until this point in time is provided. A general model that can handle the complexities of a PDP was also first presented and fully formulated. For an overview of the VRPPD see Desaulniers et al. [2002], and the survey papers of Berbeglia et al. [2007] and Parragh et al. [2008b].

Results obtained from the early research for the VRP show that the requests whose locations are geographically close to each other are likely to be serviced by the same vehicle. In the VRPPD this does not always happen due to each request having two locations which may not be close together. This is an important characteristic in the development of effective and efficient heuristics for the VRPPD.

The next section will combine the last two variants introduced and consider the PDPTW, which is the focus of this research.

2.7 The VRP with Pickup, Delivery and Time Windows

The VRP with pickup, delivery and time windows (PDPTW) can be found in a variety of real world applications, most commonly in courier services. It is a combination of the VRPTW (see Section 2.5) and the VRPPD (see Section 2.6). As well as capacity restrictions, the time window, pairing and precedence restrictions add further constraints to the problem. This constrained nature and real-world applicability is one of the reasons this specific variant is the focus of this research.

The early research surrounding the PDPTW started with the literature for the single-vehicle DARP, first discussed in Section 2.6. This is where people instead of goods are transported and, for the cases represented here, time windows are now taken into account. Time windows are often more restrictive in the DARP compared to that of the PDPTW as it is the customers who now specify an appropriate pickup or delivery time or both.

An early exact method for the DARP was that of Psaraftis [1983a] who modified the dynamic algorithm approach discussed in Psaraftis [1983b]. This algorithm could only handle up to 10 requests (20 locations), showing the limitations of the early exact methods. A similar dynamic programming approach was presented by Desrosiers et al. [1986], this time the algorithm could handle up to 40 requests. Sexton and Bodin [1985a] and Sexton and Bodin [1985b] consider a variant where desired delivery times are specified by the customers.

The literature advanced to the multiple-vehicle DARP where early approximation methods include that of Jaw et al. [1986], where an insertion algorithm was presented (see Section 2.8.1). For this case time windows for either the pickup or delivery of a request were defined based on a prescribed level of tolerance for lateness. For example, the customer could state they would allow 15 minutes either side of their desired pickup or delivery time for service to take place.

The first metaheuristic for the multiple-vehicle DARP can be found in Cordeau and Laporte [2003], where a tabu search heuristic was applied. This time the customer imposes a time window of pre-specified width on the arrival time of their pickup and the departure time of their delivery. An important feature of this approach is the allowance of infeasible solutions during the search. These solutions are then penalised in a weighted objective function which considers not only the cost of the solution but the total violation of load, total route time, time window and ride time constraints.

The research for the PDPTW again started with the case of a single-vehicle, and like many others, adapted methods already found in the literature. The single-vehicle PDPTW is considered by Van Der Bruggen et al. [1993] where a LS method based on the Lin-Kernigham algorithm for the TSP was developed (see Lin and Kernigham [1973] and Section 2.8.1).

The PDPTW for the multiple-vehicle case, which is the focus of our research, was first addressed by Dumas et al. [2001]. Here, a set partitioning formulation and a column generation scheme were presented to solve it to optimality. The algorithm was adapted from Desrosiers et al. [1986] and instances with up to 55 requests and 22 vehicles were solved. More information on the early literature for the PDPTW can be found in the survey paper by Savelsbergh and Sol [1995].

The first metaheuristic proposed to solve the PDPTW was the reactive tabu search approach of Nanry and Barnes [2000], (see Section 2.8.3.1). In this work a new set of instances based on those of Solomon [1987] were constructed. This was the first fully implemented method to be effectively applied to a set of up to 100 request multiple-vehicle instances. A two phase method by Lau and Liang [2001] where a construction heuristic is added to the tabu search algorithm of Nanry and Barnes [2000], improved on the previous results. Following this many more heuristic and metaheuristic methods have been applied to solve the problem.

Li and Lim [2001] have also produced instances for the PDPTW, which are also generated from Solomon's benchmark instances (Solomon [1987]), by pairing up the locations within routes. These have since been used as the main basis for comparison of algorithms for the PDPTW and will be used within this research (Section 3.3). A tabu-embedded SA algorithm was proposed by Li and Lim [2001] (see Section 2.8.3.1) and it was tested on the instances derived by Nanry and Barnes [2000] for comparison. A two-stage hybrid algorithm for the PDPTW was presented by Bent and Van Hentenryck [2006]. The first stage uses a simple SA algorithm to decrease the number of routes, while the second stage uses LNS to decrease the total travel cost. The results showed that there may be benefits in adopting solution approaches with more than one stage and is therefore something we consider in this research. An adaptive LNS heuristic was proposed by Ropke and Pisinger [2006a], see Section 2.8.3.2 for more details on this approach.

The use of population based algorithms to solve the PDPTW are found in the literature after the turn of the century (see Section 2.8.3.3). A grouping GA (GGA) was applied to solve the PDPTW by Pankratz [2005a], this work is further extended by Ding et al. [2009]. A memetic algorithm for the PDPTW using a selective route exchange crossover

was presented by Nagata and Kobayashi [2010b]. The first stage of the algorithm was a route minimising heuristic, based on a guided ejection search, presented in Nagata and Kobayashi [2010a].

Metaheuristics based on learning mechanisms are applied to solve the PDPTW by Lim et al. [2002] who applied a squeaky wheel optimisation and compared their results using the instances of Li and Lim [2001]. Dergis and Dohmer [2008] showed that the approach of indirect LS with greedy decoding produced results competitive with both Li and Lim [2001] and Pankratz [2005b]. More recently an application of the ant colony system was used to solve the PDPTW by Carabetti et al. [2010]. The approach was again compared to the results for the instances of Li and Lim [2001]. For more information on learning mechanisms see Section 2.8.3.3. The results published by Li and Lim [2001], Pankratz [2005a], Dergis and Dohmer [2008] and Ding et al. [2009] contain the best known results used for comparison in this thesis.

Other extensions to the PDPTW which are not to be considered in this research include the vehicle routing problem with simultaneous pickup and delivery (VRPSPD). This is where deliveries to be transported to locations are supplied from a single depot at the beginning of the scheduling horizon. Loads are required to be picked up from these locations and taken to the same depot at the end of the scheduling horizon. Real life examples of this problem occur in the soft drink industry where empty bottles must be returned. The VRPSPD was first considered by Min [1989], but there was then nearly 10 years without any work published on this problem. Due to the increased focus on environmental protection, re-usable packaging, and goods to be recycled or re-manufactured, research has again restarted into the problem. Recent articles include that of Montané and Galvão [2006] and Subramanian et al. [2010].

The pickup and delivery problem with transfer opportunities, where requests are allowed to be transferred between vehicles, was first considered by Shang and Cuff [1996] and is a further extension to the PDPTW. Mitrovic-Minic and Laporte [2006] consider transhipment where one vehicle collects the load to be transported at the pickup location, then drops it at a transhipment point, and another vehicle then transports the load to the delivery location. Recent literature on this problem includes that of Cortés et al. [2010] where the option for requests to be transferred from one vehicle to another at a specific location is added.

The methods applied to the PDPTW and those to be applied in this thesis will be discussed in more detail in the next section.

2.8 A History of Methods Applied in this Thesis

Over the past 40 years exact algorithms for the VRP have evolved from basic branch and bound schemes to highly sophisticated mathematical programming applications. However, the best exact algorithms for the PDPTW can still only solve instances involving approximately 100 locations (see Baldacci et al. [2011] and Baldacci et al. [2012]). As real instances often exceed this size and as solutions often need to be determined quickly, most algorithms suggested in the literature are heuristics or more recently powerful metaheuristics. For these reasons our research will concentrate on using these heuristic and metaheuristic approaches. The need for a solution to be determined quickly will become increasingly important when considering the dynamic variant of the problem where decisions need to be made in real-time (see Chapter 5). An extensive survey that was entirely devoted to exact algorithms for the VRP was carried out by Laporte and Nobert [1987].

Between 1964 and the early 1990s numerous heuristic methods were put forward to solve the VRP. Although some of them are purely constructive, the majority have an improvement phase. These heuristics are called 'classical' and they do not contain mechanisms allowing the objective function to deteriorate from one iteration to the next. This feature is present in the more recent metaheuristic approaches that have been developed over the past twenty years or so.

The remainder of this section provides further information on the approaches adapted to solve the PDPTW and the methods to be applied in this thesis.

2.8.1 Insertion Heuristics

Insertion heuristics for the VRP are purely constructive algorithms. They build feasible solutions by inserting, at each iteration, an un-routed request into a current partial route or into a new route. This process is performed either sequentially, one route at a time, or in parallel, where several routes are considered simultaneously. Sequential construction does not attempt to allocate an additional vehicle unless no more requests can be feasibly added to the existing routes. Parallel construction on the other hand, initially pre-specifies the number of vehicles to be used, but more vehicles can be added if the estimated number cannot feasibly service all requests. Two key questions are posed in the design of such methods: which request should be selected next for insertion, and where will the request be inserted? To address these questions researchers have considered criteria such as the minimum addition of distance or time and maximum

savings. A brief introduction to insertion heuristics will now be provided followed by an overview of insertion heuristics for the PDPTW.

Several construction heuristics for the VRPTW are proposed by Solomon [1987]. One insertion method is a time-oriented nearest neighbour heuristic. This initialises a route by finding the un-routed request closest to the depot. At each iteration the request closest to the last is added to the end of the route. When no feasible insertions are left in the existing route, a new route is added until all requests are assigned.

The most successful insertion heuristic of Solomon [1987] is the 'I1' heuristic. For this case a route is first initialised with a seed, the seed is selected by finding either the geographically furthest un-routed request in relation to the depot or the un-routed request with the lowest allowed starting time for service. At each iteration a new request is inserted into the current partial route between two adjacent locations in the best feasible insertion position. When no more feasible insertions can be found a new route is started until all requests are assigned. The criterion for insertion tries to maximise the greatest benefit derived from servicing a request on the partial route being constructed, rather than servicing the request on a single route. A similar idea is employed by Li and Lim [2001] for the PDPTW where in this case a route is first initialised with the request which has the maximum combined distance from the depot.

Some of the early literature on the use of insertion algorithms for the PDPTW is as follows. In Jaw et al. [1986] the authors develop an insertion heuristic where requests are selected in order of increasing earliest pickup time and are inserted into the route in a position which adds the lowest additional cost. The construction phase adopted by Van Der Bruggen et al. [1993] to solve the single-vehicle PDPTW starts with an initial route obtained by visiting the locations in order of increasing centres of their time window, taking precedence and capacity constraints into account. A greedy insertion heuristic is applied by Nanry and Barnes [2000] which at each iteration chooses the insertion with the lowest additional cost to the objective function.

A more recent insertion based construction heuristic for solving the PDPTW is considered by Lu and Dessouky [2006]. It not only considers the classical increment in distance, but also the cost of reducing the slack time due to the insertion. The slack time at a location is referred to as the difference between the end of the time window for service at that location and the actual time that service takes place. So instead of always choosing the request with the lowest cost of insertion, it may be better to select an insertion which does not use as much of the available slack to leave more opportunities for future insertions. We will look to exploit this when considering the dynamic problem later in this research.

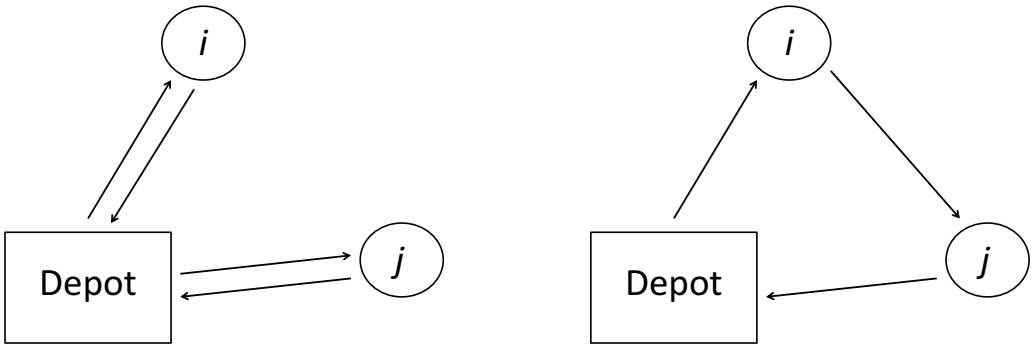
A recent review of construction heuristics for the PDPTW was carried out by Hosny and Mumford [2009b]. Several construction heuristics are also investigated in this work with the aim of finding methods capable of producing good starting solutions for metaheuristic algorithms. A simple sequential algorithm was developed that produced comparable results, yet is simple to code and fast to run. The main difference between this algorithm and other insertion heuristics is that it does not try to find the best insertion position for each request in the route, but accepts any feasible insertion. Therefore the algorithm eliminates bias towards either the pickup or the delivery location, which is one of the main drawbacks of the 'classical' insertion methods.

These methods will further be investigated in our research of the PDPTW in Section 3.4. The next section will overview the literature for improvement heuristics.

2.8.2 Improvement Heuristics

Two types of move operator can be defined for attempting to improve a solution, *intra-route* moves and *inter-route* moves. Intra-route moves consist of improving each route separately, whereas inter-route moves act on several routes simultaneously. It is common in the literature to alternate between these two methods within the same improvement heuristic. This section provides further information on the improvement heuristics adapted to solve the PDPTW and the methods which will be applied in this thesis. It begins with an introduction into the main improvement methods developed to solve the VRP.

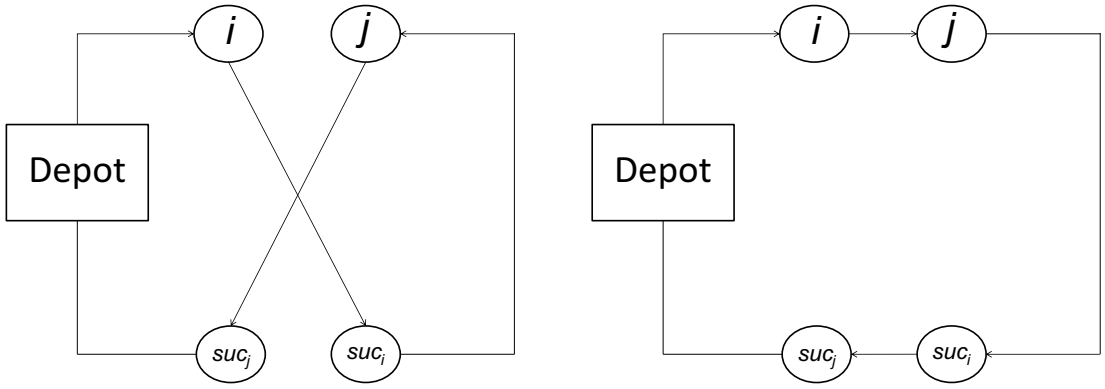
One of the first improvement heuristics for the VRP was the Clarke and Wright [1964] savings heuristic. It starts with an initial solution made up of a back and forth route to each request from the depot. At each iteration, it merges a route ending with location i with another route starting with location j, maximising the saving s_{ij}, where $s_{ij} = c_{i0} + c_{0j} - c_{ij}$ and c_{ij} is the cost of the route travelling from location i to location j, where location 0 is the depot. This is provided the merge is feasible and the process stops when no more feasible routes can be merged. An example of the saving heuristic for 2 locations is provided in Figure 2.3.



(a) Before heuristic is applied (b) After heuristic is applied

Figure 2.3: Savings Heuristic by Clarke and Wright [1964]

Route improvement methods iteratively modify the current solution by performing local searches for better neighbouring solutions. In the case of arc exchanges, a neighbourhood comprises of the set of solutions that can be reached from the present one by swapping a subset of k ($k \in \mathbb{Z}$) arcs between solutions. Perhaps the best known heuristic for the TSP is the arc exchange heuristics of Lin [1965].

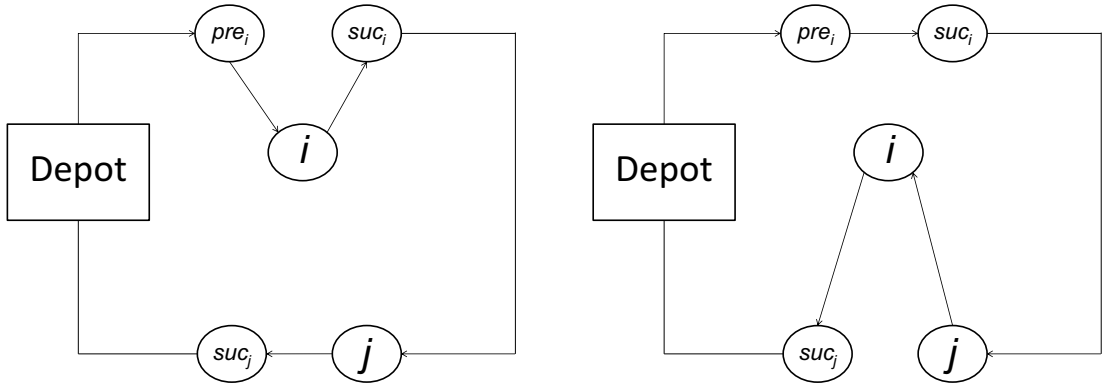


(a) Before heuristic is applied (b) After heuristic is applied

Figure 2.4: *2-opt* Arc Exchange Heuristic by Lin [1965]

Here, the *k-opt* procedure is one where the term '*k optimal*' implies that no further improvement can be made by removing k arcs in the solution and replacing them by k others. Van Der Bruggen et al. [1993] propose a local improvement procedure for the single-vehicle PDPTW based on arc-exchanges following the variable-depth search procedure of Lin and Kernigham [1973] for the TSP. Figure 2.4 shows an example of the arc exchange heuristic of Lin [1965] for the case of a *2-opt* procedure.

For the case of edge exchanges, Or [1976] introduced an operator designed for the TSP which has since been adapted to the VRP and its variants. An *Or-opt-1* exchange considers each request in turn and tries to improve the solution by re-inserting the request at another location. The *Or-opt* heuristic extends this by considering sequences of 1, 2 and 3 adjacent locations in a solution. For more information on handling edge exchanges in the VRP see Kindervater and Savelsbergh [1997]. Figure 2.5 shows an example of the edge exchange operator of Or [1976] for the case of the *Or*-opt-1 exchange.

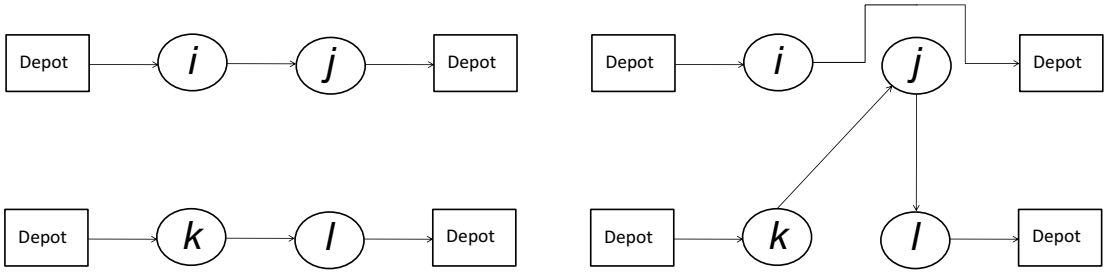


(a) Before heuristic is applied (b) After heuristic is applied

Figure 2.5: Edge Exchange Operator by Or [1976]

Another well-known improvement heuristic, the λ-interchange generation mechanism ($\lambda \in \mathbb{Z}$), was first applied to the VRP by Osman [1993]. From an initial solution, a λ-interchange between a pair of routes is the replacement of a subset of the first route, with a subset of the second, to get two new routes and a new neighbouring solution. The neighbourhood of a given solution is the set of all solutions generated by the λ-interchange mechanism for a given λ, where the size of each subset selected must be less than or equal to λ. The order in which neighbours are searched must be specified.

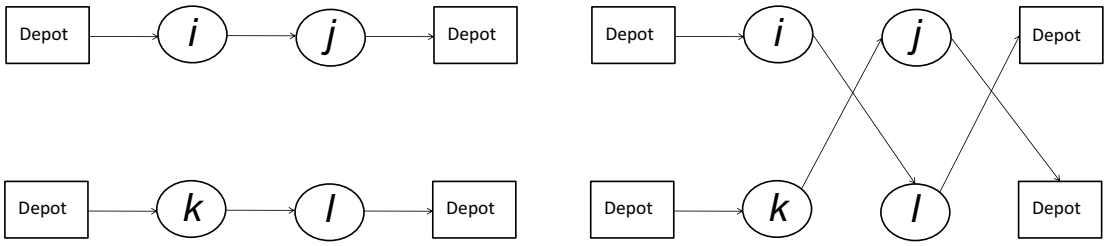
Considering the case where $\lambda = 1$, i.e. a subset may be of size zero or one, then the 1-interchange mechanism uses two processes to generate neighbouring solutions. A *shift* process denotes the shift of one request from one route to another. If the first vehicle selected only contains one request then this would result in one vehicle having no requests assigned to it, hence, the number of vehicles would be reduced. This is an important property of the λ-interchange generation mechanism. The second process considered is an *interchange* process which exchanges a request from one route with a request in another route.



(a) Before heuristic is applied (b) After heuristic is applied

Figure 2.6: Shift Operator by Osman [1993]

Figure 2.6 shows an example of the *shift* operator and Figure 2.7 shows an example of the *interchange* operator. These have been extended to the PDPTW by Li and Lim [2001] and will be investigated within this research.



(a) Before heuristic is applied (b) After heuristic is applied

Figure 2.7: Interchange Operator by Osman [1993]

Two methods for accepting alternative solutions when implementing iterative improvement heuristics to the VRP are discussed by Osman [1993]. The *best-improve* strategy examines all solutions in the neighbourhood of the current solution and accepts the one which yields the best solution according to a given acceptance criterion. The *first-improve* strategy immediately accepts the first solution in the neighbourhood which satisfies the acceptance criterion. These will be examined in more detail whilst investigating the PDPTW in Section 3.7.

Various iterative route improvement heuristics are compared by Potvin and Rousseau

[1995] for the VRPTW where it is clear that only a few improvement heuristics are useful when time windows are the main concern. It is shown that the classical *k-opt* exchange heuristic, defined above, is not well adapted to problems with time windows, as routes are usually sequenced according to their time windows. Hence, it will not be investigated further in this research.

A descent algorithm is a local search method which attempts to improve a solution via an iterative improvement method, accepting only improving solutions, and stopping when a local optimum has been achieved. These algorithms are flexible and simple to implement, however they have major limitations. The local optimum achieved may be far from the global optimum, hence the creation of metaheuristics to overcome local optimality. A survey by Van Breedam [2001] compares descent heuristics to metaheuristics for the VRP. The next section will overview the literature on metaheuristic approaches applied to the PDPTW.

2.8.3 Metaheuristics

Metaheuristics are procedures designed to find a good solution to difficult optimisation problems. Metaheuristics make few assumptions about the problem being solved, and so are useable for a variety of problems (see Blum and Roli [2003]). In general they contain mechanisms allowing the objective function to both worsen and improve from one iteration to the next, with a bias towards accepting improving moves. This allows a larger search space of solutions to be explored compared to the case where only improving changes to the objective function are made. Metaheuristic algorithms for the VRP have been developed over the past twenty years and have often improved on the earlier approaches. The stopping criteria for metaheuristics need to be determined, generally the longer the computing time, the higher the probability of finding a better solution. One drawback of metaheuristics, as is the case with heuristics methods, is that there is no guarantee of solution quality.

Using the classification of Laporte [2009] metaheuristics can be broadly classed into three main categories, local search (LS), population search and learning mechanisms. In a survey by Gendreau et al. [1997] it is shown that impressive computational results can be achieved by applying LS algorithms to solve the VRP. A more thorough analysis reveals that the various approaches are not equally successful and that a fair amount of problem-specific knowledge must be embedded in any algorithm. In this research we will concentrate on LS based metaheuristics as these have produced promising results in the literature so far with regards to the PDPTW and in particular, tabu search and LNS. These methods are also able to provide good quality solutions in a reasonable

amount of time which is crucial when considering the dynamic variant of the problem.

A software library of LS heuristics that allow a user to quickly generate solutions to VRP instances was created by Groër et al. [2010]. The core of the library is the implementation of seven LS operators, most of which have been discussed previously in Section 2.8.2.

An introduction to tabu search and LNS followed by their applications in solving the PDPTW will be provided in the following section. A brief overview will then be given on both population search and learning mechanisms for the PDPTW. For survey papers on metaheuristics applied to the VRP see, Laporte and Osman [1995], Gendreau et al. [1997] and Bräysy and Gendreau [2005b].

2.8.3.1 Tabu Search

Tabu search was first introduced by Glover [1986] and has been used to solve many problems. Tabu search is an extension on steepest descent in that it does not stop when no further improvement can be found and a local minimum has been reached. Instead the best solution found in the current neighbourhood is selected, even if it is worse with regards to the objective function, than the best solution found so far or the current solution. This could allow the search to escape from a local minimum.

A tabu list, often referred to as a short-term memory, is used to store information regarding the latest modifications to a solution. The algorithm is able to determine whether a solution with a specific attribute has been visited before, therefore preventing cycling. If the latest modification to a solution is deemed 'tabu', then this is only allowed where the modification yields an improvement to the best solution found so far. Determining the length of a tabu list (often known as the tabu tenure) is a vital design feature of tabu search as it determines how many iterations a specific modification to the solution remains in the tabu list. One example of a stopping criterion for tabu search can be encountering a given number of iterations without gaining an improvement to the best found solution so far.

For more information on determining an appropriate tabu tenure and stopping criteria for solving the VRP see Osman [1993]. There are many methods used in which to enhance a tabu search including intensification and diversification of the search, a survey on tabu search heuristics for the VRPTW carried out by Bräysy and Gendreau [2005b] provides more details on this.

There are various applications of a tabu search heuristic applied to the PDPTW, the

first of these being by Nanry and Barnes [2000] who present a reactive tabu search approach to solve the PDPTW. Reactive tabu search differs from the classical in that it monitors previously visited solutions and dynamically adjusts the algorithms search parameters based on its assessment of the quality of that exploration. For example, it may increase or decrease the length of the tabu list to diversify or intensify the search. The attribute stored in the tabu list at each iteration is the request number and the position of that request in the solution, i.e. its route and its position in that route.

Results show that the dominance of the precedence and coupling constraints are critical in developing appropriate strategies for the PDPTW and a major factor in the marked efficiency exhibited by the algorithm. Lau and Liang [2001] improved on the methods of Nanry and Barnes [2000] by using the same tabu search heuristic but this time paired with a construction heuristic. This provides evidence once more that there may be advantages in combining more than one heuristic or metaheuristic approach in an algorithm.

Li and Lim [2001] use a tabu-embedded SA approach to solve the PDPTW. Their approach differs to previous approaches as the attribute to be stored in the tabu list is an eigenvalue structure used to represent a solution. It uses the number of vehicles, total travel cost, total schedule duration and total waiting time to distinguish between solutions. Since the probability of two different solutions having the same eigenvalue is very small, it is reasonable to regard two solutions as the same if they share the same eigenvalue. Their results are compared to that of Nanry and Barnes [2000] and it is concluded that their approach is the first to solve practical sized multiple-vehicle PDPTW problem instances with various distribution properties.

The tabu search heuristic adopted be Gendreau et al. [2006] for the PDPTW follows the general guidelines provided by Glover [1989]. An adaptive memory and a decomposition procedure are added to this basic scheme to diversify and intensify the search. For the adaptive memory, a pool of routes taken from the best solutions visited thus far is exploited to restart the search in a new unexplored region of the search space. The decomposition procedure focuses the search on smaller sub-problems. In this case, the tabu list exploits the objective value of the new solution produced through a particular move. Results show that the tabu search heuristic can cope with complex dynamic environments and therefore is an attractive method for this research.

A tabu search algorithm is used to solve the VRP with simultaneous pickup and delivery by Montané and Galvão [2006]. Every movement in the solution space is characterised by two sets of edges which define its attributes. These are the edges to be inserted into the solution and the edges to be removed from the solution. A movement is considered

'tabu' when the edges to be inserted into the solution and the edges to be removed from the solution are 'tabu'. Best results were achieved by applying fixed tabu tenures proportional to the number of requests. This attribute will be investigated for the PDPTW in Chapter 4. The search is intensified and diversified using information kept on the most frequently used edges. Results showed that neither intensification nor diversification of the search significantly improved the final solution for the instances used here. This is something we will need to consider in this research.

This section has highlighted the key decisions which need to be made when applying a tabu search algorithm to solve the PDPTW and provides evidence of the competitive results which can be achieved by applying this metaheuristic approach. The key decisions to be taken include the initial solution generation, the attribute to be stored within the tabu list, the length of the tabu list, the neighbourhood and the stopping criteria. These will be discussed in more detail in Section 4.3.

2.8.3.2 Large Neighbourhood Search

LNS is based on a process of continual relaxation and optimisation. In the context of VRPs the technique might explore a large neighbourhood of a current solution by selecting a number of requests to remove from the solution and then re-inserting these requests using a constraint-based search tree. Two factors can affect the way in which the search operates, firstly how the set of requests is chosen for removal, secondly, the process used to re-insert them.

Shaw [1998] introduces LNS to solve both the VRP and the VRPTW. In this work a 'relatedness function' to decide which requests to remove from the solution is applied. For example if two locations were closely located these would be highly related, as if one of these is removed from a route and re-inserted into another route, then both would need to be removed. The re-insertion process uses a branch and bound technique with constraint propagation and heuristics.

A two-stage hybrid algorithm for the PDPTW is presented by Bent and Van Hentenryck [2006] which adapts the method of Shaw [1998] to the PDPTW. The first step is a simple SA algorithm applied to limit the number of vehicles and the second a LNS which is applied to minimise the total travel cost. After removal of a set of requests, a single request is chosen at random to be reinserted into the solution. The remaining requests to be inserted are sorted based on the relatedness criterion to the first request. This approach is chosen based on the fact that Shaw [1998] found a weakness in their LNS with regards to the instances of Solomon [1987]. For those with a longer scheduling

horizon, it failed to limit the number of vehicles required.

Ropke and Pisinger [2006a] also apply LNS to the PDPTW. Their method consists of a number of competing sub-heuristics that are used with a frequency corresponding to their historic performance. This adaptive LNS differs from earlier methods in the fact that a number of removal and insertion heuristics are applied during the same search. Simple and fast heuristics are used for the insertion method as opposed to the branch and bound and SA methods used previously.

This highlights the benefits of applying a LNS with regards to the PDPTW, in particular in conjunction with other LS methods. This will be examined within our research in Section 4.5. The final section overviews the literature on the PDPTW that adopts other metaheuristic approaches.

2.8.3.3 Other Metaheuristic Approaches

The following metaheuristic approaches are outside the scope of this research, but a brief overview of each method is provided followed by an example of how it has been applied to the PDPTW.

SA algorithms have not been well researched in the area of VRPs. This may be due to the fact that initial research found they did not perform well compared to other metaheuristics. They will therefore not be investigated within this research. It should be noted however, SA has been applied successfully to the PDPTW in hybrid approaches, see Li and Lim [2001], where a hybrid metaheuristic called tabu-embedded SA is developed.

A population search works with a population of solutions, GAs are the best known examples of this. A GA is an adaptive heuristic search method based on population genetics, the basic concepts were developed by Holland [1975]. A GA evolves a population of individuals encoded as chromosomes by creating new generations of offspring through an iterative process. At each iteration, parents are extracted from the current population and recombined to create offspring and the worst solutions in the population are then removed. This allows characteristics of good solutions to be passed from one generation to the next. It is also standard to apply a diversification mechanism, called mutation to the offspring before considering their inclusion in the population. An example of a random mutation for the VRP could be swapping locations in a single route with a small probability.

The VRP with capacity constraints, distance constraints, time windows or precedence

constraints are more difficult to encode on a chromosome than the TSP, therefore sophisticated permutation operators must be developed (Gendreau et al. [1997]). A fair amount of ingenuity is required to apply GAs to VRPs and as a consequence there are fewer examples of this within the literature and there is much less computational evidence available to assess their potential.

Pankratz [2005a] is the first to present a GA for the PDPTW. Here, a grouping GA (GGA) is proposed where each gene in a chromosome represents a group of requests that are assigned to a single vehicle, rather than a single request. The length of the chromosome is therefore equal to the number of vehicles in the solution. As a consequence, a chromosome only covers the grouping aspect of the requests and not the routing information; instead, the routes are constructed and maintained using a separate heuristic. A multi-strategy grouping GA is studied by Ding et al. [2009], which improves on the work of Pankratz [2005a].

Learning mechanisms use information previously generated about good solutions, i.e. at each iteration they use the information obtained from previous iterations. Ant Colony Optimisation (ACO) is a form of a learning mechanism which mimics the behaviour of ants foraging for food and laying pheromone on their paths. For the VRP, this idea translates into gradually giving more weight to the routes between two locations appearing frequently in good solutions.

An ACO is applied to the PDPTW by Carabetti et al. [2010]. The proposed methodology can be divided into two phases, the construction phase and the refinement phase. In the construction phase, an initial solution is created through an ACO metaheuristic with the elitism concept. This concept is one in which only the best ant can lay down pheromone, this attempts to improve the generated solutions by the ants at each iteration of the construction phase. The refinement phase is made up of a descent method with three neighbourhood structures. After the solution is refined the pheromone is layered over the selected route in order to guide the next search.

Another learning mechanism which has recently been applied to the PDPTW by Lim et al. [2002] is squeaky wheel optimisation (SWO), first introduced by Joslin and Clements [1999]. SWO is a *construct-analyse-prioritise* cycle, where an initial solution is constructed by a greedy algorithm. Decisions are made in an order determined by priorities assigned to the elements of the problem. The solution is then analysed to find the elements of the problem that are causing 'trouble' and 'blame' is assigned to them. The priorities of the trouble makers are then increased according to the magnitude of the blame, causing the greedy constructor to deal with these sooner on the next iteration. The two neighbourhood operators of Li and Lim [2001] (Section 3.6)

are used to improve the solution and this cycle repeats until a termination condition occurs. The priorities sort the requests first on vehicle blame, then travelling distance blame, then schedule duration blame and finally waiting time blame.

Dergis and Dohmer [2008] discuss an indirect (evolutionary) LS heuristic for the PDPTW. In indirect search, solutions are encoded such that the problem of securing feasibility is separated from the metaheuristic search process. Here a greedy decoding is used and the LS procedure is based on the 2-exchange neighbourhood similar to Li and Lim [2001].

2.9 Chapter Summary

In this chapter the VRP literature was introduced which included a brief history of its progress and its expansion over the last 50 years. This was followed by a summary of the class of VRPs and in particular the evolution of the VRP to the PDPTW to be studied in this thesis. In the next chapter the PDPTW will be discussed in more detail and the problem will be mathematically formulated.

It is shown that the early research into the VRP and its variants centred on exact methods; however their limitations with regards to problem size and the complexity of real-life problems quickly became apparent. The majority of the research effort over the last 20 years has since centred on heuristic and more recently metaheuristic approaches. These are appropriate for this research as they produce competitive results quickly, which is crucial when looking to adapt the problem to the dynamic variant.

The literature review highlighted that the main heuristic approaches taken to solving the PDPTW generally include a construction phase followed by an improvement phase, this approach will be followed within this research.

The main aim of the next chapter is to reflect on the aforementioned research and determine a fast and effective method to solve the PDPTW that can be adapted to a real-time setting for use within a dynamic environment.

Chapter 3

Heuristic Methods for the PDPTW

3.1 Introduction

This aim of this chapter is to produce an algorithm for the PDPTW using heuristic methods. The benefits of applying heuristic methods to VRPs, and in particular to the PDPTW, are described in detail in the literature review in Chapter 2. It is shown that, compared to most exact methods, they are capable of producing acceptable solutions to realistic problems within a reasonable amount of computational time. As this research aims to adapt the PDPTW algorithm to a dynamic environment, the algorithm developed will need to be suitable for use in real-time. Hence, minimising the computational time spent achieving a good solution is one of the main criteria for our algorithm, further supporting the use of heuristic methods.

The PDPTW, to be considered in this research, can be described as follows and is based on the definition of Savelsbergh and Sol [1995]. The problem is concerned with routing a fleet of vehicles to service a set of requests, from a central depot to a set of locations. Each vehicle must start at the depot and return to the depot before the end of the scheduling horizon. The pickup and delivery of a single request must be serviced by the same vehicle and a request's pickup must be scheduled in a route before its corresponding delivery. The total volume of all loads within a vehicle at any one time must not exceed the maximum capacity of that vehicle and the requests' pickup and delivery time windows must be adhered to. A vehicle may wait at a location if it arrives at that location before service can begin. All requests are known in advance and no uncertainty exists for this static variant of the problem.

The rest of the chapter is structured as follows. Section 3.2 provides a mathematical formulation of the problem in terms of an integer linear programming model. The

instances from the literature for the PDPTW, which are to be used to evaluate our results, are detailed in Section 3.3.

Section 3.4 investigates methods for constructing initial solutions based on insertion heuristics from the literature (see Section 2.8.1) and results are provided in Section 3.8. Section 3.6 introduces two neighbourhood operators for the PDPTW, previously applied by Li and Lim [2001], to attempt to improve on the initial solutions. Next, Section 3.7 investigates the criteria to be used for the neighbourhood operators and Section 3.8 provides results for the PDPTW after applying the insertion heuristics combined with the neighbourhood operators.

The reconstruction heuristics developed in Section 3.9 are intended to further improve the results. The final results obtained are provided in Section 3.10 and the chapter is concluded in Section 3.11.

3.2 Mathematical Formulation

Within this section the PDPTW will be fully formulated as an integer linear programming (ILP) model. ILP models are used in a wide variety of applications including routing and scheduling. The decision variable, indicating whether a specific request is assigned to a particular route, can only take the value of a 0 or a 1. The discrete nature of the variables gives rise to a combinatorial explosion of possible solutions, further supporting the use of approximation methods such as heuristics to tackle such a problem.

An ILP in standard form is expressed as:

$$\begin{aligned} \text{maximise } & c^T x \\ \text{subject to } & Ax = b, \\ & x \geq 0, \\ & \text{and } x \text{ integer}, \\ & \text{where the entries } c, b \text{ and } A \text{ are integer} \end{aligned} \tag{3.1}$$

To define the PDPTW, let M be the number of vehicles used, C be the maximum length of the scheduling horizon and Q the maximum capacity of each vehicle. Let $V = \{v_0, v_1, \dots v_n\}$ be a set of geographically dispersed locations where v_0 denotes the depot and n is even. The set $N = V \setminus \{v_0\}$ defines the set of locations for the requests and is partitioned into two subsets of equal size. The subset N^+ denotes the set of pickup locations and N^- the set of delivery locations. Therefore, $N^+ \cup N^- = N$, $N^+ \cap N^- = \emptyset$ and $|N^+| = |N^-| = \frac{n}{2}$ = number of requests. For each pair of locations (v_i, v_j) $(0 \leq i \neq j \leq n)$ a non-negative distance t_{ij} is known. In our case $t_{ij} = t_{ji}$ and distance is assumed to be equal to time.

In this problem, each location $v_i \in N$ has an associated demand q_i, a service time s_i and a service time window $[e_i, l_i]$. For this case, e_i is the earliest time that service at location v_i can begin and l_i is the latest time that service at location v_i can begin. With regards to the demand, $q_i > 0$ for all $v_i \in N^+$ and $q_i < 0$ for all $v_i \in N^-$. For the depot, $q_0 = 0$, $s_0 = 0$, $e_0 = 0$, and $l_0 = C$.

To formulate the PDPTW as an ILP model, the following variables are introduced:

$$x_{ij}^k = \begin{cases} 1, \text{ if vehicle } k \text{ goes from location } i \text{ to location } j \\ 0, \text{ otherwise.} \end{cases}$$

y_i = load of the vehicle servicing location v_i, after service is complete, $y_0 = 0$

a_i = the arrival time at location v_i

d_i = the departure time at location v_i

If a vehicle reaches location v_i before time e_i, it needs to wait until e_i before the service can take place. Let w_i be the waiting time at location v_i, then if $a_i < e_i$, $w_i = e_i - a_i$.

The following constant is also introduced:

$$z_{ij} = \begin{cases} 1, \text{ if location } v_i \text{ and location } v_j \text{ are the corresponding pickup and} \\ \text{ delivery locations of a single request} \\ 0, \text{ otherwise.} \end{cases}$$

$$\sum_{k=1}^{M}\sum_{j=1}^{n} x_{ij}^{k} = 1, \qquad \forall i \in N \quad (3.2)$$

$$\sum_{j=1}^{n} x_{0j}^{k} = 1, \qquad \forall k = 1 \dots M \quad (3.3)$$

$$\sum_{i=1}^{n} x_{i0}^{k} = 1, \qquad \forall k = 1 \dots M \quad (3.4)$$

$$\sum_{i=1}^{n} x_{ih}^{k} - \sum_{j=1}^{n} x_{hj}^{k} = 0, \qquad \forall h \in N, \forall k = 1 \dots M \quad (3.5)$$

$$\sum_{l=1}^{n} x_{li}^{k} z_{ij} - \sum_{p=1}^{n} x_{pj}^{k} z_{ij} = 0, \qquad \forall i, j \in N, \forall k = 1 \dots M \quad (3.6)$$

$$y_j \leq Q, \qquad \forall j \in N \quad (3.7)$$

$$x_{ij}^{k}(y_j - y_i - q_j) = 0, \qquad \forall i, j \in N, \forall k = 1 \dots M \quad (3.8)$$

$$x_{ij}^{k}(d_i + t_{i,j}) \leq a_j, \qquad \forall i, j \in N, \forall k = 1 \dots M \quad (3.9)$$

$$d_i = \max\{a_i, e_i\} + s_i, \qquad \forall i, \in N \quad (3.10)$$

$$a_i \leq l_i, \qquad \forall i, \in N \quad (3.11)$$

$$z_{ij}\, a_i \leq a_j, \qquad \forall i, j \in N \quad (3.12)$$

$$x_{i0}^{k}(d_i + t_{i,0}) \leq C, \qquad \forall i, \in N, \forall k = 1 \dots M \quad (3.13)$$

The constraints are interpreted as follows: Constraint 3.2 ensures that each location is visited exactly once, while constraints 3.3 and 3.4 ensure that each vehicle departs from and arrives at the depot. Constraint 3.5 ensures that if a vehicle arrives at a location then it must also depart from that location and constraint 3.6 ensures that the pickup and delivery of a request is carried out by exactly one vehicle. Constraints 3.7 and 3.8 together form the capacity constraints, that the total loads within a vehicle at any one time cannot exceed the maximum capacity. The time window constraints are ensured by 3.9, 3.10 and 3.11. Finally, the precedence constraint is ensured by 3.12 and the constraint on the maximum length of the scheduling horizon is ensured by 3.13.

The objective function, 3.14, is to minimise the total distance travelled over all routes subject to the following constraints.

$$\text{Minimise} \sum_{k=1}^{M} \sum_{i,j \in V} t_{ij} x_{ij}^{k} \tag{3.14}$$

The next section will provide details on the instances to be used in this research in order to evaluate the solutions obtained for the PDPTW.

3.3 PDPTW Instances of Li and Lim [2001]

The problem instances generated by Solomon [1987] for the VRPTW have widely become recognised as the standard instances for this problem type. The instances we will consider have 100 requests, travel times between locations are equal to the corresponding distances, and a homogeneous fleet is assumed. The data used to determine the requests' coordinates and demands are based on data from the standard set of routing test problems given in Christofides et al. [1979].

A description of the instances is as follows. There are 6 sets of instances each with a different length of the scheduling horizon. The geographical locations of requests vary with instance type and those which are randomly dispersed are generated from a random uniform distribution. The sets where instances have locations dispersed randomly are denoted R1 and R2, the sets where the instances have clustered locations are denoted C1 and C2, and finally, the sets where instances have semi-clustered locations are denoted by RC1 and RC2. The problem sets ending in a 1 have a short scheduling horizon and the time and route length constraints allow only a small number of requests to be serviced by the same vehicle. The problem sets ending in a 2 have a longer scheduling horizon. This, coupled with the larger vehicle capacities and requests

with wide time windows, permits many requests to be serviced by the same vehicle. A summary of this information is provided in Table 3.1.

		Number of instances	Vehicle capacity	Scheduling horizon	Service time	Location distribution	Time windows
Type 1	LC1	9	200	1236	90	Clustered	Narrow
	LR1	12	200	230	10	Random	Narrow
	LRC1	8	200	240	10	Mixed	Narrow
Type 2	LC2	8	700	3390	90	Clustered	Wide
	LR2	11	1000	1000	10	Random	Wide
	LRC2	8	1000	960	10	Mixed	Wide

Table 3.1: Summary Information for the VRPTW Instances of Solomon [1987]

A brief summary of how the data for these instances was generated is now provided, for more detailed information see Solomon [1987]. To develop both the random and partially clustered instances, first the percentage of requests to receive time windows was randomly generated. For this number of requests, a random permutation was generated and the time windows were assigned. The time windows for the remaining requests were then allocated a width equal to the scheduling horizon.

The method for the clustered instances was to first run a 3-*opt* heuristic on each cluster, to create routes, and then select an orientation for each route. A description of the 3-*opt* heuristic used here can be found in the literature review in Section 2.8.2. The time window constraints were generated by choosing the centre as the arrival time at each location; the width was then derived as above. This approach identified a very good, possibly optimal set of solutions.

The PDPTW instances that are to be explored in this research are generated by Li and Lim [2001] and extend the problem instances outlined above. These were created by randomly pairing the locations within routes in solutions obtained by Li et al. [2001]. This differs from the approach of Nanry and Barnes [2000] who took the 9 instances from the set LC1, whose best found solutions had since been proved optimal (see Nanry and Barnes [2000]), and paired up the locations appearing in the routes of the optimal solutions.

An advantage of the approach taken by Li and Lim [2001] is that they were not restricted to generating PDPTW instances using only the instances with a proved optimal solution for the VRPTW, as most of the Solomon [1987] instances have not yet been optimally solved. Another advantage is that the paired pickup and delivery locations

could be much more randomly dispersed in real-life problems, so they may not necessarily be paired up within the same route as were the VRPTW solutions.

All problem instances have 100 real locations, based on the 100 original requests for the VRPTW, with several additional 'dummy' locations, used for pairing up the locations in routes. The number of overall requests in each set is provided in Table 3.2. All other information is the same as in Table 3.1 for the case of the VRPTW.

lc101	53	lr101	53	lrc101	53	lc201	51	lr201	51	lrc201	51
lc102	53	lr102	55	lrc102	53	lc202	51	lr202	50	lrc202	51
lc103	52	lr103	52	lrc103	53	lc203	51	lr203	51	lrc203	51
lc104	53	lr104	52	lrc104	54	lc204	51	lr204	50	lrc204	51
lc105	53	lr105	53	lrc105	54	lc205	51	lr205	51	lrc205	51
lc106	53	lr106	52	lrc106	53	lc206	51	lr206	50	lrc206	51
lc107	53	lr107	52	lrc107	53	lc207	51	lr207	51	lrc207	51
lc108	53	lr108	50	lrc108	52	lc208	51	lr208	50	lrc208	51
lc109	53	lr109	53					lr209	51		
		lr110	52					lr210	51		
		lr111	54					lr211	50		
		lr112	53								

Table 3.2: Number of Requests for the Instances of Li and Lim [2001]

In our experiments comparisons with the best known results from the literature will be made to assess the quality of our algorithm. For our case, the objective will be to minimise the total distance travelled, as was the case for Pankratz [2005a] and Pankratz [2005b] for the dynamic variant of the problem. This will allow a direct comparison with the results achieved for the static PDPTW and the dynamic PDPTW in Chapter 6.

The results of Li and Lim [2001], Dergis and Dohmer [2008] and Ding et al. [2009] show that for the PDPTW, 54 of the 56 best known solutions, for the instances of Li and Lim [2001], are the same if choosing the objective of reducing the number of vehicles required, rather than the total distance travelled. Only in two cases can a solution be achieved with one fewer vehicles but with an increase to the total distance travelled. For the instances LC104 and LRC101, a solution has been found by Dergis and Dohmer [2008] and Ding et al. [2009] (and by Li and Lim [2001] for the case of LRC101), that uses one fewer vehicles with an increase to the total distance travelled.

Comparisons can therefore be made with the results of Li and Lim [2001], who apply

a prioritised objective function with the order: (1) minimise the number of vehicles; (2) minimise the total travel distance; (3) minimise the total schedule duration; and (4) minimise the total waiting time. Comparisons can also be made with Ding et al. [2009], whose objective is similar to this, although it does not include minimising the total schedule duration. Finally, comparisons can be made with the results of Dergis and Dohmer [2008], whose objective is to minimise the number of vehicles followed by minimising the total travel distance.

The remainder of this chapter examines methods previously discussed in the literature for the PDPTW, such as initial insertion heuristics and neighbourhood search operators.

3.4 Constructing Initial Solutions

Insertion heuristics for VRPs build feasible solutions by inserting, at each iteration, an un-routed request into a current partial route or into a new route. This process is performed either sequentially, one route at a time, or in parallel, where several routes are considered simultaneously. Two key questions are posed in the design of such methods, which request to select next for insertion and where to insert the request.

To construct an initial feasible solution, this research will investigate insertion heuristics based on those previously examined in the literature for the PDPTW (see Section 2.8.1). The main insertion heuristics applied to the PDPTW, which have produced competitive results in the literature, include those of Nanry and Barnes [2000], Li and Lim [2001], Pankratz [2005a], Lu and Dessouky [2006] and Hosny and Mumford [2009b]. These will be investigated further in this section.

The first method to be considered is a simple greedy heuristic applied by Nanry and Barnes [2000]. At each iteration it inserts the request, from all remaining requests, that evokes the lowest additional cost to the objective function. A request can be inserted into an existing route or into a new route and this process is performed until all requests are assigned.

A different approach was adopted by Li and Lim [2001] which builds routes sequentially. A route is first initialised with a request using the criteria of maximum combined farthest distances to depot, minimal combined latest bound of time windows and minimal combined period of time windows. No further details are provided by Li and Lim [2001] with regards to the weights of these criteria and how the cost of insertion for each request is calculated. The route is then greedily completed by adding, at each

iteration, the request from all remaining requests, which evokes the lowest additional cost to the objective function. This is performed until no further requests can be added to that route. A new route is then initialised and the process is repeated until all requests have been inserted.

The cost of reducing the slack time due to an insertion is considered in the insertion heuristics of Pankratz [2005b] and by Lu and Dessouky [2006]. The slack time at a location is referred to as the difference between the end of the time window, for service at that location, and the actual time that service takes place. For the case of the PDPTW both the time at the pickup location and delivery location need to be considered. For the case of Lu and Dessouky [2006] the authors aimed to investigate whether selecting an insertion which does not use as much of the available slack would leave more opportunity for future insertions. The slack insertion heuristic examined in this research is based on that applied by the H1 and H2 heuristics of Pankratz [2005b] for the dynamic variant of the problem. It first sorts the requests in the order of their slack time and then greedily inserts them into the solution, as above, in ascending order, meaning the most urgent requests are inserted first.

Hosny and Mumford [2009b] show that a simple sequential insertion heuristic, when paired with a metaheuristic approach, produces results comparable to SINTEF [2004]. The main difference between this method of insertion is that it does not try to find the best insertion position for each request in the route, but it accepts the first feasible insertion. This produces savings with regards to the computational time. The requests in this case were first sorted according to the distance from the depot to their delivery location and then inserted in descending order. This identifies that the results of the initial solution may not be critical, with regards to the final solutions achieved, when pairing with a more advanced approach.

Due to the observation of Hosny and Mumford [2009b], another insertion heuristic is investigated in this thesis which relaxes the constraints of the greedy heuristic to save on computational time, with the aim that the quality of the overall final solution is not lost. A similar procedure was applied by Pankratz [2005a] in order to generate initial solutions to create an initial population for their GA. The procedure selects requests to be inserted in a random order, starting with a single empty vehicle. For each selected request to be inserted, all feasible insertions in all existing routes of the current (partial) solution are examined. This is achieved by examining all possible insertion positions for the pickup and delivery locations in each route, taking into account the precedence and capacity constraints. Additionally, a new vehicle is allocated which is temporarily initialised with the selected request. Among all feasible insertions identified, the one that causes the minimal increase in the total distance travelled is chosen. This is

performed until all requests are assigned.

This method is outlined in Algorithms 1 and 2.

Algorithm 1 BestInsert (Set of Requests, Set of Routes)

```
1: Initialise LocalMin ← ∞
2: for (Set of all Requests) do
3:     Let r be the request
4:     for (Set of all Routes) do
5:         Let v be the route
6:         Let the pickup location of r be p
7:         Let the delivery location of r be d
8:         for (All feasible insertion positions of p in v) do
9:             Insert p in v
10:            for (All feasible insertion positions of d in v) do
11:                Insert d in v
12:                Calculate △cost */△cost is the change in solution cost due to the insertion/*
13:                if (△cost < LocalMin) then
14:                    LocalMin ← △cost
15:                    v_best ← v */v_best is the current best route of request r/*
16:                    r_best ← r */r_best is the current best request in route v/*
17:                    p_best ← p */p_best is the current best insertion of p/*
18:                    d_best ← d */d_best is the current best insertion of d/*
19: Insert request r_best in route v_best in positions p_best and d_best
```

Algorithm 2 RandomInsertion

```
1: Let M ← 0 */ M is the number of vehicles used */
2: Let s ← ∅ */ s is the partially constructed solution */
3: repeat
4:     for (Each unassigned request r, in a random order) do
5:         Initialise an empty vehicle v in s
6:         Run BestInsert ({r} , ∀ routes ∈ s)
7:         if v_best = M + 1 then
8:             M ← M + 1
9:         else
10:            Eliminate empty vehicle in s
11: until (All requests have been inserted)
```

The following section will investigate initial results for the insertion heuristics outlined above.

3.5 Results for the Insertion Heuristics

This section provides results for the initial insertion heuristics defined in Section 3.4. The insertion heuristics will be referred to as follows: the randomised heuristic of Pankratz [2005a] will be known as *random*, the method of Nanry and Barnes [2000] will be known as *greedy*, that of Li and Lim [2001] will be known as *max_dist*, that of Pankratz [2005b] will be known as *slack* and finally, that of Hosny and Mumford [2009b] will be known as *acc_first*. Results are compared using the instances of Li and Lim [2001] as outlined in Section 3.3.

Each of the insertion heuristics to be examined in this section are deterministic in nature accept the *random* method. To avoid bias towards the random heuristic, the results for *random* are the average solution cost achieved after 100 runs. For all other methods results are simply the solution cost found after a single run.

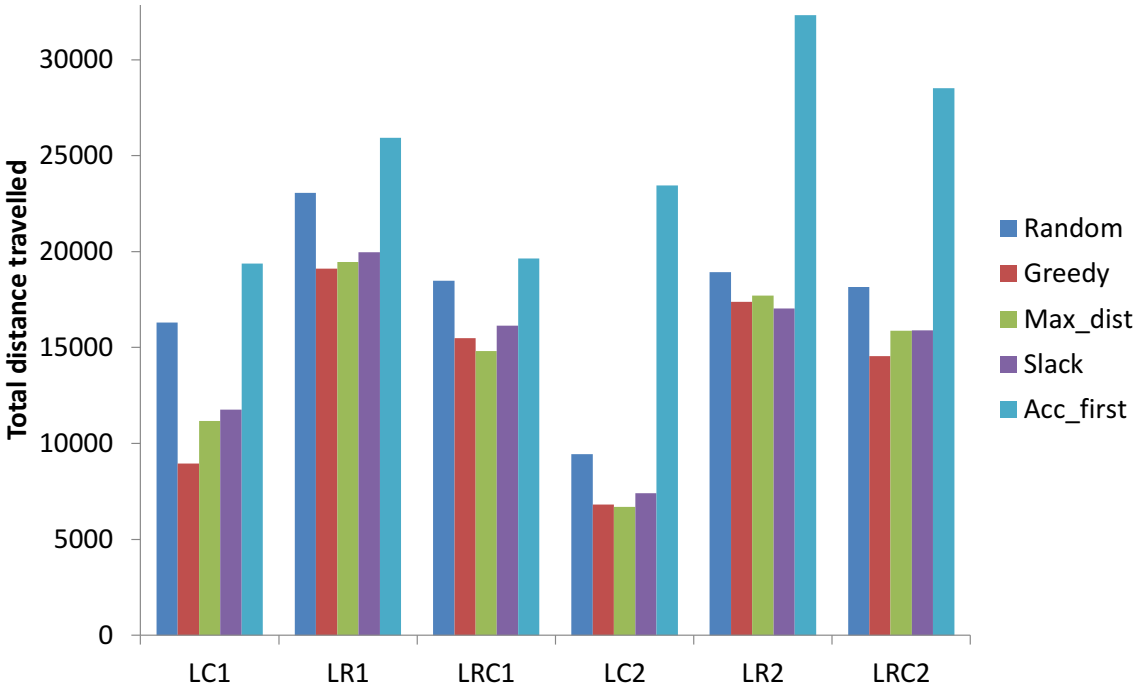


Figure 3.1: TD achieved by each of the Insertion Heuristics for each set of instances

Figure 3.1 shows the results of applying each of the 5 insertion heuristics, for each set of instances. The result for each set is the total distance travelled (TD) over all instances in that set and the figures are reported in Table 3.3. The total number of vehicles (NV) required over all instances in each set is also provided.

	Random		Greedy		Max_dist		Slack		Acc_first	
	TD	NV	TD	NV	TD	NV	TD	NV	TD	NV
LC1	16300.75	109	8943.85	91	11172.16	99	11762.23	100	19383.74	106
LR1	23052.16	185	19114.06	182	19457.74	190	19975.43	188	25924.26	214
LRC1	18484.47	128	15481.32	126	14822.67	127	16136.83	130	19647.10	139
LC2	9435.00	28	6816.62	34	6698.44	30	7400.02	30	23447.92	38
LR2	18931.75	47	17387.26	47	17715.32	42	17028.63	46	32322.19	58
LRC2	18157.14	39	14559.54	39	15871.32	38	15898.45	39	28523.06	49
Total	104361.27	536	82302.65	519	85737.65	526	88201.59	533	149248.27	604

Table 3.3: TD and NV achieved by each of the Insertion Heuristics for each set of instances

From the results shown in Table 3.3 and Figure 3.1 we can see that the insertion method that achieved the overall lowest initial solutions is the *greedy* method. As described previously, at each iteration the *greedy* method selects the feasible insertion position of a request, which results in the minimum increase in distance. For this case, it appears to generate solutions with a lower overall total travel distance. As expected, the results for *acc_first* were significantly higher, due to the fact that it does not try to find the best insertion position for each request in the route, but accepts any feasible insertion.

It was discussed in Section 3.3 that there is a direct link between the objectives of reducing the total travel distance and minimising the number of vehicles. From Table 3.3 it can be seen that it is again the *greedy* method that obtains the overall minimum number of vehicles required for the solutions achieved. These are however only initial solutions and require further improvement before comparisons should be made to the best known solutions. It is known from Hosny and Mumford [2009b] that the solutions are dramatically changed during the improvement phase; hence this should be investigated further before an insertion method is chosen as a basis for this research.

As the *random* method is the only non-deterministic insertion heuristic of those considered here, Table 3.4 and Table 3.5 provides summary statistics on the solutions achieved after 100 runs on each instance, over the 6 sets of instances. In Table 3.4 the minimum (Min) value is the average taken over all instances in the set of the minimum total travel distance achieved after 100 runs on each instance. The average (Avg) is the average taken over all instances in the set of the total travel distance achieved after 100 runs on each instance. The standard deviation (SD) is the average taken over all instances in the set of the standard deviations achieved after 100 runs on each instance. Finally, the coefficient of variation (CV) is the average taken over all instances in the

set of the coefficient of variations achieved after 100 runs on each instance. The average total travel distance over all instances in the set is reported for comparison with the SD and CV.

	Min	Avg	Max	SD	CV
LC1	1396.09	1811.19	2205.62	170.94	10%
LR1	1621.63	1921.01	2215.92	1441.99	75%
LRC1	1924.01	2310.56	2690.72	1275.64	55%
LC2	683.60	1179.37	1941.97	2141.55	186%
LR2	1363.95	1721.07	2036.36	1475.22	86%
LRC2	1760.94	2269.64	2771.16	1627.49	73%

Table 3.4: Summary Statistics for the Average TD achieved by the *Random* Insertion Heuristic for each set of instances

The CV is useful in interpreting the results here because the SD of data must always be understood in the context of the mean of the data. In contrast, the CV is a normalised measure of dispersion, as it is the ratio of the standard deviation to the mean. For comparison between instances with widely different means (as is the case here), we should look at the CV to make comparisons on the spread of the results. In this case it is stated as a percentage difference from the mean.

Table 3.4 shows a large variation in the solutions achieved for each instance by the *random* insertion heuristic. Looking at the average CV for each set, it is the clustered set, in particular those with a short scheduling horizon, which achieves the lowest variability of results. These are the instances, which are seen in the literature, as the easiest instances to solve and the solutions have been proven optimal (see Nanry and Barnes [2000]).

It is clear that by taking the minimum value obtained by the *random* heuristic that the results could be greatly improved. Table 3.5 investigates this further where the figures reported are now the total over all instances in the set and not the average, therefore are comparable with those in Table 3.3.

From Table 3.5 it is evident that if the minimum value had been taken after 100 runs of the heuristic, rather than the average value obtained, then the totals achieved would have been lower for each set of instances than any of the other methods as seen in Table 3.3. To investigate this approach further Table 3.6 provides information on the computational times of each of the 5 insertion heuristics, again by each set of instance.

The result is the average time it takes to compute an initial solution on an instance in that set. Therefore the result for the *random* method is the average time taken to complete 100 runs of the algorithm for each instance and for all other methods it is the average time taken to complete a single run on each of the instances.

	Min	Avg	Max
LC1	12564.81	16300.75	19850.56
LR1	19459.61	23052.16	26591.02
LRC1	15392.08	18484.47	21525.75
LC2	5468.77	9435.00	15535.78
LR2	15003.45	18931.75	22399.99
LRC2	14087.54	18157.14	22169.25
Total	81976.26	104361.27	128072.35

Table 3.5: Summary Statistics for the TD achieved by the *Random* Insertion Heuristic for each set of instances

As the *random* method is straightforward, the time taken to complete 100 runs is on average still less than a second, as seen in Table 3.6. From the evidence provided, continuing in this research we shall take the result from the *random* insertion method as the minimum achieved after 100 runs.

	Random	Greedy	Max_dist	Slack	Acc_first
LC1	0.27	0.06	0.02	0.01	0.01
LR1	0.21	0.05	0.02	0.01	0.01
LRC1	0.19	0.05	0.02	0.01	0.01
LC2	0.96	0.24	0.08	0.03	0.02
LR2	1.27	0.29	0.14	0.03	0.02
LRC2	0.89	0.22	0.09	0.02	0.02

Table 3.6: Average CT required by each of the Insertion Heuristic for each set of instances (seconds)

There is potential to add randomisation to the other insertion heuristics considered in this section and not just to the *greedy* heuristic. Preliminary results into four randomised variations of the above insertion methods can be found in Appendix A. Results show that of the randomised methods investigated, it is still the *random* heuristic applied so far in this chapter that achieves the most promising results. This supports its application further in this research.

The next section will introduce two neighbourhood operators from the literature, to improve on the initial solutions generated by the insertion heuristics. It will be investigated as to whether achieving a higher quality initial solution is significant in reducing the cost of the solution when applying an additional improvement heuristic. It is often the case in the PDPTW, as stated in Hosny and Mumford [2009a], that the initial solution is drastically changed during the improvement phase.

3.6 Neighbourhood Search Operators

A local search algorithm iteratively modifies a current solution by moving from one solution to another solution in its neighbourhood. For the case of the VRP, a neighbour of a current solution could differ by the insertion of a single request in another route. To attempt to improve on the initial solutions constructed in Section 3.4, a route improvement heuristic based on that of the λ-interchange generation mechanism (see Section 2.8.2), first applied to the VRP by Osman [1993], is introduced.

3.6.1 The *Shift* Operator

The *shift* operator denotes the shift of one request from one route to another. This is shown in Figure 3.2.

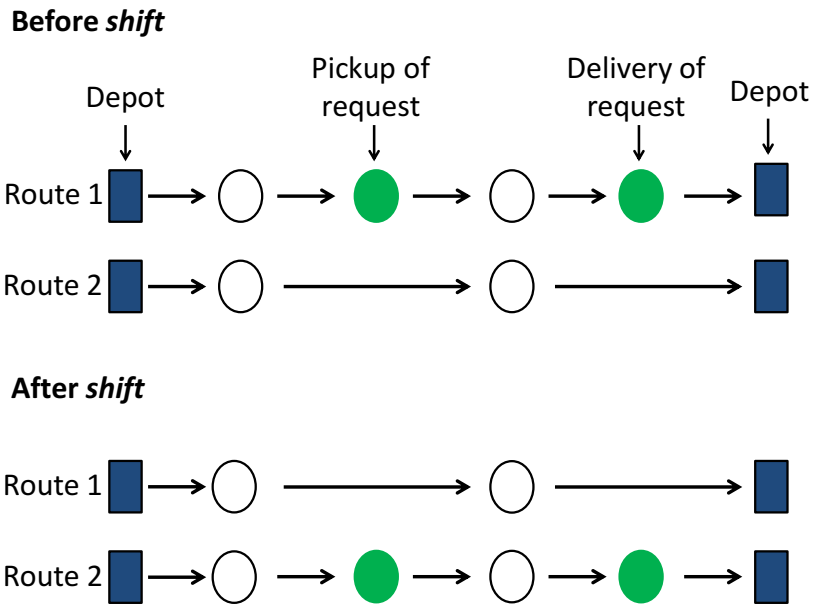


Figure 3.2: Shift Operator

If the first vehicle selected only contains one request, then this will result in an empty vehicle (a vehicle with no requests), hence this operator allows the number of vehicles to be reduced. This is an important property of the λ-interchange generation mechanism.

It needs to be determined which request is to be shifted. This could be a random request or the best move of all requests. Another decision to be made is where the request should be moved. One approach is to use a greedy method to examine all feasible insertions; another could be to accept the first feasible insertion position. Due to the results of preliminary investigations, the two most successful methods will be compared here.

A greedy *shift* operator will examine all feasible insertions of all requests in all other routes and then accept the one which yields the largest reduction in total travel distance to the solution. A random *shift* operator will select a request at random to be moved. If no feasible improving move involving this request is found, then another request is selected at random without replacement. Once a feasible improving move is found, all requests are again available for selection. The procedure is carried out until every request has been considered and no feasible improving move is found, i.e. no requests remain available to be selected. For the case of the greedy *shift*, the procedure is carried out until there remains no feasible improving move for any request.

It seems obvious that if every request is examined at each iteration, rather than examining one request at random, then greater improvement per iteration could be made. However, the added computational cost of this will need to be investigated.

3.6.2 The *Exchange* Operator

The *exchange* operator swaps a request from one route with a request of another and is demonstrated in Figure 3.3.

For this case, the requests chosen to be exchanged, requires further investigation. Preliminary investigations showed that examining every pair of requests at each iteration would involve a large amount of computational time. Therefore the method of choosing the first request to swap at random will be chosen. However, it still needs to be determined how the second request to swap should be chosen.

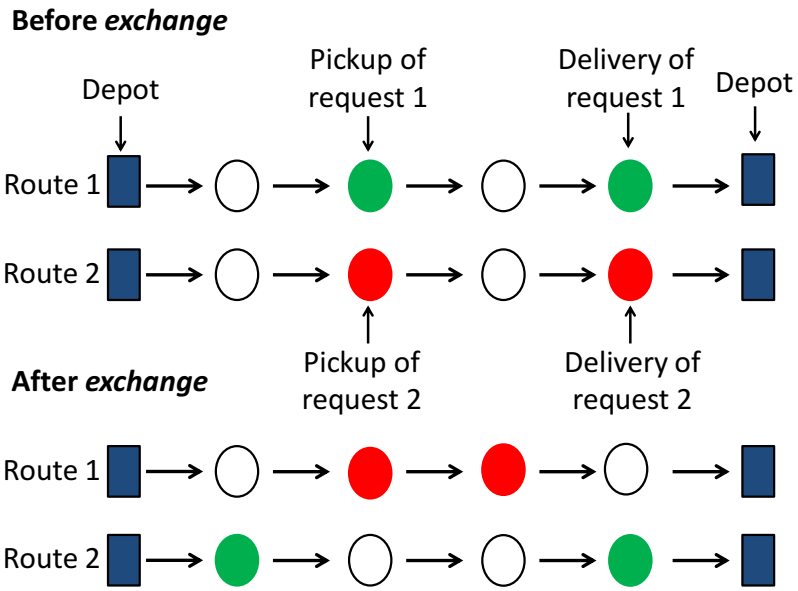


Figure 3.3: Exchange Operator

Investigating all feasible swaps with each remaining request could still be computationally expensive. Another option could be to also choose the second request at random. The criterion of selecting the first request at random and then greedily exchanging it in the solution, denoted part-random *exchange*, will be compared to choosing both requests at random, denoted random *exchange*.

The part random *exchange* operator will select the first request at random to be exchanged. If no feasible and improving swap with any other request in a different route is found, then the request is removed from those available to be swapped and another request is selected at random. Once a feasible improving swap is found, all requests are again available for selection. The procedure is carried out until there are no more requests available to be swapped, i.e. all requests have been examined and no feasible improving swap has been identified.

For the random *exchange*, the first request to be swapped is again selected at random from a first set of available requests. The second request is then selected at random from all available requests in a second set of requests. If a feasible improving swap is found, then the requests are swapped and all requests become available once more in both sets. If one is not found, then the second request is removed from those available in the second set and another second request is selected at random. This procedure is carried out until no requests remain available in the second set of requests. The request then selected first is removed from those available in the first set and another request is chosen at random. The procedure stops when there is no remaining request to be swapped in the first set, i.e. all requests have been removed from this set as no

feasible swap had been found.

The varying criteria for the neighbourhood operators are now investigated in the next section and results for each method are compared.

3.7 Determining Criteria for the Neighbourhood Operators

This section looks to investigate both the random and greedy *shift* operators, and the part-random and random *exchange* operators defined in Section 3.6. Both the total travel distance of the solutions achieved and the computational time will be compared in order to determine which features would be most suitable to proceed with in this research.

Results for total travel distance in the remainder of this section are the best solution found after 100 runs for each set of instances and for each of the 5 insertion methods. The computational times are the average time to complete 100 runs for each instance in the set.

Table 3.7 provides the results of applying the greedy *shift* operator with the part-random *exchange* operator and the computational times are provided in Table 3.8.

	Random	Greedy	Max_dist	Slack	Acc_first
LC1	7811.11	7975.42	8528.73	9277.96	8672.03
LR1	15 789.15	16 386.81	16 324.54	16 356.74	16 827.93
LRC1	12 189.09	12 542.08	12 629.24	12 982.36	13 076.54
LC2	5198.89	5829.71	5598.63	6252.14	7254.47
LR2	13 077.43	14 137.56	13 842.88	14 108.92	14 403.65
LRC2	11 316.01	11 969.33	12 180.64	12 322.51	12 539.35
Total	65 381.68	68 840.92	69 104.65	71 300.63	72 773.97

Table 3.7: TD achieved for the Greedy *Shift* and Random *Exchange* operators by each of the Insertion Heuristics and for each set of instances

	Random	Greedy	Max_dist	Slack	Acc_first
LC1	199.77	40.53	72.69	34.95	212.93
LR1	212.75	88.69	127.17	150.36	276.65
LRC1	183.63	90.02	91.16	112.34	213.46
LC2	537.13	170.76	262.62	364.35	1273.73
LR2	1432.95	1174.53	1094.88	535.35	3758.61
LRC2	991.91	466.70	772.41	619.03	2160.51

Table 3.8: Average CT required for the Greedy *Shift* and Random *Exchange* operators by each of the Insertion Heuristics and for each set of instances (seconds)

Table 3.9 provides the solutions when applying the random *shift* operator paired with the random *exchange* and the computational times for these solutions are provided in Table 3.10.

	Random	Greedy	Max_dist	Slack	Acc_first
LC1	7666.61	7892.64	7822.06	8070.78	7875.34
LR1	15 322.48	15 348.04	15 427.57	15 392.28	15 418.76
LRC1	11 688.76	11 621.45	12 039.97	12 082.86	11 855.22
LC2	5114.27	5605.43	5370.19	5481.52	5956.36
LR2	12 312.20	12 762.97	12 425.40	12 636.45	12 491.65
LRC2	10 711.71	10 886.69	10 790.89	10 985.64	10 743.94
Total	62 816.03	64 117.21	63 876.08	64 649.53	64 341.27

Table 3.9: TD achieved for the Random *Shift* and Part-Random *Exchange* operators by each of the Insertion Heuristics and for each set of instances

	Random	Greedy	Max_dist	Slack	Acc_first
LC1	32.81	19.14	25.88	28.47	49.58
LR1	13.56	14.91	12.70	13.63	13.81
LRC1	10.02	11.67	9.17	10.20	10.52
LC2	1709.40	751.90	986.42	2123.07	2553.68
LR2	8426.42	4483.59	9990.82	8671.04	6777.13
LRC2	1624.05	1695.35	1910.85	1752.23	1393.59

Table 3.10: Average CT required for the Random *Shift* and Part-Random *Exchange* operators by each of the Insertion Heuristics and for each set of instances (seconds)

The results provided in Table 3.11 are for the random *shift* operator applied with the random *exchange*. Table 3.12 provides the computational time required to achieve these results.

	Random	Greedy	Max_dist	Slack	Acc_first
LC1	7799.41	7892.72	7889.91	9007.61	8081.89
LR1	15 609.71	15 683.66	15 660.32	15 870.98	15 990.19
LRC1	11 999.79	11 991.69	12 322.76	12 446.95	12 377.09
LC2	5171.77	5678.75	5494.69	5876.32	6141.93
LR2	12 578.79	13 331.60	12 959.67	13 246.73	12 946.31
LRC2	10 903.35	11 178.72	11 092.29	11 404.29	11 215.62
Total	64 062.82	65 757.14	65 419.63	67 852.88	66 753.03

Table 3.11: TD achieved for the Random *Shift* and Random *Exchange* operators by each of the Insertion Heuristics and for each set of instances

	Random	Greedy	Max_dist	Slack	Acc_first
LC1	2.95	6.06	2.57	1.35	3.39
LR1	2.91	5.25	2.48	2.44	2.90
LRC1	2.32	4.78	2.01	2.08	2.50
LC2	22.07	32.84	25.97	21.88	26.49
LR2	88.50	107.65	109.36	34.13	83.69
LRC2	27.71	45.25	35.31	20.29	29.19

Table 3.12: Average CT required for the Random *Shift* and Random *Exchange* operators by each of the Insertion Heuristics and for each set of instances (seconds)

It can be seen by comparing the initial solutions (see Section 3.5), to the results achieved after the application of the neighbourhood operators (see Tables 3.7, 3.9 and 3.11), that the local search methods significantly decrease the total distance travelled over all the instances.

Comparing Tables 3.7 and 3.11, it can be seen that the random *shift* operator improves on the results of the greedy *shift* operator for all sets of instances and for all of the insertion methods. Tables 3.8 and 3.12 show a significant increase in computational time of the greedy *shift* operator compared to the random *shift* operator. On average the computational time increases by over 37 times. Therefore, the random *shift* operator will be chosen over the greedy *shift* for further use in this research.

The results in Tables 3.10 and 3.12 show that the computational time of searching each exchange for a random request for instances with a longer scheduling horizon is dramatically increased compared to selecting both requests at random. The greatly increased computational time could be because these instances have a wider scheduling horizon and wider time windows, therefore there is a larger search space of possible solutions to be explored. There is however an improvement in total travel distance for the results when applying the part-random *exchange* method compared to the random *exchange* as seen when comparing Tables 3.9 and 3.11.

This method is therefore not appropriate for use in our research due to computational time being an important factor. Therefore applying both the greedy *shift* operator and the part-random *exchange* operator will not be considered.

Algorithm 3 SHIFT

```
1: Start from a solution s
2: Let S be the set of all requests
3: repeat
4:     for (Each request r ∈ S, in a random order) do
5:         Let M be the number of vehicles in s
6:         Remove r from s
7:         Run BestInsert ({r} , ∀ routes ∈ s)
8:         Let new solution be s'
9:         if s' is better than s then
10:            s ← s'
11:            Reset S to be the set of all requests
12:        else
13:            Remove r from S
14: until (S = ∅)
```

The random *shift* operator to be studied further in this research is therefore outlined in Algorithm 3 and the random *exchange* operator to be studied further in this research is outlined in Algorithm 4.

Algorithm 4 EXCHANGE

```
1: Start from a solution s
2: Let S_1 be the set of all requests
3: repeat
4:     for (Each request r ∈ S_1, in a random order) do
5:         Let current vehicle of r_1 be v_1
6:         Remove r_1 from s
7:         Feasible=False
8:         Let S_2 be the set of all requests
9:         while (Feasible=False & S_2 ≠ ∅) do
10:            Choose a request r_2 at random from S_2
11:            Let current vehicle of r_2 be v_2
12:            if (v_1 ≠ v_2) then
13:                Remove r_2 from s
14:                Run BestInsert ({r_1} , {v_2})
15:                Run BestInsert ({r_2} , {v_1})
16:                Let new solution be s'
17:                if s' is better than s then
18:                    s ← s'
19:                    Feasible ← True
20:            if (Feasible=False) then
21:                Remove r_2 from S_2
22:        if (Feasible=True) then
23:            Reset S_1 to be the set of all requests
24:        else
25:            Remove r_1 from set S_1
26: until (S_1 = ∅)
```

The next section will provide results for the operators compared to the best known results in the literature.

3.8 Results for the Neighbourhood Operators

This section will provide results for the initial insertion heuristics discussed in Section 3.4 combined with the neighbourhood operators introduced in Section 3.6. When applying the local search methods, the *shift* operator and then the *exchange* operator are applied to the initial solution until no further improvement can be made.

It is clear that the addition of the neighbourhood operators results in an increase in computational time, however this is dramatically increased for the case of the instances with a longer scheduling horizon, as seen in Table 3.12. This increase however is lowest for the case of the *random* heuristic. The *random* heuristic also achieves the lowest overall minimum total travel distance, as seen in Table 3.11.

Summary statistics for the results provided in Table 3.11, when applying the random

shift operator and the random *exchange* operator are now provided. Table 3.13 provides the average total distance achieved after 100 runs on each instance, averaged over each instance in the set. Table 3.14 contains the average standard deviation (SD) for 100 runs over each instance in the set and the average coefficient of variation (CV) for 100 runs over each instance in the set.

	Random	Greedy	Max_dist	Slack	Acc_first
LC1	997.26	907.98	929.44	1073.32	1171.61
LR1	1434.66	1394.60	1403.98	1419.73	1453.95
LRC1	1689.78	1609.61	1637.06	1675.67	1720.99
LC2	824.79	757.35	715.10	816.49	944.94
LR2	1330.93	1335.88	1315.61	1338.53	1334.19
LRC2	1609.41	1527.95	1557.14	1624.06	1606.25

Table 3.13: Average TD achieved by each of the Insertion Heuristics and the Neighbourhood Operators for each set of instances

From the results in Table 3.13 it can be seen that the average results achieved by the *random* heuristic for each set of instances are greater overall than the *greedy* method and the *max _dist* method. However, looking at the CV for each instance and insertion method, provided in Table 3.14, these two methods provide the lowest variation in solutions achieved over 100 runs. Results achieved on average are therefore lower, but the minimum values obtained by the *random* method are not achieved.

	Random		Greedy		Max_dist		Slack		Acc_first	
	SD	CV	SD	CV	SD	CV	SD	CV	SD	CV
LC1	115.90	12%	22.42	2%	37.12	4%	36.85	4%	182.83	13%
LR1	68.32	5%	34.35	2%	39.12	3%	42.79	3%	62.04	4%
LRC1	88.91	5%	47.38	3%	41.47	3%	60.20	4%	87.38	5%
LC2	90.66	11%	20.15	3%	15.73	2%	31.57	4%	95.49	10%
LR2	84.10	6%	49.07	4%	55.88	4%	51.77	4%	76.78	6%
LRC2	122.88	8%	62.07	4%	76.03	5%	78.59	5%	94.46	6%

Table 3.14: Average SD and CV achieved by each of the Insertion Heuristics and the Neighbourhood Operators for each set of instances

As the *random* insertion heuristic achieved the most promising results, it will be adopted as the method of insertion to generate an initial feasible solution for the

PDPTW in this research. The process of applying the neighbourhood operators to the initial solution generated from the *random* insertion method is defined in Algorithm 5.

Algorithm 5 InitialAlgorithm

```
1: Initialise s_best to an arbitrary large value
2: for (i = 0; i < 100; i++) do
3:     Run RandomInsertion
4:     Let initial solution be s
5:     Let Improve = TRUE
6:     while (Improve = TRUE) do
7:         Run Shift
8:         Run Exchange
9:         Let new solution be s'
10:        if (cost(s') < cost(s)) then
11:            s ← s'
12:        else(Improve=False)
13:    if (cost(s) < cost(s_best)) then
14:        s_best ← s {where s_best is the current best solution}
```

Table 3.15 shows the results achieved by applying Algorithm 5, compared to the minimum results found in the literature for minimising the total travel distance. Again the total travel distance is represented by TD and the number of vehicles by NV. The NV used in the best known solution is based on the objective of minimising the total travel distance. As stated in Section 3.3, for a small number of cases there exists a solution with a lower number of vehicles than that stated in Table 3.15, but this results in an increase to the total travel distance.

	Best known		Our result		% Increase	
	TD	NV	TD	NV	TD	NV
LC1	7445.42	90	7799.41	91	5%	1%
LR1	14635.41	143	15609.71	162	7%	13%
LRC1	11088.34	93	11999.79	107	8%	15%
LC2	4713.26	24	5171.77	27	10%	13%
LR2	10652	31	12578.79	45	18%	45%
LRC2	9064.98	26	10903.35	41	20%	58%
Total	57599.41	407	64062.82	473	11%	16%

Table 3.15: Comparison of TD and NV achieved by the InitialAlgorithm to the Best Known solutions from the literature

In total, 4 out of 56 of the best found solutions are achieved: 2 of these from the set LC1 and 2 from the set LC2. This confirms the belief that the solutions of the

clustered instances are easier to solve than those with random locations (Nanry and Barnes [2000]). For the set LC1 the solutions achieved only require 1 more vehicle when compared to that of the best known solutions and for LC2, there is an increase of only 3 vehicles on the best known. This also suggests that minimising the number of vehicles is comparable to minimising the total travel distance.

On average the results were ≈11% above the best known solutions, which indicates further room for improvement. This is particularly the case for the problems with randomly located requests and a longer scheduling horizon; these appear to be the most challenging for our algorithm. They have both solutions which have a significant increase in total travel distance and in the number of vehicles required by the solutions.

The likely reason is that the solution space for these problems seems to be larger, due to the randomness of the locations and the larger width of time windows. This means there are many more feasible insertion positions for the requests, so a larger number of feasible solutions to be explored. This will need to be considered when applying more sophisticated heuristics in the following chapter.

The next section will look to improve on the neighbourhood operators by considering reconstruction heuristics; these will also attempt to reduce the number of vehicles in the solutions.

3.9 Reconstruction Heuristics

It can be seen from the results in Section 3.8 that there is scope for further improvements to be made in the solutions achieved after the neighbourhood operators. The main disadvantage of the current *shift* and *exchange* operators is that they attempt to insert a request, into a route, without making any change to the current ordering of the locations already within that route. Naturally, a higher proportion of neighbourhood moves will be seen to retain feasibility, if the existing ordering in a route can also be changed, though of course this will bring additional computational time. Therefore we propose four different reconstruction heuristics.

Two types of method can be identified for improving a solution: *intra-route* moves and *inter-route* moves. *Intra-route* moves act on the requests in a single route, whereas *inter-route* moves act on several routes simultaneously. The two neighbourhood operators examined so far perform *inter-route* moves between requests of different routes. The first of the four reconstruction heuristics introduced looks at performing *intra-route* moves, i.e. moving a request within a route.

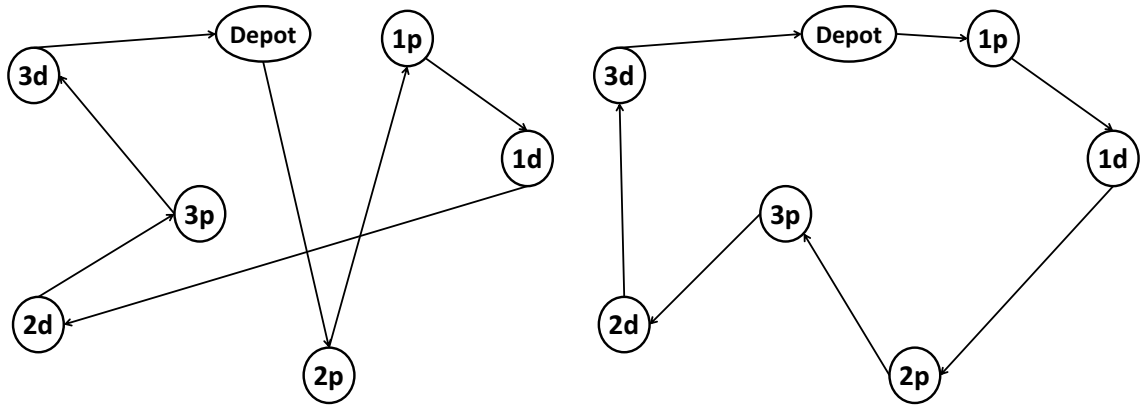
3.9.1 Single Move within a Route

The *single move* within a route operator examines each route individually. It looks to remove a request from a route and re-insert it into a different position in the same route, by either making a change to the pickup location, the delivery location or both. Figure 3.4 shows an example of how the reconstruction of a single move of one request within a route can decrease the total travel distance of that route. By removing the pickup and delivery locations of request 2, they can then be re-inserted into new positions that improve the solution. This method is based on *Or*-opt exchanges (see Or [1976] and Section 2.8.2) but is adapted to the PDPTW. Nanry and Barnes [2000], Li and Lim [2001] and Lau and Liang [2001] apply a variation of this neighbourhood operator along with the 2 previous operators in their local search based algorithms.

Following the criteria for insertion consistent with the *shift* and *exchange* operators, the request to be removed is selected at random. The request is removed from its route and the heuristic attempts to insert both the pickup and delivery locations of the request in all other feasible positions within that route. Only improving moves are accepted and if more than one exists, the insertion position which amounts to the largest reduction in total travel distance is accepted. The full procedure is outlined in Algorithm 6.

Algorithm 6 SINGLEMOVE

```
1: Start from a solution s
2: Let M be the number of vehicles in s
3: for (v ← 1 to M) do
4:     Let S be the set of all requests assigned to vehicle v
5:     repeat
6:         Let v_best be the current best solution of route v
7:         Choose a request r at random from {S}
8:         Remove r from v
9:         Run BESTINSERT ({r} , {v})
10:        Let new solution of route v be v'
11:        if (v' is better than v_best) then
12:            v' ← v
13:            Reset S to be the set of all requests originally in v
14:        else
15:            Remove r from S
16:    until ({S} = ∅)
```



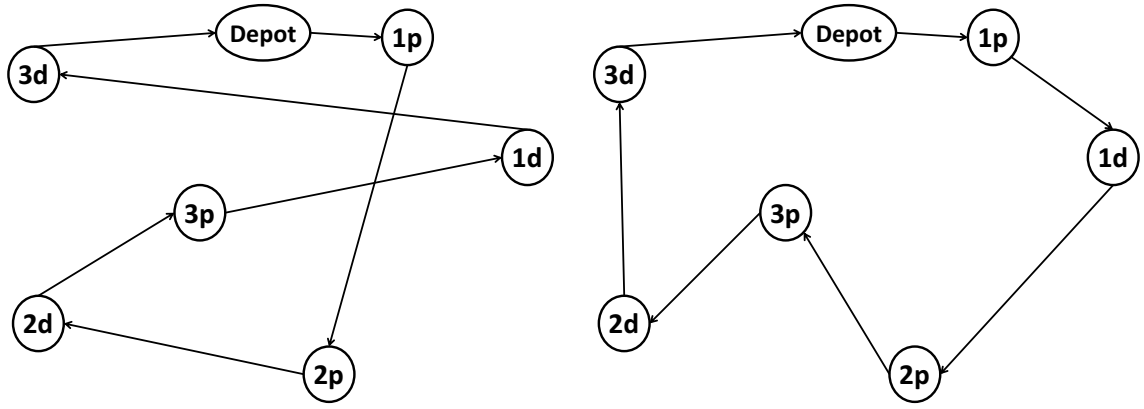
(a) Before reconstruction (b) After reconstruction

Figure 3.4: Single Move within a Route Reconstruction

Summary results are provided in Tables 3.16 and 3.17 at the end of the section. The next operator to be considered looks at re-ordering the requests of an entire route.

3.9.2 Single Route Reconstruction

The *single route* reconstruction operator attempts to re-order the locations of an entire route. An example of this is shown in Figure 3.5 where removing a single request from the route would not have achieved the best found solution, suggesting a more destructive method is needed.



(a) Before reconstruction (b) After reconstruction

Figure 3.5: Single Route Reconstruction

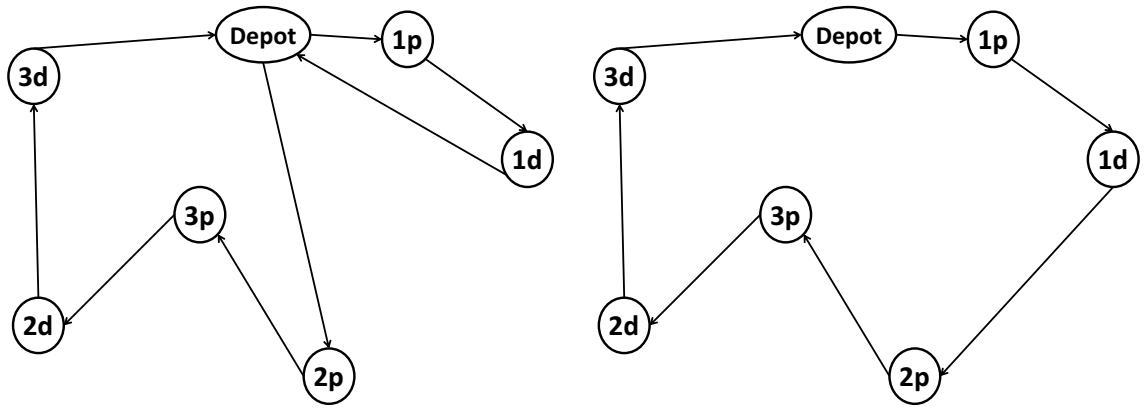
The *single route* reconstruction operator first removes all requests from a single route and attempts to re-insert them based on a given criteria. The criterion to be adopted here, which preliminary results showed to be the most promising, is by allocating the request whose pickup and delivery locations are the maximum distance from the depot first and then each of the remaining requests greedily. This is performed on each route individually. The procedure for this operator is outlined in more detail in Algorithm 7. Summary results are again provided in Tables 3.16 and 3.17 and discussed at the end of the section.

Algorithm 7 SINGLEROUTE

```
1: Let s be the current best solution
2: Let M be the number of vehicles in s
3: for (v ← 1 to M) do
4:     Let S be the set of all requests in v
5:     Let v_globalmin be the current best solution of route v
6:     Remove all requests from v
7:     Choose a request r from S based on maximum combined distance from depot
8:     Insert r into v
9:     Remove r from S
10:    Let v_best ← ∅ be the current partial solution of route v
11:    repeat
12:        Feasible = False
13:        for (Each request r in S) do
14:            Run BESTINSERT ({r} , {v})
15:            Let new solution of route v be v'
16:            if (Feasible = False) then
17:                v_best ← v'
18:                Feasible = True
19:            else
20:                if (v' is better than v_best) then
21:                    v_best ← v'
22:            Remove r from v
23:            if (Feasible = True) then
24:                v' ← v_best
25:                v_best ← ∅
26:                Remove r from S
27:            else
28:                S ← ∅
29:    until (S ← ∅)
30:    if (All requests have been re-inserted) then
31:        if (v' is better than v_globalmin) then
32:            v_globalmin ← v'
33:            Update s
```

3.9.3 Multiple Route Reconstructions

The *multiple route* reconstruction operator attempts to re-construct multiple routes simultaneously. An example of this is shown in Figure 3.6 where two routes are merged into one. Preliminary investigations show that two *multiple route* operators were effective in reducing the total travel distance of the solutions. The first of these, *double route*, takes two existing routes with the aim of constructing two new routes. All requests are removed from both the routes and each route is initialised with a single request. The first route is initialised with the request which is the maximum combined distance from the depot. The second route is initialised with the request which is the maximum combined distance from the pickup location of the first request. The routes are then constructed simultaneously using a greedy heuristic which at each iteration inserts the request, from all remaining requests, that evokes the lowest additional



(a) Before reconstruction (b) After reconstruction

Figure 3.6: Multiple Route Reconstruction

The results are provided in Table 3.16 and Table 3.17 provides the added computational time of applying the *double route* operator to the neighbourhood operators, *single move* and *single route* operators.

Algorithm 8 DoubleRoute

```
1: Let s be the current best solution
2: Let M be the number of vehicles in s
3: for (v_1 ← 1 to M − 1) do
4:     for (v_2 ← v_1 + 1 to M) do
5:         Let S_1 be the set of all requests in v_1 and v_2
6:         Let v_globalmin be the current total travel distance of routes v_1 and v_2
7:         Remove all requests from v_1 and v_2
8:         Choose a request r_1 from S_1 based on maximum distance from depot
9:         Insert r_1 into v_1, remove r_1 from S_1
10:        Choose a request r_2 from S_1 based on maximum distance from r_1
11:        Insert r_2 into v_2, remove r_2 from S_1
12:        Let v_best = ∅ be the current best solution of route v
13:        repeat
14:            Feasible = False
15:            for (All requests in S_1) do
16:                Let r be the current request
17:                for (Each vehicle v_1 and v_2) do
18:                    Let v be the current vehicle
19:                    Run BestInsert ({r} , {v})
20:                    Let new cost of v_1 and v_2 be v'
21:                    if (Feasible = False) then
22:                        v_best ← v'
23:                        Feasible = True
24:                    else
25:                        if (v' is better than v_best) then
26:                            v_best ← v'
27:                    Remove r from v
28:                    if (Feasible = True) then
29:                        v' ← v_best
30:                        v_best ← ∅
31:                        Remove r from S_1
32:                    else
33:                        S_1 ← ∅
34:        until (S_1 = ∅)
35:        if (All requests have been re-inserted) then
36:            if (v' is better than v_globalmin) then
37:                v_globalmin ← v'
38:                Update s
```

The second case *triple route*, is carried out on 3 routes, with the aim of reducing the number of routes to 2. This is only applied on a combination of routes if at least one of the routes is an outlier with regards to the number of requests present in that route. The aim is to decrease the computational time needed to search every combination of 3 routes and preliminary results showed the operator is only successful for these cases. It was found that after taking the mean and standard deviation of the number of requests within each route in a solution, if for one route the number of requests present was more than one standard deviation less than the mean; then this would be classed as an

outlier. This route would then be chosen to be reconstructed along with a combination of any two other routes. This process is outlined in Algorithm 9.

Algorithm 9 TripleRoute

```
1: if (∃ an outlier in the size of a route) then
2:     Let v_c be the chosen route
3:     Let s be the current best solution
4:     Initialise set {S_1} with the requests of v_c
5:     Run DoubleRoute (For all v_1&v_2, where v_1 ≠ v_2 ≠ v_c until {S_1} = ∅)
6:     if (All requests have been inserted) then
7:         Let new solution be s'
8:         if (s' is better than s) then
9:             s ← s'
10:            M ← M − 1
```

A summary of the results provided for each of the reconstruction heuristics above is provided in Table 3.16. Case 1 provides the results for the two neighbourhood operators, *shift* and *exchange*, outlined in Section 3.7. Case 2 provides the results for the neighbourhood operators applied with the *single move* operator outlined above. Case 3 provides the results for the neighbourhood operators applied with the *single move* and *single route* operators outlined above. Case 4 provides the results for the neighbourhood operators applied with the *single move*, *single route* and *double route* operators outlined above. Finally, Case 5 provides the results for the neighbourhood operators applied with the *single move*, *single route*, *double route* and *triple route* operators. The results are the minimum value obtained after 100 runs of the operators on each instance and are provided for each set of instances.

	Case 1	Case 2	Case 3	Case 4	Case 5
LC1	7799.41	7446.25	7445.41	7445.41	7445.41
LR1	15609.71	15011.00	15050.57	14900.78	14756.23
LRC1	11999.79	11516.82	11474.83	11283.84	11220.30
LC2	5171.77	4775.42	4778.29	4781.63	4781.63
LR2	12578.79	11354.01	11197.53	11146.56	11149.73
LRC2	10903.35	9656.99	9617.33	9487.50	9434.88
Total	64062.82	59760.49	59563.95	59045.72	58788.17

Table 3.16: TD achieved by the Neighbourhood Operators and each Case of the Reconstruction Heuristics for each set of instances

Table 3.16 shows that for each of the added reconstruction operators, there is a decrease in the total distance travelled of the solutions. It is shown that by adding the *single*

move operator to the neighbourhood operators defined in Section 3.8 that an overall average improvement of $\approx 6.7\%$ is achieved. This improvement is greatest in the case of the LRC2 set of instances at over 11%.

The results show that adding the *single route* operator to both the neighbourhood operators from Section 3.8 and the *single move* operator, that a small further improvement of $\approx 0.3\%$ is achieved. A further improvement of $\approx 0.9\%$ can be be made to the solutions by also including the *double route* operator and a further improvement $\approx 0.4\%$ can be made by including the *triple route* operator.

Overall the largest decrease is achieved in the LRC2 set of instances where by applying all of the reconstruction operators, a reduction of over 13% is achieved from the results obtained by the neighbourhood operators alone. The added computational time of applying these operators is now investigated.

	Case 1	Case 2	Case 3	Case 4	Case 5
LC1	2.95	3.73	3.81	11.08	21.34
LR1	2.91	3.57	3.68	8.97	17.21
LRC1	2.32	3.06	3.13	9.45	17.49
LC2	22.07	37.52	42.29	95.88	156.46
LR2	88.50	140.39	150.38	207.54	279.21
LRC2	27.71	43.12	45.26	94.79	148.64

Table 3.17: Average CT by the Neighbourhood Operators and each Case of the Reconstruction Heuristics for each set of instances

Table 3.17 shows the computational time added by applying the reconstruction heuristics outlined above, these are again the average time required to complete 100 runs of the algorithm for each instance. By applying all of the reconstruction heuristics there is a near five-fold increase in the computational time required. The *single move* operator alone increases the computational time by 44% and the *single move* and *single recon* operators together increase the computational time by 53%. However, it is the *multiple recon* operators that have the largest effect on the computational time - the *double recon* increased the computational time by over 2.5 times; however it is clearly the *triple recon* which gives the largest increase.

It is clear that these reconstruction operators decrease the total distance travelled in the solutions, but they require increased amounts of computational time. It is known from preliminary results that adding similar reconstruction operators to the algorithm would further improve the results, but again at a significant increase to the computational

time. There is still an improvement to be made to the solutions obtained by our algorithm; however more advanced methods may need to be considered to improve the solutions further in a reasonable amount of computational time.

The next section will provide the overall results that are achieved by applying these reconstruction heuristics to the previous neighbourhood operators and comparisons will be made to the best known solutions.

3.10 Summary of Results

Comparisons are now be made with the results achieved for the random insertion heuristic paired with the two neighbourhood operators and the 4 reconstruction heuristics, against the best results reported in the literature. The procedure is outlined in Algorithm 10.

Algorithm 10 HeuristicAlgorithm

```
1: Initialise s_best to an arbitrary large value
2: for (i = 1; i < 100; i++) do
3:     Run RandomInsertion
4:     Let initial solution be s
5:     Let Improve = TRUE
6:     while (Improve = TRUE) do
7:         Run Shift
8:         Run Exchange
9:         Run DoubleRoute
10:        Run TripleRoute
11:        Run SingleRoute
12:        Run SingleMove
13:        Let new solution be s'
14:        if (cost(s') < cost(s)) then
15:            s ← s'
16:        else
17:            Improve=FALSE
18:    if (cost(s) < cost(s_best)) then
19:        s_best ← s {where s_best is the current best solution}
20:    Next i
```

	Best known		Our algorithm		% Increase	
	TD	NV	TD	NV	TD	NV
LC1	7445.42	90	7445.41	90	0%	0%
LR1	14 635.41	143	14 756.23	147	1%	3%
LRC1	11 088.34	93	11 220.30	97	1%	4%
LC2	4713.26	24	4781.63	24	1%	0%
LR2	10 652	31	11 149.73	40	5%	29%
LRC2	9064.98	26	9434.88	33	4%	27%
Total	57 599.41	407	58 788.17	431	2%	6%

Table 3.18: Comparison of TD and NV achieved by the HeuristicAlgorithm to the Best Known solutions from the literature

Table 3.18 compares the results achieved by Algorithm 10 to the best known results found in the literature. Once again TD is the total distance travelled over all instances in the set and NV is the total number of vehicles used for all instances in the set. On average our results are ≈2% above the best known solutions in terms of total distance travelled. With regards to the best known number of vehicles, these figures are the best known number of vehicles when the objective is to minimise the total travel distance and not the number of vehicles.

In total, 26 out of 56 of the best known solutions are achieved. All of the best known solutions are achieved for the instances in the LC1 set, and 4 out of the 8 are achieved for the LC2 set. The minimum number of vehicles was also obtained in both of these cases. This confirms the general observations made in the literature that the solutions of the clustered instances are easier to solve than those with random locations (Nanry and Barnes [2000]).

For the instances with a short scheduling horizon, 22 out of the 29 best known solutions are achieved, but for the instances with a longer scheduling horizon only 4 of the best known solutions out of a total of 27 were found. This confirms that for the instances with a longer scheduling horizon, the best known solutions are more difficult to achieve than those with a short scheduling horizon.

The results show once again that it is the instances with randomly located requests and a longer scheduling horizon that are the most challenging. None of the best known solutions are achieved in the LR2 and LRC2 sets. The number of vehicles required in the solutions for these sets also greatly increased from those in the best known solutions. This further establishes the link between minimising the total distance and

the number of vehicles required.

The results indicate that further improvement can again still be made to the solutions, particularly in the instances with a longer scheduling horizon that have randomly dispersed locations. The heuristic methods investigated so far provide good results; however, it is thought it would require a dramatic increase in the computational time to improve the solutions further. It is therefore suggested that a more advanced method, such as a metaheuristic, should be considered to improve the solutions further in a reasonable amount of computational time.

3.11 Chapter Summary

In this chapter the PDPTW has been formally introduced and well-known instances for this variant of the problem have been reviewed. To first start the process of determining good quality solutions for this variant of the problem, insertion heuristics adapted from methods in the literature were examined. Section 3.6 then introduced two neighbourhood operators previously studied by Li and Lim [2001] which were modified in accordance with the aim of minimising computational time.

It is clear that the neighbourhood operators have a significant effect on reducing the overall cost of the solutions for each instance. However, preliminary results showed that examining all requests at each stage of the improvement phase resulted in significant increases in computational time which are not realistic, especially not in a dynamic environment. The adapted neighbourhood search operators applied to a *random* insertion heuristic are able to achieve 4 of the 56 best known solutions with an average increase in cost over all solutions being $\approx 11\%$.

To further improve on the results, 4 reconstruction heuristics have been introduced in Section 3.9. The final results achieve 26 of the best found solutions, and on average the results were $\approx 2\%$ above the best known solutions.

Therefore it is shown that the heuristic methods investigated are capable of finding high quality solutions in a reasonable amount of computational time. However, there are still improvements to be made, in particular in the instances with randomly located requests with a longer scheduling horizon.

The next chapter will therefore look to apply more advanced metaheuristic approaches to our algorithm in the hope of further improving the solutions in a reasonable amount of computational time.

Chapter 4

Further Methods for the PDPTW

4.1 Introduction

The aim of this chapter is to further improve the results achieved in Chapter 3 for the PDPTW. This is achieved through the introduction of metaheuristic approaches. The promising results achieved by metaheuristic approaches applied to the PDPTW, are outlined in the literature review in Chapter 2. An example of a metaheuristic applied successfully to the PDPTW is that of tabu search (see Section 2.8.3.1), this is also the most common improvement heuristic applied in a dynamic environment; we will therefore look to apply this in our research.

The rest of the chapter is outlined as follows. Section 4.2 introduces tabu search and Section 4.3 provides an investigation into the parameters to be adopted by the tabu search heuristic to be applied in this research. This includes the tabu tenure, stopping criteria and tabu attribute, which all need to be decided upon. Section 4.4 summarises the most promising results achieved. Improvements are then made to the solutions by means of a branch and bound heuristic, adapted from the LNS of Bent and Van Hentenryck [2006], defined in Section 4.5.

Section 4.6 identifies ways in which the computational time required by our algorithm can be improved. Once again, as this research aims to apply the PDPTW algorithm developed to a dynamic PDPTW, the algorithm will need to be suitable for use in a real-time environment. Finally, Section 4.7 outlines the final algorithm for the PDPTW and comparisons are made with the best known solutions from the literature. The chapter is concluded in Section 4.8.

4.2 Tabu Search Heuristic

Tabu search is a metaheuristic approach which was first introduced by Glover [1986] and has been used to solve many variants of the VRP (see Bräysy and Gendreau [2002], Cordeau and Laporte [2003] and Gendreau et al. [1999]). A full description of a tabu search is provided in Section 2.8.3.1 along with a review of the literature for adapting a tabu search heuristic to the PDPTW.

The tabu search heuristic in its simplest form introduced by Glover [1989], can be summarised as follows:

Algorithm 11 TabuSearchGlover

```
1: Start from a solution s
2: Let s_best ← s {where s_best is the current best solution}
3: Set the tabu list to ∅
4: while (Stopping criteria is not met) do
5:     Generate the neighbourhood of s through non tabu moves
6:     (or tabu moves that lead to solutions that improve s_best)
7:     Select the best solution s'
8:     if (s' is better than s_best) then
9:         s_best ← s'
10:    s ← s'
11:    Update the tabu list
```

To improve on the solutions achieved in Chapter 3 a tabu search heuristic is to be added to the *shift* operator defined in Section 3.7. Due to the large neighbourhood of the *exchange* operator, it is thought it would be too costly to apply the tabu search heuristic to this operator, with regards to the added computational time. At present the reconstruction heuristics also require a large amount of computational time, compared to the *shift* operator and the potential for improvement via these heuristics is thought to be less extensive.

Correctly determining a tabu tenure, a stopping criteria and the tabu attribute are all key decisions and will contribute to the efficiency and success of the tabu search heuristic. Based on the literature for the PDPTW, a tabu tenure and the maximum number of iterations without improvement to the best known solution both proportional to the number of requests to be serviced is common, see Nanry and Barnes [2000].

The tabu attribute chosen by Nanry and Barnes [2000], when applied to a similar *shift* operator for the PDPTW, consists of the request number and its position in the solution (i.e. its route and the position in that route). This approach covers both direct and indirect tabu moves, which are described below.

For the approach of Li and Lim [2001] (a tabu-embedded simulated annealing algo-

rithm), an eigenvalue structure is used to represent a solution and identify if it has been achieved before. This includes the number of vehicles, the total travel distance, the total schedule duration and the total waiting time for the vehicles in the solution. It is assumed that the probability that two different solutions will have the same eigenvalue is very small; hence it is reasonable to assume two solutions are the same if they share the same eigenvalue.

The approach of Montané and Galvão [2006], for the VRP with simultaneous pickup and delivery provides promising results. Here, both the edges removed and inserted within a solution are recorded in the tabu list. For example, if removing a request from a particular route results in an arrangement of locations remaining in that route which are 'tabu', then the move is indirectly classed as a 'tabu' move. This method will be investigated further for use in our research due to its encouraging results.

Adapting this attribute to the PDPTW, the edges inserted into a solution are classed as the 'direct' edges. These are the edges connecting the locations either side of the new insertions i.e. the locations before and after the insertion of the pickup location and the delivery location of a request. The edges removed from the solution are the 'indirect' edges, i.e. the edges that connected locations before and after the pickup and delivery location and that have been removed.

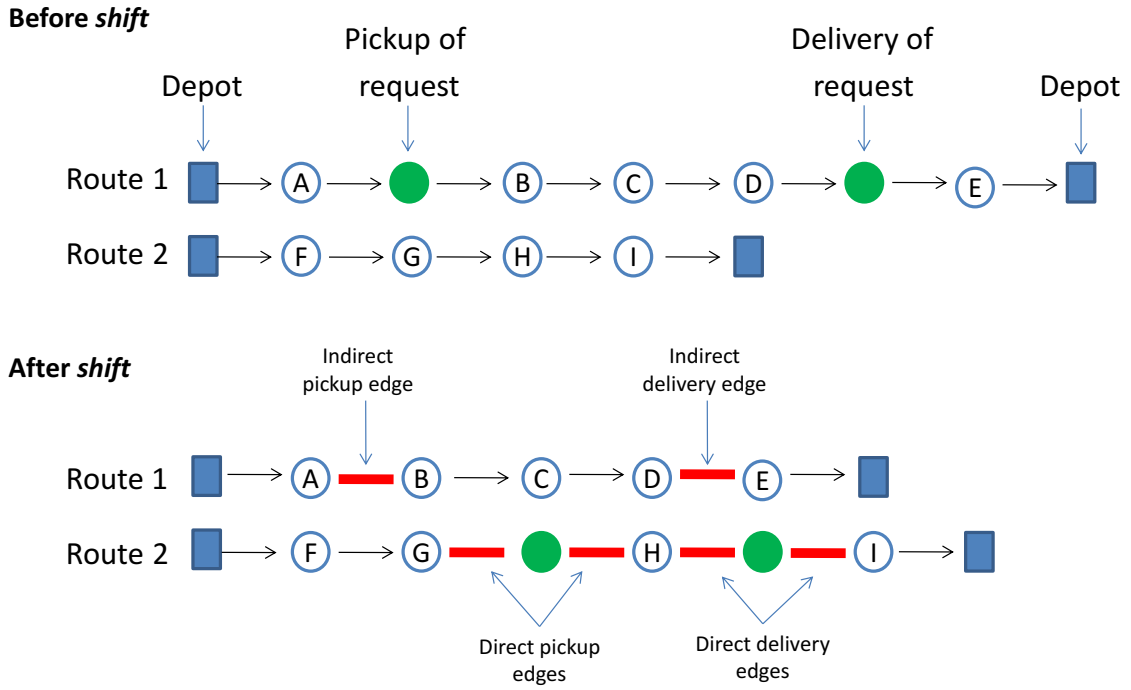


Figure 4.1: Tabu Attributes

The corresponding edges relating to the pickup location, the delivery location, or both,

can be stored in the tabu list. It will need to be determined whether to consider the direct edges, the indirect edges, or both. The varying tabu attributes are shown in Figure 4.1.

The next section will examine varying tabu tenures, cycle lengths and tabu attributes for applying a tabu search heuristic to the *shift* operator introduced in Section 3.7.

4.3 Determining Parameters

A tabu search heuristic is to be added to the random *shift* operator defined in Section 3.7, which follows the general guidelines provided in Glover [1989]. The stopping criteria chosen is based on achieving a maximum number of iterations without improvement to the best found solution or if there exists no more feasible moves to be made. To determine the tabu tenure and the maximum number of iterations without an improvement to the best found solution which achieve the most promising results, a range of values were selected and analysed. These are based on those suggested in the literature, namely those proportional to the number of requests (see Nanry and Barnes [2000]).

Using the approach of recording edges removed or inserted into the solution, the attributes to be stored within the tabu list were investigated using a total of 6 scenarios. The first scenario records only the direct pickup edge (D - P), the second scenario records only the direct delivery edge (D - D) and the third, records both the direct pickup and delivery edges (D-PD). The fourth scenario records the direct and indirect pickup edges (DI - P) and the fifth, the direct and indirect delivery edges (DI - D). The final scenario records the direct and indirect pickup and delivery edges (DI - PD).

Each of these approaches will now be explained in more detail using Figure 4.1. For the case of the direct pickup edges, these are the edges created when inserting the pickup location into the new route. The edges which are recorded are therefore the edge connecting location G to the pickup location of the request and the edge connecting this to location H. The direct delivery edge follows the same pattern being the edge connecting location H to the delivery location and the edge connecting this to location I. For the indirect edges this would be the edge connecting location A to location B and connecting location D to location E.

For the scenarios we are to investigate, each of these edges will need to be classed as 'tabu' for a move to be disallowed. For the case of the DI-PD this would include all of the edges removed and inserted into the solution for both the pickup and delivery

location. An advantage of this method means that a move can be made whereby a request is inserted into the same position for the pickup location as previously but into a different position for the delivery location. It is thought this will be beneficial for the dynamic problem where parts of the solution will be fixed during the scheduling horizon and there may be no other feasible pickup position but multiple opportunities to improve the position for the delivery location.

A summary of findings for determining the parameters can be found in the remainder of this section. For all cases the initial solutions were generated using the *random* insertion heuristic found to provide the most promising results in Section 3.8. The *exchange* operator previously defined in Section 3.7 and the reconstruction heuristics defined in Section 3.9 are also included. The procedure is therefore similar to that of Algorithm 10, whereby, the tabu search heuristic is applied to the random *shift* operator.

'No tabu' in the following tables is a comparison with the results achieved by Algorithm 10 provided in Section 3.10 where the tabu search heuristic is not added. The number of requests is represented by r and the number of services by s. Other tabu tenures and maximum numbers of iterations (max iterations), i.e. the number of iterations without improvement to the best found solution before the search is stopped, were explored with the summary findings representing the most promising of these. Tables 4.1 and 4.2 provide the initial results.

	No tabu	D-P	D-D	D-PD	DI-P	DI-D	DI-PD
LC1	7445.41	7457.23	7484.33	7489.00	7466.57	7455.84	7490.99
LR1	14756.23	15414.70	15513.21	15324.23	15437.18	15290.19	15316.60
LRC1	11220.30	11635.30	11501.84	11561.21	11429.82	11484.59	11624.57
LC2	4781.63	4961.14	5059.25	5055.25	5094.78	5050.55	5070.43
LR2	11149.73	12499.03	12462.85	12564.40	12320.16	12365.54	12362.39
LRC2	9434.88	10185.84	10381.38	10382.66	10216.78	10193.64	10218.11
Total	58788.17	62153.24	62402.86	62376.75	61965.29	61840.35	62083.09

Table 4.1: TD achieved for a Tabu Tenure = Max Iterations = r for applying the Random *shift*, by each tabu attribute and for each set of instances

	No tabu	D-P	D-D	D-PD	DI-P	DI-D	DI-PD
LC1	7445.41	7492.59	7484.38	7527.80	7463.96	7495.98	7484.80
LR1	14756.23	15367.51	15405.19	15330.54	15383.32	15469.75	15298.85
LRC1	11220.30	11611.62	11431.48	11410.66	11454.20	11561.30	11557.31
LC2	4781.63	5021.28	5102.72	5027.59	5081.47	5064.78	5061.84
LR2	11149.73	12456.69	12438.59	12406.47	12346.38	12420.70	12390.68
LRC2	9434.88	10127.58	10145.80	10176.55	10272.43	10339.08	10256.71
Total	58788.17	62077.26	62008.15	61879.60	62001.76	62351.58	62050.19

Table 4.2: TD achieved for a Tabu Tenure = r and Max Iterations = s when applying the Random *shift*, by each tabu attribute and for each set of instances

From Tables 4.1 and 4.2 it can be seen it is the tabu attribute of recording the direct and indirect delivery edges that provides the most promising results of the tabu algorithms. However, the results show that the introduction of the tabu search heuristic to the random *shift* operator does not improve on the results achieved without tabu search.

A possible reason for this might be due to the restrictions on the random shift operator. At present, a request is selected at random and the feasible move which results in a solution with the minimum total distance is accepted. If no feasible non-tabu moves or tabu-moves that improve the best found solution are identified, then another request is chosen without replacement. The search is therefore restricted to searching only the feasible moves of a single request at each iteration and there may actually be a limited number of these available for each request during the search, due to the time window and precedence constraints.

This may mean that the search is unable to escape from a local minimum under the current conditions as there may be either no feasible moves that exist in the current neighbourhood of the operator which would allow the search to do this, or because these solutions cannot be reached under the present tabu tenure or stopping criteria. Therefore the tabu search heuristic in its present form is not appropriate for use in this research.

Another variant of the *shift* operator was also considered in Section 3.6, a greedy *shift* operator, which applies the criterion of steepest descent when selecting a request to be moved. For this case, the best feasible improving move of all requests is selected at each iteration. It was shown in the results in Section 3.7 that this operator did not improve on the results of the random *shift* operator and required an increase in computational time. It was therefore discarded at this stage.

It will now be investigated to see if applying the tabu search heuristic to the *greedy* shift operator, can improve on the results, without a significant increase in the computational time required by the algorithm. The results are provided in Tables 4.3 and 4.4. The approach is once again similar to that outlined in Algorithm 10, this time replacing the random *shift* operator with the tabu search heuristic applied to the greedy *shift* operator.

	No tabu	D-P	D-D	D-PD	DI-P	DI-D	DI-PD
LC1	7445.41	7445.41	7445.41	7445.41	7445.41	7445.41	7445.41
LR1	14756.23	14641.91	14641.91	14645.45	14641.91	14641.91	14641.91
LRC1	11220.30	11119.25	11088.87	11119.25	11088.87	11088.34	11088.34
LC2	4781.63	4766.45	4766.45	4728.67	4766.45	4728.09	4718.87
LR2	11149.73	10880.54	10747.27	10859.31	10735.29	10766.20	10773.95
LRC2	9434.88	9210.04	9211.24	9228.24	9147.08	9133.70	9198.01
Total	58788.17	58063.59	57901.14	58026.32	57825.00	57803.64	57866.47

Table 4.3: TD achieved for a Tabu Tenure = Max Iterations = r when applying the Greedy *shift*, by each tabu attribute and for each set of instances

Tables 4.3 and 4.4 show that applying the tabu search heuristic to the greedy *shift* operator improves on the previous results achieved in Section 3.10. Also it is the tabu attributes of recording the direct and indirect pickup edges (DI - P) and the direct and indirect delivery edges (DI - D) that achieve the greatest improvement in total distance travelled and a stopping criterion equal to the number of services was most promising.

The added computational time of this approach needs to also be considered, since it was established in Section 3.7 that the time required by the greedy *shift* operator was greatly increased compared to that of the random *shift* operator. Increasing the maximum number of iterations appears to improve the results; however this also needs to be considered in terms of the added computational time it requires. Determining the appropriate tabu lengths, the maximum number of iterations and the tabu attribute will therefore continue.

	No tabu	D-P	D-D	D-PD	DI-P	DI-D	DI-PD
LC1	7445.41	7445.41	7445.41	77445.41	7445.41	7445.41	7445.41
LR1	14756.23	14641.91	14641.91	14641.91	14641.91	14641.91	14641.91
LRC1	11220.30	11088.87	11088.87	11089.67	11088.87	11088.34	11088.34
LC2	4781.63	4766.45	4766.45	4728.67	4725.03	4719.44	4766.45
LR2	11149.73	10786.62	10735.02	10784.39	10734.26	10736.27	10758.93
LRC2	9434.88	9104.10	9184.98	9176.81	9094.89	9131.27	9148.37
Total	58788.17	57833.34	57862.63	57866.85	57730.35	57762.63	57849.40

Table 4.4: TD achieved for a Tabu Tenure = r and Max Iterations = s when applying the Greedy *shift*, by each tabu attribute and for each set of instances

The next section will highlight the instances where the best known solution has not been achieved and will summarise the increase in computational time.

4.4 Results for Determining Parameters

Following the investigations for determining the tabu tenure, maximum iterations and tabu attribute, this section will explore the 4 cases that achieved the best results in more detail. Table 4.5 summarises the 4 cases where DI - P represents the recording of direct and indirect pickup edges and DI - D represents the recording of direct and indirect delivery edges once again.

	Tabu tenure	Maximum iterations	Tabu attribute
Case 1	No. requests	No. requests	DI - P
Case 2	No. requests	No. requests	DI - D
Case 3	No. requests	No. locations	DI - P
Case 4	No. requests	No. locations	DI - D

Table 4.5: The Tabu Search Heuristic Parameters for the best 4 Cases

To investigate each of these cases further, the results achieved will be compared to the best known solutions from the literature. Results will be analysed for the 20 instances, out of a total of 56, where the best found solution is not achieved in every case. The percentage difference in the total distance travelled compared to the best known solutions, for each of these instances and each case, are shown in Figure 4.2.

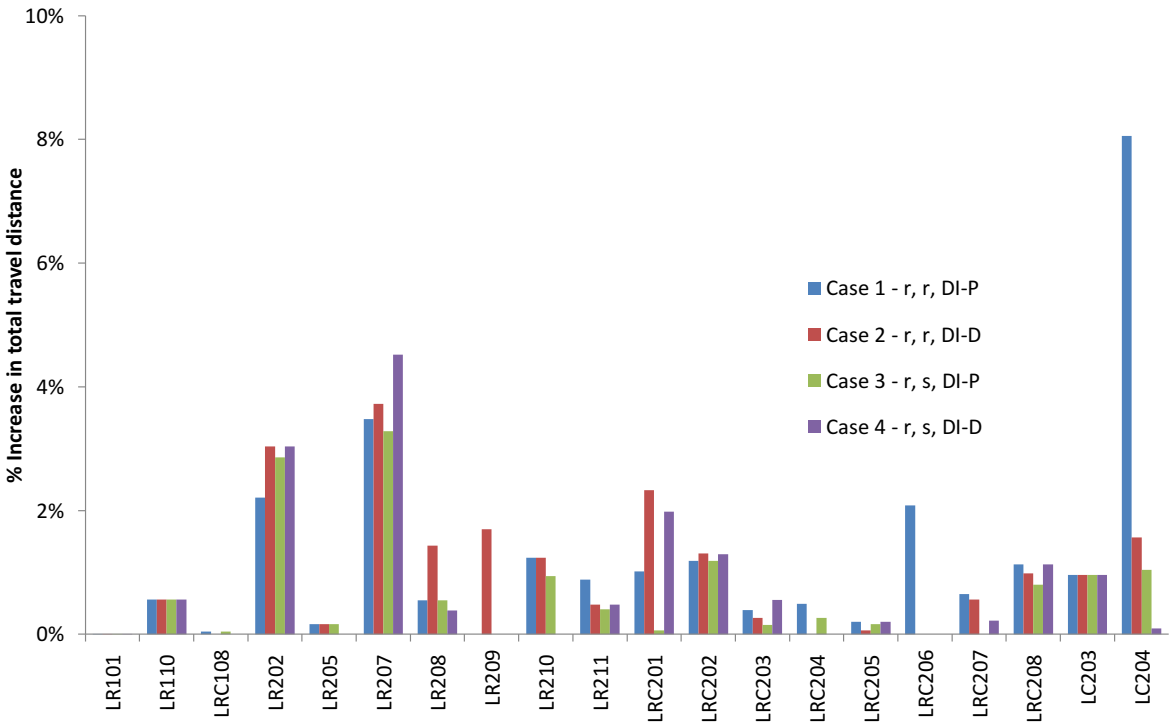


Figure 4.2: Percentage increase in TD from the Best Known solutions, by each Case for the worst instances

The results in Figure 4.2 show that the maximum percentage increase from the best known solution for all of the instances is $\approx 8\%$ (Case 1, instance LC204). However, for the other 3 cases the percentage increase for this instance is less than 2%. It can be seen that the results for each of the 4 cases differs over the set of instances. For some of the instances the best known solution has only not been achieved for one of the cases. For 12 of the instances the best found solution has not been achieved in any case, identifying that further improvement can still be made.

Table 4.6 provides the difference in the total distance travelled in the best found solutions for each case compared to the best known results provided in the literature. From the results provided in Table 4.6 it can be seen that it is Case 3 which achieves the overall lowest difference in total travel distance. In total, solutions are a distance of 130.94 from the best known solutions ($\approx 0.02\%$), with the best known solution achieved in 39 out of the 56 instances. However, 42 out of 56 best known solutions are achieved in Case 4, but with a total increase in distance of 163.24 ($\approx 0.03\%$). For Cases 1 and 2 the total increase in distance is 225.63 ($\approx 0.39\%$) and 204.26 ($\approx 0.35\%$) respectively, with 37 and 39 best known solutions achieved.

	Case 1	Case 2	Case 3	Case 4
LR101	0.02	0.02	0.02	0.02
LR110	6.48	6.48	6.48	6.48
LRC108	0.53	0.00	0.53	0.00
LR202	26.42	36.36	34.24	36.36
LR205	1.75	1.75	1.75	0.00
LR207	31.40	33.64	29.65	40.82
LR208	4.02	10.52	4.02	2.84
LR209	0.00	15.77	0.00	0.00
LR210	11.91	11.91	9.05	0.00
LR211	7.81	4.25	3.56	4.25
LRC201	14.26	32.73	0.84	27.86
LRC202	16.29	17.95	16.29	17.80
LRC203	4.27	2.87	1.66	6.04
LRC204	4.02	0.00	2.15	0.00
LRC205	2.61	0.84	2.11	2.61
LRC206	24.13	0.00	0.00	0.00
LRC207	6.89	5.94	0.00	2.35
LRC208	9.63	8.39	6.82	9.63
LC203	5.61	5.61	5.61	5.61
LC204	47.58	9.23	6.16	0.57
Total	225.63	204.26	130.94	163.24

Table 4.6: Difference in TD from the Best Known solutions after application of the Tabu Search Heuristic, by each Case for the worst instances

It is clear from the results provided in Section 3.7 that adding the greedy *shift* operator significantly increases the computational time of the algorithm. The computational times for each case are summarised in Table 4.7. The result is the total time taken to complete the algorithm with 100 iterations for each instance. 'No tabu' again refers to the results achieved by applying Algorithm 10, as detailed in Section 3.10, where the tabu search heuristic is not applied.

	No tabu	Case 1	Case 2	Case 3	Case 4
LR101	3.60	5.79	5.31	6.05	5.54
LR110	17.37	102.46	102.49	163.10	155.19
LRC108	25.15	108.52	104.98	125.02	120.15
LR202	65.95	290.52	284.70	531.48	491.08
LR205	111.90	390.54	393.38	672.16	647.96
LR207	399.72	1331.22	1315.92	2276.94	2315.90
LR208	1085.15	2410.61	2336.47	3765.21	3881.86
LR209	172.98	633.82	619.92	1070.48	1098.75
LR210	140.32	619.75	571.22	1078.94	1020.91
LR211	304.94	1287.33	1322.33	2230.51	2174.70
LRC201	58.95	92.24	93.34	137.73	128.71
LRC202	79.99	259.04	248.00	452.51	440.95
LRC203	148.68	678.11	668.76	1143.07	1158.29
LRC204	401.15	1258.22	1178.83	2082.19	2106.67
LRC205	73.54	194.52	192.73	314.70	305.27
LRC206	116.52	336.16	340.88	545.89	534.93
LRC207	156.25	576.29	558.54	1102.30	1005.98
LRC208	221.56	1094.78	1162.74	2060.35	1948.25
LC203	177.55	344.75	318.88	439.72	410.98
LC204	337.13	905.22	1000.34	1083.12	1114.71
Total	4098.41	12 919.88	12 819.76	21 281.48	21 066.77

Table 4.7: CT required after application of the Tabu Search Heuristic, by each Case for the worst instances (seconds)

From the results it can be seen that there has been an approximate three-fold increase in computational time for Cases 1 and 2. For Cases 3 and 4 the computational time has increased by over 5 times. The increase is greatest again for the instances with a longer scheduling horizon since, as we are already aware, these are the most difficult to solve. In particular, for the LR208 instance for Case 4, it took over an hour to complete 100 runs of the algorithm, with each run being on average 39 seconds long. This added computational time needs to be evaluated in context with the reduction in total travel distance achieved by the solutions and also compared with what is observed in the literature. This is investigated further in Section 4.6.

For the instance LRC201, a different minimum total travel distance is achieved for each of the 4 cases. This will now be investigated at each iteration of the tabu search

heuristic. This will now be investigated at each iteration of the tabu search heuristic. As stated previously in the section, the procedure applied is similar to that of Algorithm 10, whereby the Tabu Search Heuristic replaces the original shift operator. Therefore the tabu search heuristic followed by the *exchange* operator and the reconstruction heuristics are applied repeatedly until no further improvement can be made.

Figure 4.3 shows the total travel distance after each iteration of the tabu search heuristic, where an iteration is a single move, for each of the 4 cases. The minimum solution obtained for each case is obtained from a different initial solution, hence the different starting distances on the graph.

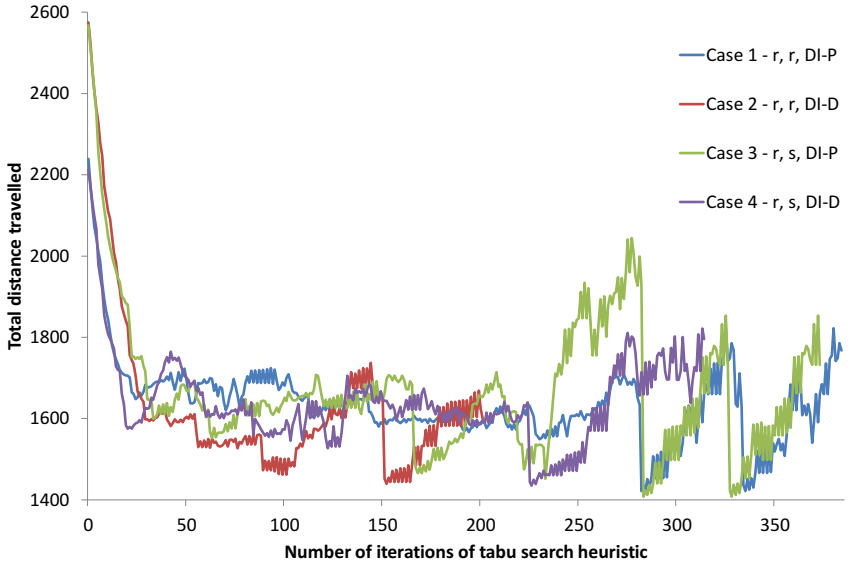


Figure 4.3: Change in Total Travel Distance at each iteration of the Tabu Search Heuristic for instance LRC201

The change in total travel distance shown in Figure 4.3 is that achieved at each iteration of the tabu search heuristic which is re-started multiple times throughout the search. Each time the algorithm has completed this heuristic it moves to the *exchange* operator and then the reconstruction heuristics where, if an improvement is achieved, the tabu search heuristic is applied again. The changes made by the other operators can therefore be identified by the vertical drops in the total travel distance at varying points on the graph. These indicate the final solution achieved by the tabu search heuristic at the end of one run of the heuristic and the starting solution at the beginning of the following run, after the other operators have been applied.

Figure 4.3 shows that Case 2 executes the minimum number of iterations of the tabu search heuristic and at each individual re-start of the procedure the number of iterations

performed is fewer. Cases 1 and 3 perform almost double the number of iterations as Case 2. It can be identified, through the peaks of the line representing Case 3, that the total distance travelled has increased more than for the other cases. This suggests that a significant alteration has been made to the solution during the search and results in the reconstruction heuristics achieving a greater improvement to the solution than had previously been possible. For this example, the solution has been able to escape from a local minimum and results in the lowest overall total travel distance achieved of the 4 cases.

To better understand how each case is able to produce varying solutions, the number of iterations without an improvement to the best found solution, recorded throughout the search, will be investigated for each case. The number of iterations without improvement starts at zero and increases to the maximum, as defined for each case in Table 4.5. If no feasible move is found, the number of iterations is set to the stopping criteria. This can be seen on the graph by a vertical straight line.

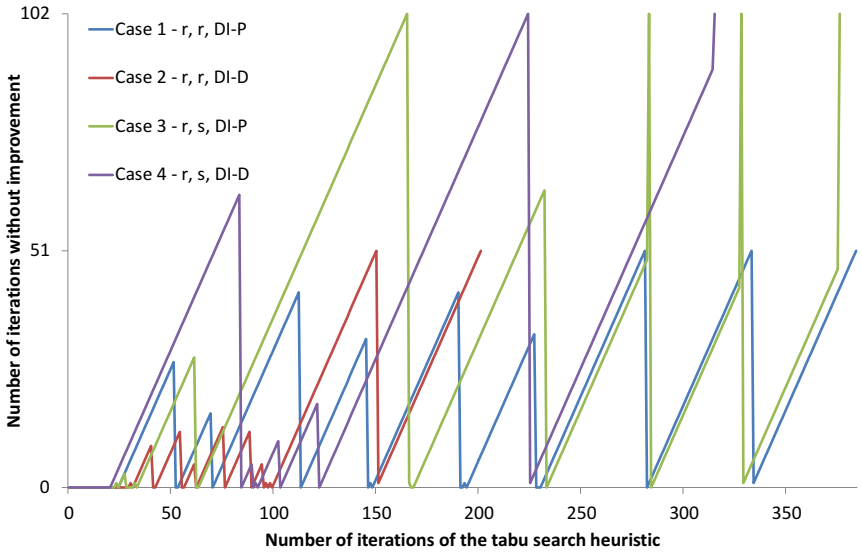


Figure 4.4: Number of iterations without improvement to the best known solution for instance LRC201

From Figure 4.4 it can be seen that, for Case 3 and 4, the search reaches a point where no more feasible moves can be made. This is during the latter part of the overall procedure. However, for Cases 1 and 2, where the maximum number of iterations is less, the search is stopped after completing r (being 51 in this instance) iterations without an improvement to the best found solution. This indicates that a stopping criteria with a higher maximum number of iterations may not be needed towards the end of the search. This could help in limiting the computational time of the algorithm.

The next section introduces the final improvement method for our algorithm, an adapted branch and bound heuristic.

4.5 Branch and Bound Heuristic

Branch and bound is a general algorithm used for finding optimal solutions to various problems in combinatorial optimisation. It consists of a logical enumeration of all feasible solutions, where large subsets of solutions can be discarded, by using information on upper and lower bounds.

To further improve our algorithm a method based on that of the large neighbourhood search (LNS) of Shaw [1998], extended by Bent and Van Hentenryck [2006] for the PDPTW, will be applied. The main idea behind LNS is to iteratively remove subsets of requests from the best found solutions and explore their feasible re-insertion positions in a systematic way. The method of Bent and Van Hentenryck [2006] first chooses a set of locations to be removed from a route according to a 'relatedness criterion'. This consists of first removing a request from its route at random. Then at each following iteration, the request which is most related to those which have previously been removed, is removed. The re-insertion is performed using a branch and bound procedure, with the limit on the bound set to the cost of the solution before the locations were removed.

For the instances with a longer scheduling horizon, due to the limited number of vehicles, the problem becomes more focused on finding the best ordering of requests to a route rather than the allocation of requests to routes. With the LR2 and LRC2 instances this becomes increasingly difficult as locations are randomly dispersed and, from the results provided in Section 4.4, it is clear that this is where improvement to our algorithm can still be made. A method which specifically focuses on optimising large portions of routes, such as a branch and bound heuristic could therefore prove successful in improving the solutions for these instances.

The adaptation of this method to our algorithm therefore involves removing routes or sub-sections of routes and applying a branch and bound heuristic to improve the ordering of the locations within that sub-section. For our algorithm each route is taken in turn and each section of the route is analysed. Routes are divided into overlapping sub-sections to ensure locations located closely to one another are considered in the same sub-section.

As branch and bound is an exact approach it can be computationally expensive. Pre-

liminary results suggest it can be applied to routes with up to 14 locations with a minimal increase in computational time to the overall algorithm (the computational times of this method will be summarised at the end of the section). In cases where there are more than this, our approach is to apply branch and bound to successive overlapping sub-sections. In cases where $n > 14$ locations, the route is split into $2\left\lceil \frac{n}{14} \right\rceil - 1$ sub-sections. For example, if a route consists of 28 locations it is split into 3 sub-sections containing locations 1-14, 7-21 and 14-28 respectively.

The branch and bound process starts with a set of currently adjacent locations. According to the constraints of the problem, partly constructed solutions can be discarded: (a) if the delivery location of a request is serviced before the corresponding pickup; (b) if there remains a location still to be serviced that can no longer be feasibly serviced within its time window; (c) if a location cannot be feasibly serviced within its time window, when placed after another location; (d) if the current total distance travelled of the sub-section exceeds the minimum recorded so far; and (e) if the minimum distance still to travel plus the current distance of the partly constructed sub-section exceeds the limit of the upper bound.

The limit of the initial upper bound is set to the total distance travelled in the sub-section before the locations are removed from the route. Branches are searched in order of the location where service can begin first and the search terminates once a complete exploration has taken place before returning the best found solution. If this new solution improves on the total distance of the sub-section before the locations were removed from the route, then it replaces the original solution.

A simple example of applying our branch and bound heuristic is now provided. The example consists of a single route with 3 requests and the starting solution can be found in Figure 4.5. The total distance of the starting solution is equal to 60 and the route is indicated by the solid lines between locations. The distances between other locations are provided via dashed lines between locations. The time window for a location is provided in a bracket next to each node.

There is a service time of 10 at each location and the start and end service times at each location in the starting solution are provided inside the corresponding node under the request number. This is accompanied by either a 'p' identifying a pickup location or a 'd' identifying a delivery location. For example '1p' identifies this location is the pickup location of request 1 and the, (10, 20), identifies that service at this location begins at time 10 and is completed at time 20. Furthermore the time window, (0, 25), identifies that service can begin at this location at time 0 and must start no later than time 25. The end of the scheduling horizon is 150; this is the latest time that the

vehicle can return to the depot.

As can be seen from Figure 4.5 the initial solution consists of leaving the depot at time 0 to service the pickup location of request 1. Service finishes at this location at time 20 and the vehicle leaves to service the pickup location of request 2. Service begins at time 30 and ends at time 40, the vehicle then leaves to service the delivery location of request 2. The vehicle arrives at time 45, but must wait until time 50 before service can begin. Service is completed at time 60 and the vehicle leaves to service the delivery location of request 1. The vehicle arrives at time 70; service begins and is completed at time 80. The vehicle then leaves to service the pickup location of request 3 at time 85 and service is completed at time 95. The vehicle leaves to service the final location, being the delivery location of request 3, arriving at time 110. The vehicle must wait at this location until time 120 before service can begin. Service is completed at time 130 and the vehicle returns to the depot at time 135, before the end of its scheduling horizon. As there are only 6 locations, they will be removed in one sub-section to be optimised by the branch and bound method.

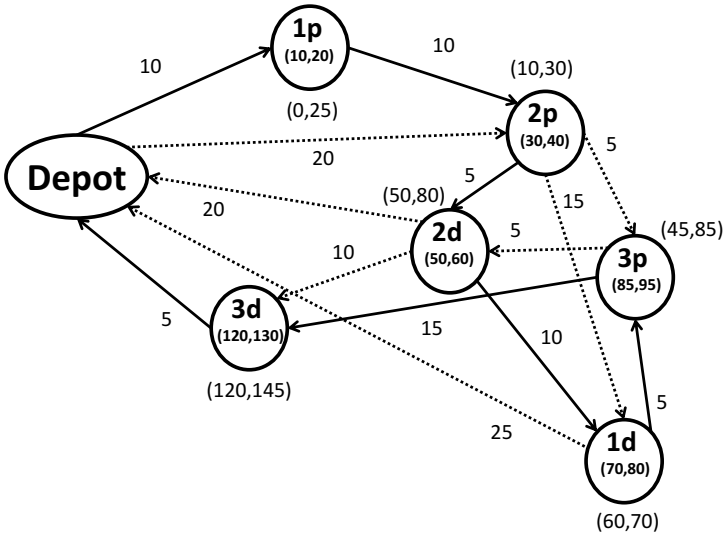


Figure 4.5: Solution before application of the Branch and Bound Heuristic

At each level in the branch and bound search tree, there is a maximum of one location for each request that can be feasibly serviced, due to the precedence constraints. Before starting the search it is clear from the time window constraints that the pickup location of request 1 must be serviced prior to any other location and must be followed by the pickup location of request 2. It is also clear the delivery location of request 3 must be serviced last; these priority constraints are added to further guide the search. The search tree for the example provided in Figure 4.5 can be found in Figure 4.6.

Starting the process of searching the tree for an improved solution, the pickup location of request 1 is the only feasible branch at the first level of the search tree and the pickup location of request 2 is the only feasible branch at the second level of the search tree, service will finish at the pickup location of request 2 at time 40 and the cost of the solution so far is 20.

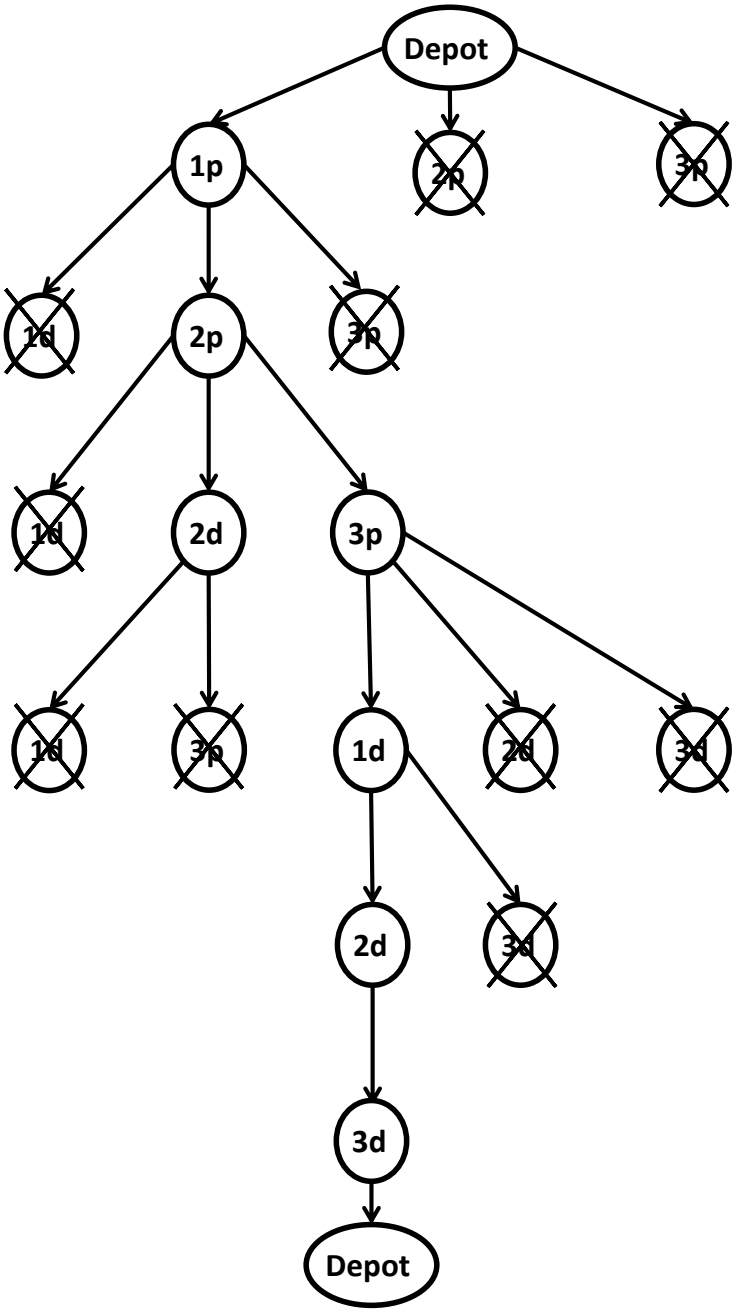


Figure 4.6: The Branch and Bound Search Tree

At the next stage there are 2 branches which can feasibly be serviced, based on the criterion, the first branch to be searched is the one where service can begin earliest,

therefore the pickup location of request 3 is chosen and this branch is traversed. Service begins at time 45 and ends at time 55 and the total distance travelled so far now becomes 25. The branches not considered here will be revisited later in the search as all possibilities need to be explored.

After servicing the pickup location of request 3, there are 3 options to consider. Due to the constraints, the only feasible branch is the delivery location of request 1 which is therefore traversed, the vehicle arrives at this location at time 60 and service begins. Service is completed at time 70 and the total distance of the solution so far now stands at 30. There are only 2 locations still to be serviced and it is clear that the delivery location of request 2 is the only branch which is feasible, as the delivery location of request 3 must be serviced last.

The final branches are then traversed, with the final location being the delivery location of request 3. The vehicle will have to wait here until time 120 until service can begin and then it returns to the depot at time 135, before the end of the scheduling horizon. All locations have now been feasibly serviced and a new solution has been found with a total distance equal to 55, which improves the original starting solution. This now becomes the new best found solution and the search continues until all branches are explored.

Stepping back up the search tree to the third level of the search, the choice of the delivery of request 2 at this stage still needs to be explored. If the delivery of request 2 had been chosen this would have been serviced at time 50 and the cost of the solution so far would have been 45. At the next stage of the search there would then have been 2 options, either the delivery of request 1 or the pickup of request 3. If the delivery of request 1 is chosen then the cost of traversing this branch is 15. This would increase the total cost of the solution so far to 40. A bound would then be calculated based on the minimum distance still to travel, which at this stage would include the return distance to the depot. It is known that the distance to the depot from the delivery location of request 1 is 25, hence this would exceed the cost of the best found solution so far and so this solution can be discarded. If the pickup of request 3 was chosen then service here could begin at time 65 and would be completed at time 75, however this then would result in the delivery of location 1 becoming infeasible due to the time window constraints. Therefore traversing the remaining branches would result in exceeding the limit on the upper bound or obtaining an infeasible solution, hence these become infeasible and the search is completed. The new best found solution achieved is shown in Figure 4.7.

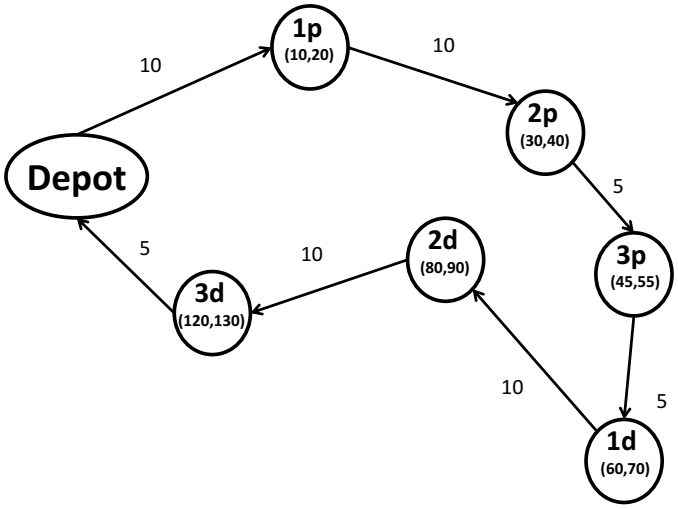


Figure 4.7: Solution after application of the Branch and Bound Heuristic

The procedure for the branch and bound heuristic is described in full in Algorithm 12.

Algorithm 12 BranchBound

Let M be the number of vehicles in the starting solution
for $(v = 0; v < M; v++)$ **do**
    Calculate the number of overlapping sub-sections required
    for (Each overlapping sub-section) **do**
        Let s be the current solution for the sub-section
        Let $v_{best} = \emptyset$ be the current best solution of route v
        Let $\{S_1\}$ be the set of all locations in the sub-section
        Calculate the priority constraints for each location in $\{S_1\}$
        Set all infeasible branches at each level as having been explored
        $start = 0$
        repeat
            for (level=start; level< $|\{S_1\}|$; level++) **do**
                Choose the unexplored branch where service at that location can begin first
                Remove this location from $\{S_1\}$
                Set branch as explored
            if (All locations have been inserted) **then**
                Let new solution be s'
                if (s' is better than s) **then**
                    $s' \leftarrow s_{best}$
                $start = level - 2$
                Re-insert last 2 locations into $\{S_1\}$
            else
                $start = level - 1$
                Re-insert last location into $\{S_1\}$
        until (All branches have been explored)
        if ($s_{best} \neq \emptyset$) **then**
            $s \leftarrow s_{best}$

The branch and bound heuristic will now be applied to the final solutions achieved by the methods shown in Section 4.4. To investigate the improvements made through the addition of the branch and bound heuristic, comparisons are again made with the best known solutions from the literature. The best 4 cases as described in Table 4.5 will continue to be analysed, this time for the 15 instances out of a total of 56 where the best found solution has still not been achieved in every case. The percentage difference in the total distance travelled compared to the best known solutions, for each of these instances and each case is shown in Figure 4.8.

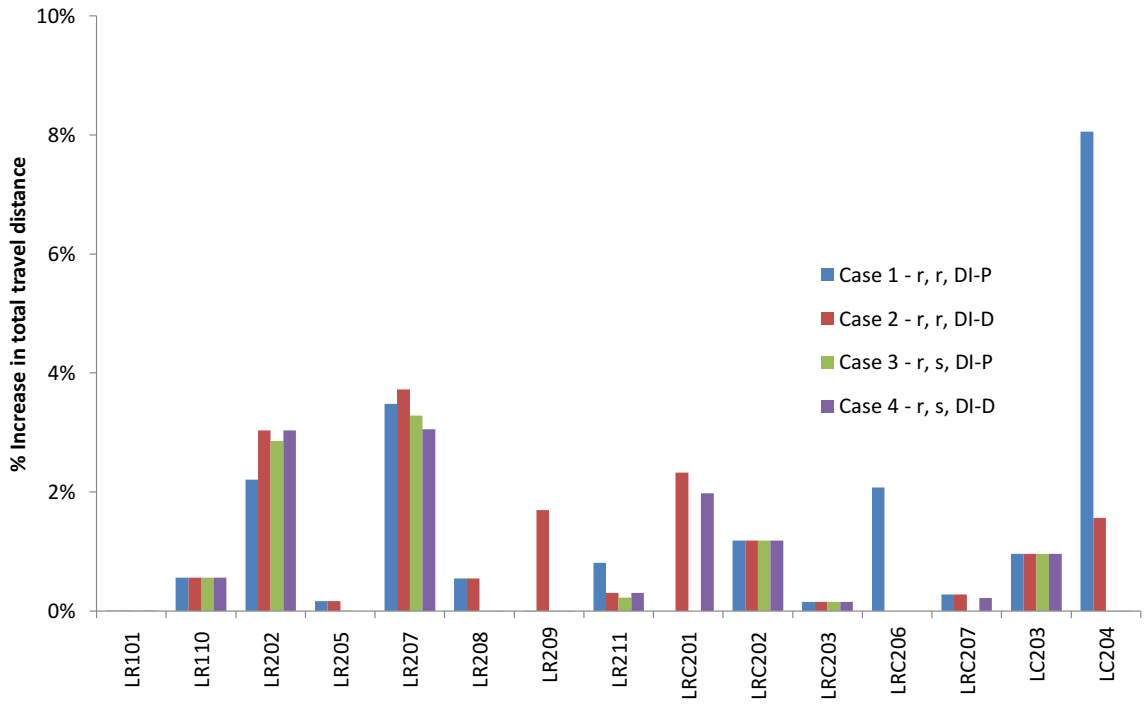


Figure 4.8: Difference in TD from the Best Known solutions after application of the Branch and Bound Heuristic, by each Case for the worst instances

Comparing Figure 4.8 to Figure 4.2, a clear improvement can be seen after the addition of the branch and bound heuristic. There are now many more instances where the best known solution has been achieved across all of the cases. Case 3 achieves 47 of the 56 best known solutions, Case 4 achieves 46 of the best known solutions and, Case 1 and 2 achieve 43 and 42 best known solutions respectively. For 8 of the instances the best known solution has not been achieved in any case, highlighting once again that an improvement to the solutions could still be made.

Table 4.8 provides results for the difference in total distance travelled from the best found solution by each method, compared to the best known results found in the literature.

	Case 1	Case 2	Case 3	Case 4
LR101	0.02	0.02	0.02	0.02
LR110	6.48	6.48	6.48	6.48
LR202	26.42	36.36	34.24	36.36
LR205	1.75	1.75	0.12	0.00
LR207	31.40	33.64	29.65	27.58
LR208	4.02	4.02	0.00	0.00
LR209	0.00	15.77	0.00	0.00
LR211	7.16	2.70	2.00	2.70
LRC201	0.00	32.73	0.00	27.86
LRC202	16.29	16.29	16.29	16.29
LRC203	1.66	1.66	1.66	1.66
LRC206	24.10	0.00	0.00	0.00
LRC207	2.98	2.98	0.00	2.35
LC203	5.61	5.61	5.61	5.61
LC204	47.58	9.23	0.00	0.00
Total	175.47	169.24	96.07	126.91

Table 4.8: Difference in TD from the Best Known solutions after application of the Branch and Bound Heuristic, by each Case for the worst instances

It is the results of Case 3 which again achieves the most promising results. A decrease of $\approx$ 27% has been achieved in the total difference in distance from the best known solutions compared to if the branch and bound heuristic had not been applied. For Case 1 a decrease of 22% is achieved, for Case 2 a decrease of 17% is achieved and for Case 4 again a decrease of 22% is achieved.

The overall improvement is summarised in Table 4.9, using the results of Case 3, which achieved the overall minimum total distance travelled. Results are summarised both prior to the branch and bound heuristic being applied (corresponding to the results achieved in Section 4.4) and after the addition of the heuristic (corresponding to the results provided in Table 4.8). As before, TD represents the total distance travelled and NV represents the number of vehicles required by the solution.

	Best known		Without BB		With BB	
	TD	NV	TD	NV	TD	NV
LC1	7445.42	90	7445.41	90	7445.41	90
LR1	14635.41	143	14641.91	144	14641.91	144
LRC1	11088.34	93	11088.87	93	11088.34	93
LC2	4713.26	24	4725.03	24	4718.87	24
LR2	10652.00	31	10734.26	33	10718.00	33
LRC2	9064.98	26	9094.89	28	9082.93	28
Total	57599.41	407	57730.35	412	57695.45	412

Table 4.9: Comparison of TD and NV achieved after application of the Branch and Bound Heuristic to the Best Known solutions for each set of instances

It can be see that the addition of the branch and bound heuristic improves the solutions achieved. The average total travel distance for the 56 instances before the branch and bound phase is 57730.35, which is reduced to 57695.45, a decrease of 0.06%. Note that there is no improvement achieved with the instances with a short scheduling horizon, but these are very close to the best known solutions, with only 2 instances remaining where the best known solution has not been achieved. The overall solutions are now on average 0.17% above the best known solutions. This confirms that it is the instances with a longer scheduling horizon that can benefit from a method which focuses on optimising large portions of locations in routes.

The next section will investigate the computational times of our algorithm and will attempt to reduce the overall time required.

4.6 Improving Run Times for Our Algorithm

It is clear that the computational times of our algorithm still needs to be addressed. This section looks to improve the computational time required for both the tabu search heuristic and the branch and bound heuristic.

Addressing the tabu search heuristic first, presently at each iteration this method identifies the best feasible move from all requests. Each request is first removed from its route and it is checked whether this results in an indirect tabu move. As stated previously, if removing a request from a particular route results in an arrangement of locations remaining in that route which are 'tabu', then the move is indirectly classed as a tabu move. If so, this is noted, but the search continues. The best feasible insertion

position of that request is identified then is checked as to whether this results in a direct tabu move. The change in total travel distance is then found.

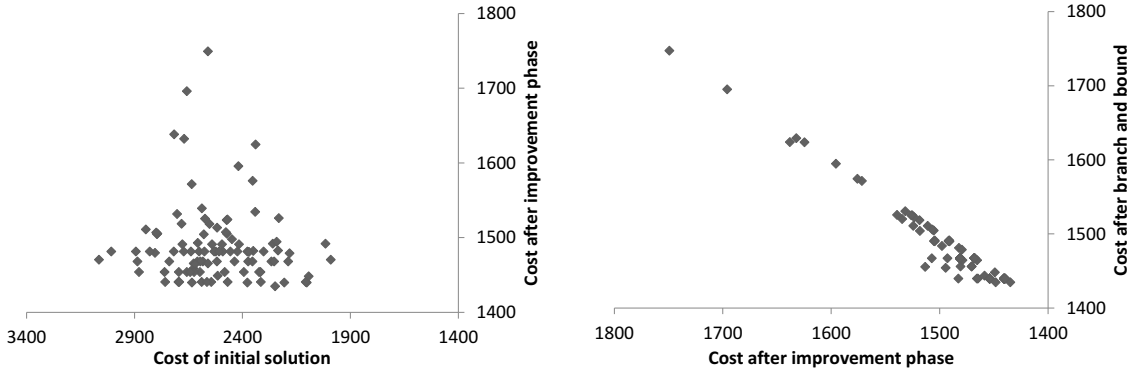
If both the removal and the insertion of the request did not result in a tabu move, then this is a feasible move and it is checked whether it is the best found move for that request at this iteration. If so, it is stored. If either the removal or insertion of the request resulted in a move that was deemed 'tabu' but the total distance travelled of the new route is better than the overall best found solution, then it is also stored. After all requests are investigated the request whose best stored move results in the lowest overall total travel distance is accepted.

One way of decreasing the computational time of the tabu search heuristic could be to stop the search for a request immediately if its removal from its route results in an indirect move which is deemed 'tabu'. This may result in a move which was deemed indirectly tabu, but still improved the overall best solution, not being executed. However, preliminary results show that this did not affect the overall minimum solution achieved for each instance, but did improve the computational times of the algorithm for each case by an average of 23%. This approach is therefore adopted.

It is hoped that further improvements can be made to the computational times of the algorithm by identifying whether only a subset of the most promising solutions achieved in the initial construction phase may be used during the improvement phase. Figure 4.9a contains a plot of the total distance travelled of the solutions after the initial construction phase compared to after the improvement phase, for 100 runs using Case 3 and instance LRC201.

Investigations are also performed to determine if the branch and bound heuristic may be performed on a subset of the most promising solutions found and still achieve competitive results. Figure 4.9b provides a scatter plot of the total distance achieved by our algorithm both before and after the branch and bound phase, for 100 runs on instance LRC201.

It is clear that there is no correlation between achieving a lower initial solution and achieving a lower solution after the improvement phase. The p-value for this test is 0.132, not significant even at the 10% level. Therefore, it is probably not useful to select a proportion of initial solutions with a certain initial cost to apply the improvement phase to.



(a) Before and after the improvement phase (b) Before and after the branch and bound heuristic

Figure 4.9: Correlations of Total Travel Distance before and after Improvement Heuristics

There is however a significant correlation between achieving a lower cost after the tabu search heuristic and achieving a lower cost for the final solution after the branch and bound heuristic. The p-value of <0.000 shows this result is significant at the 1% level and with a correlation coefficient of 0.91, that this is a strong positive relationship. These are both 1-tailed tests since the results will only stay the same or improve after each phase.

The total travel distance obtained after the branch and bound heuristic is applied for the best 10% of solutions from the improvement phase is compared to the total travel distance of the remaining 90% of solutions. A Wilcoxon signed rank test shows that there is a significant difference in the results achieved at the 5% level (even at 1%). Therefore, it seems reasonable to select a small subset of low cost solutions after application of the improvement phase, which can then be improved via the branch and bound heuristic without any loss in the quality of the best found solution

Preliminary results suggest that selecting the best 10% of solutions achieved after the improvement phase to which the branch and bound heuristic is then applied, results in no loss in solution quality, but a further reduction in the computational time. On average the computational time is decreased by $\approx 7\%$. Due to this further reduction in computational time and no loss in the quality of the solutions achieved, this approach will also be adopted.

The computational times required for Cases 3 and 4 are still not comparable to what is achieved in the literature, therefore final improvements are attempted with Cases 1 and 2, which apply a lower maximum number of iterations. It is interesting to investigate the aspiration criterion of the tabu search heuristic. As stated above, it was found that no longer accepting a move that is indirectly 'tabu', even if it improves the overall best

found solution, does not result in a loss in the total distance achieved in the best found solution.

Referring back to Figures 4.3 and 4.4 in Section 4.4, it was clear that in Cases 3 and 4, a greater change was made to the solution during the search, which resulted in the reconstruction heuristics achieving a greater improvement to the solutions. This was not achieved in Cases 1 and 2; hence the solution was not significantly altered during the search, giving inferior results. For the case of a direct 'tabu' move, if this was no longer permitted, it could result in furthering the search in unexplored regions for Cases 1 and 2 and could result in escaping from a local minimum which had not been previously possible.

Preliminary investigations show that removing the aspiration criterion from the search, did not produce as good a solution for the cases where the maximum number of iterations is equal to the number of services was applied. This is what would be expected. However, for Case 2, where a lower number of maximum iterations was applied, the difference in the total distance travelled compared to the best known solutions was improved by 10%. For Case 1, results were similar to those previously achieved.

Algorithm 13 TabuInsert (Set of Requests, Set of Routes)

```
1: Initialise LocalMin ← ∞
2: for (Set of all Requests) do
3:     Let r be the request
4:     for (Set of all Routes) do
5:         Let v be the route
6:         Let the pickup location of r be p
7:         Let the delivery location of r be d
8:         for (All feasible insertion positions of p in v) do
9:             if (Insertion position of p is not 'tabu') then
10:                Insert p in v
11:                for (All feasible insertion positions of d in v) do
12:                    if (Insertion position of d is not 'tabu') then
13:                        Insert d in v
14:                        Calculate △cost {△cost is the change in solution cost due to
   the insertion}
15:                        if (△cost < LocalMin) then
16:                            LocalMin ← △cost
17:                            v_best ← v */v_best is the current best route of request r/*
18:                            r_best ← r */r_best is the current best request in route v/*
19:                            p_best ← p */p_best is the current best insertion of p/*
20:                            d_best ← d */d_best is the current best insertion of d/*
21: Insert request r_best in route v_best in positions p_best and d_best
```

Due to the improvements made to the solutions applying a lower maximum number of iterations and the increase in computational times for Cases 3 and 4, Case 2 will

be adopted for the tabu search heuristic to be applied in this research. This consists of a tabu length equal to the maximum number of iterations, equal to the number of requests. The tabu attribute to be stored is the direct and indirect delivery edges. The overall procedure is outlined in Algorithms 13 and 14.

Algorithm 14 TABUMOVE

```
1: Let r be the number of requests
2: Let tabu_tenure = r
3: Let max_iteration = r
4: Let s be the current best solution
5: Let s_global ← s
6: Set tabu_list = ∅
7: repeat
8:     Let S_1 be the set of all requests where removing that request from its route is not 'tabu'
9:     Let s_best ← ∅ */ s_best is the current best solution
10:    for (All available requests in S_1) do
11:        Choose a request r from S_1
12:        Remove r from s
13:        Run TABUINSERT ({r} , ∀ routes ∈ s)
14:        Let new solution be s'
15:        if (s_best = ∅) then
16:            s_best ← s'
17:        else
18:            if (s' is better than s_best) then
19:                s_best ← s'
20:    if (s_best ≠ ∅) then
21:        s ← s_best
22:        if |tabu_list| = tabu_tenure then
23:            Delete first element in tabu_list
24:        Add latest move to the end of tabu_list
25:        if (s_best is better than s_global) then
26:            s_global ← s_best
27:            iteration ← 0
28:        else
29:            iteration ← iteration + 1
30:    else
31:        iteration ← max_iteration
32: until iteration ← max_iteration
```

The next section summarises the final results achieved by our algorithm where comparisons are made with the best known solutions from the literature.

4.7 Summary of Results

Our final algorithm for the PDPTW, as outlined in Algorithm 15, differs from others in the literature as a single run involves multiple re-starts and only a portion of the best found solutions are passed to the final improvement phase.

Table 4.10 provides summary results for our algorithm where the initial phase consisted of 100, 200 and 300 iterations and 10% of the best found solutions are passed to the final improvement phase. They are compared to the best known results in the literature. Once again, TD is the total distance travelled over all instances in the set and NV is the total number of vehicles required. With regards to the best known number of vehicles, these are the best known number of vehicles when the objective is to minimise the total travel distance and not the number of vehicles.

	Best known		100 iterations		200 iterations		300 iterations	
	TD	NV	TD	NV	TD	NV	TD	NV
LC1	7445.42	90	7445.41	90	7445.41	90	7445.41	90
LR1	14635.41	143	14642.42	144	14642.42	144	14642.42	144
LRC1	11088.34	93	11088.34	93	11088.34	93	11088.34	93
LC2	4713.26	24	4728.09	24	4718.87	24	4718.87	24
LR2	10652.00	31	10729.58	33	10713.81	33	10652.00	31
LRC2	9064.98	26	9129.83	28	9093.23	27	9081.27	27
Total	57599.41	407	57763.67	412	57702.07	411	57628.30	409

Table 4.10: Comparison of TD and NV achieved by the MetaheuristicAlgorithm to the Best Known solutions, by varying numbers of iterations for each set of instances

Our results are $\approx 0.05\%$ above the best known solutions in terms of total distance travelled for 300 iterations. For the case of 100 and 200 iterations results are 0.29% and 0.18% above the best known solution respectively. This shows our algorithm produces competitive results.

Algorithm 15 MetaheuristicAlgorithm

```
1: Declare number of iterations of initial improvement phase
2: Initialise s_best to an arbitrary large value
3: Initialise best_list to ∅
4: Let bb_iterations = 10%iterations
5: for (i = 1; i < iterations; i++) do
6:     Run RandomInsertion
7:     Let initial solution be s
8:     Let Improve = TRUE
9:     while (Improve = TRUE) do
10:        Run TabuMove
11:        Run Exchange
12:        Run DoubleRoute
13:        Run TripleRoute
14:        Run SingleRoute
15:        Run SingleMove
16:        Let new solution be s'
17:        if (cost(s') < cost(s)) then
18:            s ← s'
19:        else
20:            Improve=FALSE
21:    if (|best_list| < bb_iterations) then
22:        Add s to best list
23:    else
24:        if (s < s_worst) then
25:            Delete s_worst from best_list
26:            Add s to best_list
27:    Let s_worst be the solution in best_list where cost(s) is greatest
28: Let s_best be the current best solution
29: for (All solutions in best_list) do
30:    Let initial solution be s
31:    Run BranchBound
32:    if (cost(s)) < cost(s_best) then
33:        s_best ← s
```

In Table 4.11 results of our algorithm are provided for 100, 200 and 300 iterations and are compared with those of Li and Lim [2001], Pankratz [2005a], Dergis and Dohmer [2008] and Ding et al. [2009]. Caution is needed when making a direct comparison with these results due to the differences in the objective functions applied.

Our objective function is to minimise the total travel distance, which is comparable to that of Pankratz [2005a]. Li and Lim [2001] however use a prioritised objective function with the order being: (1) minimise the number of vehicles; (2) minimise the total travel distance; (3) minimise the total schedule duration; and (4) minimise the total waiting time. The objective of Ding et al. [2009] is similar to this although it does not include minimising the total schedule duration. The objective of Dergis and Dohmer [2008] is to minimise the number of vehicles followed by minimising the total

travel distance.

Considering the results in Table 4.11 for 300 runs of our algorithm, we achieve the best known solutions for 51 of the 56 problem instances and with a total travel distance of 57628.34, this is competitive with the state of the art. In fact, a lower total travel distance is achieved by our algorithm compared to any of the results from the literature.

For the instance LC203 the best known result since that of Li and Lim [2001] has been obtained. In Li and Lim [2001] and Pankratz [2005a] Euclidean distances calculated directly from the instances were rounded to 2 decimal places and this could account for some small discrepancies when comparing the total distance travelled for the instances of LR101 and LR207. It is therefore, really only for 2 instances that solutions are not comparable to the best known. For 200 iterations, 49 of the best known solutions are achieved and for 100 iterations, 44 of the best known solutions are achieved.

For Li and Lim [2001] 40 of the 56 best known solutions are achieved with a total travel distance of 58184.91, for just 100 iterations our algorithm improves on these results. Dergis and Dohmer [2008] achieve 47 of the 56 best known solutions with a total travel distance of 57678.40. For 200 iterations our algorithm improves the number of best found solutions, but 300 iterations are needed to improve on the total travel distance achieved. Ding et al. [2009] achieve 51 of the 56 best known solutions with a total travel distance of 57652.05; this is comparable to our results for 300 iterations.

A total travel distance of 57638.48 is achieved by Pankratz [2005a], whose results are directly comparable to those obtained by our algorithm. This is improved by our algorithm for the case of 300 iterations. However, only 43 of the best known solutions are found, which is improved upon by 100 iterations of our algorithm. Our algorithm performs consistently well across the varying instance types whereas Li and Lim [2001] and Pankratz [2005a] struggle with the instances of a longer scheduling horizon, in particular LR2 and LRC2.

It should be noted that for the instances LC104 and LRC101, a solution has been found by Li and Lim [2001], Dergis and Dohmer [2008] and Ding et al. [2009] (and by Li and Lim [2001] for the case of LRC201), that requires one fewer vehicle. However, this increases the total travel distance of the solution. These are the only instances where the objectives of minimising the number of vehicles and minimising the total travel distance do not share an identical best known solution. If we had have accepted the solutions for the two cases above, the total travel distance for the case of 300 iterations would still be less than Li and Lim [2001] and Dergis and Dohmer [2008]. This shows the robustness of our algorithm to changes in the objective function as it also achieved the solutions stated above, however they were disregarded due to the

	100 runs	200 runs	300 runs	Li & Lim	Pankratz	Dergis et al.	Ding & Dohmer
LC101	828.94	828.94	828.94	828.94	828.94	828.94	828.94
LC102	828.94	828.94	828.94	828.94	828.94	828.94	828.94
LC103	827.87	827.87	827.87	827.86	827.86	827.86	827.86
LC104	818.60	818.60	818.60	861.95	818.60	860.01	860.01
LC105	828.94	828.94	828.94	828.94	828.94	828.94	828.94
LC106	828.94	828.94	828.94	828.94	828.94	828.94	828.94
LC107	828.94	828.94	828.94	828.94	828.94	828.94	828.94
LC108	826.44	826.44	826.44	826.44	826.44	826.44	826.44
LC109	827.82	827.82	827.82	827.82	827.82	827.82	827.82
LC1	7445.43	7445.43	7445.43	7488.77	7445.42	7486.83	7486.83
LR101	1650.80	1650.80	1650.80	1650.78	1650.80	1650.80	1650.80
LR102	1487.57	1487.57	1487.57	1487.57	1487.57	1487.57	1487.57
LR103	1292.68	1292.68	1292.68	1292.68	1292.68	1292.68	1292.68
LR104	1013.39	1013.39	1013.39	1013.39	1013.99	1013.99	1013.39
LR105	1377.11	1377.11	1377.11	1377.11	1377.11	1377.11	1377.11
LR106	1252.62	1252.62	1252.62	1252.62	1252.62	1252.62	1252.62
LR107	1111.31	1111.31	1111.31	1111.31	1111.31	1111.31	1111.31
LR108	968.97	968.97	968.97	968.97	968.97	968.97	968.97
LR109	1208.96	1208.96	1208.96	1239.96	1208.96	1208.96	1208.96
LR110	1166.34	1166.34	1166.34	1159.35	1165.83	1159.35	1159.35
LR111	1108.90	1108.90	1108.90	1108.90	1108.90	1108.90	1108.90
LR112	1003.77	1003.77	1003.77	1003.77	1003.77	1003.77	1003.77
LR1	14 642.42	14 642.42	14 642.42	14 666.41	14 642.51	14 636.03	14 635.43
LRC101	1703.21	1703.21	1703.21	1708.80	1703.21	1708.80	1708.80
LRC102	1558.07	1558.07	1558.07	1563.55	1558.07	1558.07	1558.07
LRC103	1258.74	1258.74	1258.74	1258.74	1258.74	1258.74	1258.74
LRC104	1128.40	1128.40	1128.40	1128.40	1128.40	1128.40	1128.40
LRC105	1637.62	1637.62	1637.62	1637.62	1637.62	1637.62	1637.62
LRC106	1424.73	1424.73	1424.73	1425.53	1424.73	1424.73	1424.73
LRC107	1230.14	1230.14	1230.14	1230.15	1230.14	1230.14	1230.15
LRC108	1147.43	1147.43	1147.43	1147.97	1147.43	1147.96	1147.43
LRC1	11 088.34	11 088.34	11 088.34	11 100.76	11 088.34	11 094.46	11 093.94
LC201	591.56	591.56	591.56	591.56	591.56	591.56	591.56
LC202	591.56	591.56	591.56	591.56	591.56	591.56	591.56
LC203	591.17	591.17	591.17	585.56	591.17	591.17	591.17
LC204	599.83	590.60	590.60	591.17	590.60	590.60	590.60
LC205	588.88	588.88	588.88	588.88	588.88	588.88	588.88
LC206	588.49	588.49	588.49	588.49	588.49	588.49	588.49
LC207	588.29	588.29	588.29	588.29	588.29	588.29	588.29
LC208	588.32	588.32	588.32	588.32	588.32	588.32	588.32
LC2	4728.10	4718.87	4718.87	4713.83	4718.87	4718.87	4718.87
LR201	1253.23	1253.23	1253.23	1263.84	1253.23	1253.23	1253.23
LR202	1234.03	1231.91	1197.67	1197.67	1197.67	1197.67	1197.67
LR203	949.40	949.40	949.40	949.40	952.29	949.40	949.40
LR204	849.05	849.05	849.05	849.05	849.05	849.05	849.05
LR205	1055.77	1054.02	1054.02	1054.02	1054.02	1054.02	1054.02
LR206	931.63	931.63	931.63	931.63	931.63	931.63	931.63
LR207	930.63	930.63	903.06	903.06	903.60	903.06	903.05
LR208	734.85	734.85	734.85	734.85	736.00	734.85	734.85
LR209	930.59	930.59	930.59	937.05	932.43	930.59	930.59
LR210	976.13	964.22	964.22	964.22	964.22	964.22	964.22
LR211	884.29	884.29	884.29	927.80	888.15	896.76	884.29
LR2	10 729.60	10 713.82	10 652.01	10 712.59	10 662.29	10 664.48	10 652.00
LRC201	1439.67	1406.94	1406.94	1468.96	1407.21	1406.94	1406.94
LRC202	1390.56	1390.56	1390.56	1374.27	1385.25	1374.27	1374.27
LRC203	1089.07	1089.07	1089.07	1089.07	1093.89	1089.07	1089.07
LRC204	818.66	818.66	818.66	827.78	818.66	818.66	818.66
LRC205	1302.20	1302.20	1302.20	1302.20	1302.20	1302.20	1302.20
LRC206	1174.86	1170.99	1159.03	1162.91	1159.03	1159.03	1159.03
LRC207	1062.05	1062.05	1062.05	1424.60	1062.05	1062.05	1062.05
LRC208	852.76	852.76	852.76	852.76	852.76	865.51	852.76
LRC2	9129.83	9093.23	9081.27	9502.55	9081.05	9077.73	9064.98
Total	57 763.72	57 702.11	57 628.34	58 184.91	57 638.48	57 678.40	57 652.05

Table 4.11: Comparison of TD achieved by the MetaheuristicAlgorithm to the best known solutions of Li and Lim [2001], Pankratz [2005a], Dergis and Dohmer [2008] and Ding et al. [2009]

increase in distance.

The computational times and number of vehicles for the results provided in Table 4.11 are provided in Table 4.12. For the approach of Li and Lim [2001] the overall number of independent runs per instance is not reported and the average solution quality is not discussed, therefore it is not known what the computational times reported are comparable to. The best results of Pankratz [2005a] and Ding et al. [2009] are reported after 30 runs of their algorithm and for Dergis and Dohmer [2008], best results are reported after 10 runs.

Caution should be applied when making comparisons due to differences in computational time and computer specification. The algorithms of Li and Lim [2001] and Ding et al. [2009] are implemented in C++, whereas the procedure of Pankratz [2005a] is implemented in JAVA and the programming of Dergis and Dohmer [2008] were carried out in Pascal. The experiments of Dergis and Dohmer [2008] were carried out on a 2.8-GHz P4 machine, those of Ding et al. [2009] are performed on a 1-GHz P3 machine, while the times reported by Pankratz [2005a] were obtained on a 2-GHz P4 machine. Li and Lim [2001] reported they used an i686 PC under Linux.

Firstly, considering the number of vehicles, the results of our algorithm are directly comparable to the results of Pankratz [2005a]. A lower number of vehicles have therefore been achieved for the case of 300 iterations.

Comparing the computational times, with caution, the results for 300 iterations improve on those of Pankratz [2005a] and Ding et al. [2009]. The results for 200 iterations improve on those of Li and Lim [2001] and are closer to those of Dergis and Dohmer [2008]. It is clear the run time of our algorithm for the instances with a short scheduling horizon, improves on the results provided in the literature. However, it appears further improvement can still be made for the computational times required by the instances with a longer scheduling horizon.

	100 runs		200 runs		300 runs		Li & Lim		Pankratz		Ding et al.		Dergis & Dohmer	
	NV	CT	NV	CT	NV	CT	NV	CT	NV	CT	NV	CT	NV	CT
LC101	10	12	10	25	10	37	10	33	10	1651	10	24	10	
LC102	10	29	10	58	10	89	10	71	10	2060	10	105	10	
LC103	10	107	10	209	10	316	10	191	10	2560	10	440	10	
LC104	10	246	10	482	10	714	9	1254	10	4197	9	1621	9	
LC105	10	15	10	30	10	45	10	47	10	1750	10	41	10	
LC106	10	17	10	34	10	52	10	43	10	1908	10	70	10	
LC107	10	19	10	38	10	56	10	54	10	1979	10	99	10	
LC108	10	42	10	87	10	130	10	82	10	2430	10	205	10	
LC109	10	110	10	221	10	332	10	255	10	3543	10	616	10	
LC1	90	598	90	1184	90	1773	89	2030	90	22 078	89	3220	89	2294
LR101	19	5	19	11	19	16	19	87	19	2010	19	17	19	
LR102	17	39	17	80	17	118	17	1168	17	2686	17	50	17	
LR103	13	59	13	117	13	176	13	169	13	2486	13	198	13	
LR104	9	180	9	357	9	541	9	459	9	4105	9	845	9	
LR105	14	15	14	30	14	45	14	69	14	2129	14	33	14	
LR106	12	29	12	58	12	87	12	87	12	2440	12	112	12	
LR107	10	78	10	154	10	233	10	287	10	2870	10	316	10	
LR108	9	118	9	238	9	358	9	415	9	3075	9	469	9	
LR109	11	40	11	84	11	128	11	348	11	3114	11	217	11	
LR110	11	96	11	198	11	298	10	547	11	4264	10	542	10	
LR111	10	97	10	202	10	300	10	179	10	3166	10	427	10	
LR112	9	231	9	457	9	694	9	638	9	5061	9	1309	9	
LR1	144	987	144	1986	144	2993	143	4453	144	37 406	143	4535	143	4526
LRC101	15	11	15	22	15	33	14	119	15	2282	14	42	14	
LRC102	12	38	12	76	12	114	13	152	12	2866	12	151	12	
LRC103	11	75	11	148	11	231	11	175	11	2784	11	324	11	
LRC104	10	135	10	266	10	406	10	202	10	3247	10	807	10	
LRC105	13	28	13	57	13	86	13	179	13	2645	13	97	13	
LRC106	11	32	11	64	11	96	11	459	11	2869	11	145	11	
LRC107	11	67	11	131	11	200	11	154	11	2932	11	391	11	
LRC108	10	103	10	202	10	301	10	650	10	3350	10	758	10	
LRC1	93	489	93	966	93	1467	93	2090	93	22 975	92	2715	92	2378
LC201	3	80	3	157	3	240	3	27	3	1778	3	12	3	
LC202	3	200	3	384	3	579	3	94	3	3465	3	194	3	
LC203	3	346	3	668	3	983	3	145	3	5750	3	1053	3	
LC204	3	866	3	1678	3	2584	3	746	3	10 384	3	4500	3	
LC205	3	98	3	199	3	297	3	190	3	2822	3	106	3	
LC206	3	166	3	333	3	506	3	88	3	3726	3	256	3	
LC207	3	156	3	304	3	454	3	102	3	4111	3	372	3	
LC208	3	171	3	345	3	520	3	178	3	4258	3	487	3	
LC2	24	2081	24	4067	24	6164	24	1570	24	36 294	24	6979	24	1589
LR201	4	97	4	187	4	276	4	193	4	3103	4	169	4	
LR202	4	257	4	525	3	785	3	885	3	7035	3	1249	3	
LR203	3	685	3	1419	3	2020	3	1950	3	11 445	3	2772	3	
LR204	2	1816	2	3560	2	5345	2	2655	2	14 595	2	6326	2	
LR205	3	362	3	761	3	1121	3	585	3	4991	3	1034	3	
LR206	3	652	3	1294	3	1914	3	747	3	7771	3	1786	3	
LR207	3	1432	3	2768	2	3970	2	1594	2	12 563	2	5140	2	
LR208	2	2357	2	4677	2	7218	2	3572	2	15 932	2	9967	2	
LR209	3	593	3	1177	3	1758	3	2773	3	7107	3	2712	3	
LR210	3	519	3	1091	3	1728	3	1482	3	8584	3	2636	3	
LR211	3	1202	3	2463	3	3624	2	4204	3	14 356	3	9618	3	
LR2	33	9970	33	19 920	31	29 757	30	20 640	31	107 482	31	43 409	31	14 820
LRC201	5	98	4	197	4	283	4	266	4	3044	4	264	4	
LRC202	4	250	4	486	4	718	3	987	4	4851	3	818	3	
LRC203	3	633	3	1212	3	1826	3	1605	4	8045	3	2847	3	
LRC204	3	1181	3	2369	3	3461	3	3634	3	13 719	3	5908	3	
LRC205	4	190	4	362	4	540	4	639	4	4416	4	498	4	
LRC206	3	318	3	640	3	951	3	445	3	4187	3	840	3	
LRC207	3	529	3	1063	3	1599	3	607	3	6569	3	2045	3	
LRC208	3	1208	3	2282	3	3499	3	4106	3	9664	3	7451	3	
LRC2	28	4405	27	8611	27	12 876	26	12 289	28	54 495	26	20 672	26	7376
Total	412	18 531	411	36 733	409	55 031	405	43 072	410	280 730	405	81 529	405	32 983

Table 4.12: Comparison of NV and CT achieved by the MetaheuristicAlgorithm to the best known solutions of Li and Lim [2001], Pankratz [2005a], Dergis and Dohmer [2008] and Ding et al. [2009]

4.8 Chapter Summary

This chapter has investigated both a tabu search heuristic and a branch and bound heuristic for the PDPTW. For the tabu search heuristic, the tabu tenure and stopping criteria have been determined and a novel tabu attribute has been introduced. After identifying that final improvements can be made to the solutions by improving the ordering of locations within individual routes, a branch and bound heuristic has also been developed.

It has been shown that the methods applied in this chapter and those introduced previously in Chapter 3 generate results which are competitive with state of the art results found in the literature. The results achieved obtain the best known solutions in 51 out of a possible 56 instances with the algorithm appearing to perform consistently well over all types of instance. A new best found minimum total travel distance is also achieved.

One of the main advantages of our approach is the speed of individual constructions. In this case, it has allowed us to produce large samples of solutions in times that are consistent with other approaches. This advantage can be exploited when applying these methods to the dynamic variant of the problem in Chapter 6.

The next chapter will review the literature for the dynamic VRP and in particular the dynamic PDPTW. Previous methods applied to solve the dynamic PDPTW will then be investigated to allow planning for the implementation of the algorithm, developed in this chapter, to a dynamic environment.

Chapter 5

The Dynamic PDPTW: A Literature Review

5.1 Introduction

Over the last decade there has been an increased interest in technologies that allow for real-time processing of information in the distribution industry. This is a result of the advances in communication and information technologies and an increase in e-commerce. E-commerce is the buying and selling of a product or service over electronic systems such as the internet and other computer networks. Due to these advances there is an increased demand from customers for a fast and flexible service.

Past research for the VRP has mainly concentrated on the static variant of the problem where all requests are known in advance and no uncertainty exists. The dynamic VRP (DVRP) requires planning methods to react to dynamically revealed information. Such problems are found in many transportation domains including courier services and dial-a-ride services. The DVRP has received much less interest in the literature compared to the VRP and in particular the dynamic PDPTW (DPDPTW) has received little attention, hence the reason for this research.

This chapter will provide an overview of the research for the DVRP and its variants followed by a detailed review of the methods applied to solve the DPDPTW. Section 5.2 provides an overview of the research for the DVRP. The variants of the DVRP are reviewed in Section 5.3 including the DPDPTW. The methods applied in the literature to solve the DPDPTW will be investigated in Section 5.4, including the key solution strategies which need to be determined when dealing with a real-time problem. Section 5.5 introduces measures to evaluate the degree of dynamism of an instance.

Section 5.6 examines the instances available in the literature for the DVRP and its variants, conclusions will be drawn on which instances may provide the most interesting for use in this research. The chapter is then concluded in Section 5.7.

5.2 The Dynamic VRP

The first reference to a DVRP is due to Wilson and Colvin [1977] who studied a single-vehicle DARP in which requests appear dynamically. Later, Psaraftis [1980] introduced the concept of *immediate request* where a customer requesting service always wants to be serviced as early as possible, which requires immediate re-planning of the current schedules.

A discussion into the possible methodological implications of the differences between static and dynamic vehicle routing was first introduced by Psaraftis [1988] and a DVRP, the so-called 'dynamic travelling salesman problem' was formulated.

The next step in the research of the DVRPs was made by Bertsimas and Van Ryzin [1991]. They introduced and analysed a model for stochastic and dynamic vehicle routing for a single vehicle with no capacity restriction. The time of arrival, location and on-site service of each request were stochastic. This was extended in Bertsimas and Van Ryzin [1993] for multiple capacitated vehicles. For more information on the main issues in the rapidly growing area of DVRPs in the early 1990's see Psaraftis [1995].

Work into the DVRP after the turn of the century included that of Ichoua et al. [2000] where a strategy for assigning requests which included diversion was presented. Diversion in this case consists of allowing a vehicle to be diverted away from its current destination, even if it is already en-route to that location, in order to serve a request that has dynamically arrived in the vicinity of its current position. The results for a tabu search algorithm are used to compare the new strategy, with and without diversion, for a dynamic problem motivated by a courier service application.

Ghiani et al. [2003] reviewed some of the existing literature on the DVRP with an emphasis on potential developments in the field of parallel computing. Real-time and time-dependent travel times as well as real-time demands are accounted for. Another algorithm, based on an ACO, to solve the DVRP was proposed by Montemanni et al. [2005]. It described how a DVRP can be seen as a sequence of static VRP-like instances, meaning the scheduling horizon can be split into a series of time intervals as adopted by Kilby et al. [1998]. Therefore, new requests received during each interval, are taken

into account only at the end of that interval. A mechanism to transfer information about good solutions from one static VRP to the following one is included as well as an advanced commitment time, where only requests within a specific next time are committed to. This will be explained in more detail in the following sections, paying particular attention to how it has been applied to the DPDPTW.

A comparative study between dynamic adaptive particle swarm optimisation (PSO) and variable neighbourhood search (VNS) for the DVRP was performed by Khouadjia et al. [2012]. In order to evaluate the dynamic performance of their approach, several indicators such as accuracy, stability and reactivity of the algorithm were measured. The accuracy measures the quality of the dynamic solution compared to the static solution and the stability refers to how consistent the algorithm is when the environment changes. It is interesting to note that the two methods performed differently based on the instance size: PSO behaved better in the smaller instances with VNS outperforming it in the two largest ones. This difference between results for varying instance type will be analysed in Section 6.5.

For more information on DVRPs, see a recent survey by Pillac et al. [2013] who classify routing problems from the perspective of information quality and evolution. More detail will now be provided for the available literature on the variants of the DVRP.

5.3 Variants of the DVRP

The first application of a tabu search heuristic to solve the dynamic VRPTW (DVRPTW) was that of Gendreau et al. [1996]. The application was motivated by the local operation of a long distance express courier service. Here, the algorithm is stopped and restarted each time a new request occurs and the general idea is to maintain a pool of good solutions known as the *adaptive memory*. This work is furthered in Gendreau et al. [1999] where requests with soft time windows must be dispatched in real-time to a fleet of vehicles. The tabu search heuristic is implemented on a parallel platform to decrease the computational time required.

The DVRPTW is studied more recently by Hong [2012], where 3 main problems are addressed. The first is when and how to decompose the problem into a series of static VRPTWs, the second is whether the static VRPTW can be quickly and efficiently solved within a given time, and finally, how to merge the latest known requests to the current solution. They propose a LNS, however, there are disadvantages of applying this method in a dynamic environment and these are highlighted. As LNS is a single

point search procedure, this means that the algorithm only retains one best feasible solution for each step. It is shown that a better solution would have been obtained if the search had made one or two 'wrong turns', i.e. the best known solution could have been found using the current second or third best feasible solutions at a point in time. In terms of dynamic vehicle routing this means that some locations deviating from a current best insertion position at a particular point in time, may achieve the best solution at the end of the scheduling horizon. This will be further investigated in Section 7.5.

Another problem, the dynamic DARP (DDARP), is studied by Teodorovic and Radivojevic [2000] where two approximate reasoning algorithms are developed. The first is used to make a decision about which vehicle a new request should be inserted in and the second determines how to incorporate the new request within the existing route. Both decisions aim to minimise the increase to the total travel time and the total distance travelled encountered by adding the new request.

A DDARP for a taxi service in a metropolitan area is investigated by Caramia et al. [2001]. A heuristic based approach is presented which iteratively solves a single-vehicle sub-problem to optimality. This is achieved by applying an exact dynamic programming approach, which is suitable as the capacity of the vehicle is small. The travel times used are deterministic and the minimum expected travel time is computed for any particular day and time which becomes the lower bound on the time window. A stretch factor specifies the maximum acceptable ratio between the actual and minimum expected travel time to create the upper bound of the delivery time window. The algorithm is set to update the current solution every time a new request arrives and local search techniques are used to find an optimal solution for the sub-problem defined by a single vehicle.

A number of parallel implementations of a tabu search heuristic previously developed for the static DARP are compared by Attanasio et al. [2004]. The algorithm works as follows; first a static solution is constructed on the basis of the requests known at the start of the scheduling horizon. When a new request arrives, the algorithm performs a feasibility check, if the request is accepted it is inserted and the algorithm tries to improve the current solution.

A dynamic DARP with time windows is considered by Coslovich et al. [2006]. It focuses on the case where a customer asks a bus driver, directly at a bus stop, whether they can be serviced. The main objective is to dynamically insert as many unexpected requests as possible in the previously planned routes. This is provided that the level of 'dissatisfaction' and deviation from the desired delivery time of advance requests is

minimised and that the excess ride time of both advance and unexpected requests does not exceed a given upper bound.

A recent study into waiting strategies for the dynamic DARP was proposed by Yuen et al. [2009]. They propose three waiting strategies, drive first (DF), wait first (WF) and modified dynamic wait (MDW). This work carries on from that of Mitrovic-Minic and Laporte [2004] for the DPDPTW which is discussed in more detail within the next section.

Real-life dynamic routing problems often differ to the standard variants studied in the literature. They often have idiosyncratic problem specific characteristics along with particular constraints that need to be satisfied. DRIVE (Dynamic Routing of Independent VEhicles) is a planning module discussed by Savelsbergh and Sol [1998] which is incorporated in a decision support system for the direct transportation of packages at Van Gend and Loos BV. This is the largest road transportation company in the Benelux, a union of states comprising three neighbouring countries in north-western Europe: Belgium, the Netherlands and Luxembourg.

The heart of DRIVE is a branch-and-price algorithm for the general PDP which uses construction and improvement algorithms to generate additional routes. In this case, a request can have one or more delivery locations and lunch and night breaks need to be considered. The planner focuses primarily on the short-term decisions within the next hour and, as soon as the plan is accepted, these become fixed. In the morning the planner focuses on the schedule for that day, and in the afternoon, the scheduler also focuses on work for the following day to enable the number of vehicles to rent for the following day to be determined.

The dynamic multi-period VRP (DMPVRP), where requests are considered over a multi-period time horizon, is considered by Wen et al. [2010]. The problem originates from a large distributor operating in Sweden and the objective is to minimise the total travel costs and waiting times, while balancing the daily workload over the scheduling horizon. Within the DMPVRP, for each request a set of consecutive periods during which delivery can take place are known, these can start as early as the day after the arrival of the request. The scheduling horizon is divided into a number of days and requests that have not been scheduled, including the new requests accumulated over a given day, are considered for scheduling the following day. A three-phase heuristic method is proposed where routes are constructed via VNS and post-optimised via a tabu search algorithm.

Early approaches to solve the DPDPTW starts again with the case for the single-vehicle and was explored by Jih and Yung-Jen Hsu [1999]. Here requests may change

during execution of the algorithm and the eventual execution of the route, therefore the objective function is to minimise a combination of total travel time and waiting time, with a penalty for any delays or an overloaded vehicle. A hybrid approach consisting of a GA and dynamic programming is explored. The approach starts by using dynamic programming to generate optimal routes, if these are not found within the allocated time between intervals, the partially constructed routes are passed to the GAs. The dynamic programming approach adopted is that proposed by Psaraftis [1980] and Psaraftis [1983a].

Other variants of the DVRP that have been considered, but are outside the scope of this research, include that of Li et al. [2009] who investigate a real-time vehicle re-routing problem with time windows. This is for the case where service undergoes disruption due to vehicle breakdowns. In such problems, one or more vehicles need to be re-routed in real-time. The problem was solved by a dynamic programming based algorithm that heuristically solved the shortest path problem.

A dynamic vehicle routing problem with multiple delivery routes is considered by Azi et al. [2012], here vehicles perform deliveries over multiple routes during the scheduling horizon. A LNS heuristic previously developed for the static problem is applied, with the decision being whether a new request is accepted or not.

More recently, Bouros et al. [2011] introduce the dynamic PDP with transfers. Requests can be transferred between vehicles meaning vehicles may have to deviate from their routes. Here satisfying a request is treated as a shortest path problem, from the location of the pickup to the location of the delivery of a request. The static PDP with transfers was considered by Cortés et al. [2010] and Mitrovic-Minic and Laporte [2006] and is discussed briefly in Section 2.7.

The next section of the review will describe in more detail the available literature on the DPDPTW and the methods adapted to solve it.

5.4 Methods Applied to the DPDPTW

This section of the literature review will examine in more detail the methods applied to solve the DPDPTW. There are many things to consider prior to the implementation of these methods, firstly the key basic solution strategies need to be decided upon. The recent survey of Berbeglia et al. [2010] provides a survey on the recent approaches taken to solve the DVRP, and in particular the DPDPTW, and describes the two main solution strategies for tackling the DPDPTW.

5.4.1 Dynamic Strategies

The first strategy involves solving a static problem each time a new request is recieved. However, one important drawback is that performing a complete re-optimisation, every time new information arrives, may be too time consuming and therefore inadequate for a real-time setting. The second strategy is where the static algorithm is applied only once at the beginning of the scheduling horizon to obtain an initial solution. When new information is received, the current solution is then updated using heuristic methods such as insertion heuristics sometimes coupled with a local search algorithm. In the intervals between time instants where new requests are received more robust optimisation methods might also be applied to the current solution, a tabu search heuristic as used by Gendreau et al. [2006] is most common. In Gendreau et al. [2006] the algorithms are re-started every time a new request arrives. The alternative, using a rolling horizon framework and the idea of splitting the scheduling horizon into time intervals, is used in Montemanni et al. [2005].

When adopting a rolling horizon framework, there is the option to use an advanced commitment strategy, as used in Montemanni et al. [2005] for the DVRP, which only plans a specific time interval into the future. The requests chosen to be inserted and the order in which they are inserted can be based on the amount of slack available, the time arrival of the requests, or their delivery time window. The standard solution methodology for the DPDPTW is the use of a rolling horizon framework (see Psaraftis [1988]).

Mitrovic-Minic et al. [2004] describe a double-horizon based heuristic to consider the impact of a decision both on a short-term and on a long-term horizon. In particular, how better managing slack time in the distant future may help to reduce the overall total distance travelled for the solution. Similar to this is the request buffering strategy of Pureza and Laporte [2008] which evaluates the viability of inserting the incoming request at a later time. If a request can be feasibly serviced in the next time interval, either by its insertion in an existing route or by its insertion in a new route, then it can be buffered, hence it is not inserted in that interval.

Decisions need to be made as to which requests in the current solutions are to be updated at each time interval. This could include all those which have not yet been serviced, as in Mitrovic-Minic et al. [2004]. For this case, all non-fixed requests are removed from the partly constructed solution and are then re-inserted along with the new requests in descending order of their slack time. For the case of the H1 heuristic of Pankratz [2005b] this simply inserts the new request into the partly constructed solution without making any change to the current ordering of the locations within

the route. These two methods are discussed in more detail in Section 6.4 where an example is provided.

There is also the possibility of transferring information about good solutions from one interval to the next. This is applied in Pankratz [2005b] who use an adaptive memory to store multiple solutions at each interval.

5.4.2 Insertion Heuristics

As was the case for the static counter-part, insertion heuristics are the most common way of constructing an initial solution for the DPDPTW and the general methods of insertion are similar to those used for the PDPTW. They are also the most common approach in the literature to insert new requests that arrive dynamically throughout the scheduling horizon but are adapted to the dynamic environment by defining a set of criteria for insertion.

Seven variations of a cheapest insertion procedure were used for testing the waiting strategies of Mitrovic-Minic and Laporte [2004]. Once again, the slack time of a request is equal to the difference between the total time still available to service the request and the direct travel time between its pickup and delivery locations.

The criteria investigated were as follows:

1. Insertion is performed immediately upon the arrival of a new request :
2. Insertion is performed at every 15 minute interval throughout the scheduling horizon :
   - **a)** all new requests are eligible for insertion,
     - **i)** requests are not sorted.
     - **ii)** requests are sorted by their slack time.
     - **iii)** requests are sorted by their delivery deadline.
   - **b)** only requests whose pickup deadline is within the next 2 hours are eligible for insertion,
     - **i)** requests are not sorted.
     - **ii)** requests are sorted by their slack time.
     - **iii)** requests are sorted by their delivery deadlines.

Two routing heuristics have been developed for the double-horizon based approach of Mitrovic-Minic et al. [2004]. The first is a constructive heuristic consisting of a cheapest insertion and re-insertion procedure. The cheapest insertion procedure is applied to new requests accumulated over a certain time period, followed by the re-insertion of all scheduled requests whose pickup location has not yet been serviced. Before insertion, the requests are sorted in increasing order of slack time.

Similarly, results for Pankratz [2005a] are also compared across two versions of an insertion heuristic. The first, H1, inserts each new request immediately after its arrival, without fundamentally changing the routes of the previous schedule. Where a portion of requests are static, an initial solution is generated by inserting all known requests in the order of slack time. The second, H2, performs a total revision every time a new request arrives. All requests, except for those which are fixed, are discarded and a new schedule is generated from scratch, by inserting all non-fixed requests in ascending order of their slack time.

Another method of inserting new requests is applied by Fabri and Recht [2006]. The approach is based on that of Caramia et al. [2001] for the DDARP, as outlined in the last section, where waiting times are now permitted. The algorithm updates the current solution any time a new request arrives. The request is successively assigned to each vehicle and the single-vehicle routing problem is solved. If a feasible solution for at least one vehicle is found, the request is accepted and assigned to the vehicle with the minimum additional cost, otherwise, it is rejected. It is shown this heuristic works well, as long as the number of requests assigned to one vehicle does not exceed a certain limit.

More recently a simulation tool developed for studying the performance of a large scale DPDPTW is introduced by Hyytiä et al. [2010]. Generally, the order in which requests are assigned to vehicles has a substantial effect on the performance of the sequential method. However, Hyytiä et al. [2010] find that, for large instances, it is sufficient to consider the insertion approach where the order of locations already inserted in a route are kept the same, without any significant loss in performance. These results suggest that freely exchanging locations between vehicles, achieves only a relatively small improvement in performance but a relatively large increase in computational effort. This will be investigated further in Section 6.4.

In general the results indicate the use of classical insertion heuristics are well researched in the area of DPDPTW. The next section will overview the improvement heuristics applied to the DPDPTW during the scheduling horizon.

5.4.3 Improvement Heuristics

The tabu search of Mitrovic-Minic et al. [2004] and Mitrovic-Minic and Laporte [2004] is a simplified version of the method introduced by Gendreau et al. [1998], with neighbourhoods defined by means of ejection chains. For the case of Gendreau et al. [1998] a request (with both its pickup and delivery location) is taken from one route and moved to another route, thus forcing a request from that route to move to yet another route, and so on. The process starts by selecting two requests i and j in routes r_i and r_j which are both removed from their current routes. Request i is then inserted into r_j and another request is chosen, the best insertion of request i in route r_j is calculated using an approximation function. The chain may be of any length and may be cyclic or not. That is, it may end with the insertion of the last request in the route that started the process (i.e. r_i), thus producing a cycle, or in some other route not yet included in the ejection chain (note that a chain of length 0 corresponds to a request being re-scheduled in its own route).

A greedy approach to examine all feasible insertion positions of inserting a request into a new route is applied by Mitrovic-Minic et al. [2004] and Mitrovic-Minic and Laporte [2004] to extend the method of Gendreau et al. [1998]. For the case of Mitrovic-Minic et al. [2004] the tabu search heuristic runs while requests are being accumulated. For Mitrovic-Minic and Laporte [2004] the tabu search heuristic may be used or omitted. If it is used, the insertion procedure is applied every k minutes and the tabu search runs while new requests are accumulated.

For the GGA of Pankratz [2005b], based on that of Pankratz [2005a], an initial population is generated using a combination of a random and greedy insertion heuristic as described in Section 2.8.1 and the best solution found so far constitutes the basis for the implementation process, until the occurrence of new requests. Every time a new request arrives, a snapshot of the physical transportation process is taken to determine the current state of execution. A solution is then obtained which contains the fixed parts of the original best solution.

For this case, a pickup is considered fixed in a route if its pickup location has been serviced, or a vehicle is en-route to service it, but its delivery location has not yet been serviced. A request is fixed if it has either already been serviced or a vehicle is currently on route to its delivery location. A straightforward synchronisation method is proposed, which tries to preserve as much of the grouping information originally encoded in the solution as possible. As a result, a solution is obtained which is fully adapted to the current state of execution, but still exhibits similarities to the previous solution.

In the case of Fabri and Recht [2006], whenever there is no new request to be assigned in a particular time interval, the LS procedure starts with the current solution and is executed until either a new request arrives or the search is completed.

For the case of Gendreau et al. [2006] each new request is inserted in every solution in the adaptive memory. The adaptive memory holds a number of partially constructed solutions from the previous interval and it is shown this quickly produces at least one solution of very high quality. A local descent is then applied to the best solution in the memory which stops at the first local minimum using the neighbourhood structure based on ejection chains. The tabu search heuristic used follows the general guidelines provided in Glover [1989], where an adaptive memory and a decomposition procedure are added to the basic scheme, to diversify and intensify the search. In the first case, a pool of routes, taken from the best solutions visited thus far is exploited to re-start the search in a new unexplored region of the search space. In the second case, the problem is decomposed to focus the search on smaller sub-problems. A parallel implementation was developed to increase the computational work performed between the occurrences of a new requests.

5.4.4 Waiting Strategies

Within the DPDPTW, the presence of time windows means that vehicles may have to wait at various locations along their routes. The solution quality may therefore be affected by the way the waiting time is distributed along vehicle routes. A way of taking future requests into account is the use of waiting strategies. In a dynamic context, it may sometimes be beneficial to wait at a location in anticipation of future requests. This could both reduce the overall distance travelled and maximise the probability of serving a request. Alternatively, the vehicle could move to another location from which future requests could be easily reached, for example a median computed on the basis of the locations and frequencies of the past known requests.

Theoretical and experimental results on different problems that have shown the importance of waiting strategies are, Mitrovic-Minic and Laporte [2004], Branke et al. [2005] and Ichoua et al. [2006]. Similarly, a buffering strategy consists of holding a request before assigning it to a vehicle route. Pureza and Laporte [2008] show the advantages of waiting and buffering strategies for the dynamic PDP with no capacity constraints.

It is identified by Mitrovic-Minic et al. [2004] that a route defined by the solution at a particular point in time, may be fixed for the next few hours. This is due to the fact that, the pickup location may be scheduled for service in the near future, while

the corresponding delivery location may be scheduled for service much later. Once the pickup of the request has been serviced, then the corresponding delivery location is fixed to that route. If the total travel distance is to be minimised, it is shown that it may be preferable to accumulate slack time in the distant future, rather than concentrating on minimising distance, since a longer slack time may make future request insertions easier.

Therefore the short-term goal of reducing travel distance is applied to the first portion of a solution and the long-term goal, a minimisation of a linear combination of distance and time, is applied to the portion of the solution in the distant future. Results show that this approach is superior to the standard rolling horizon (Psaraftis [1988]), often used in PDPTW algorithms.

The article of Mitrovic-Minic and Laporte [2004], compares four waiting strategies on a set of DPDPTW instances generated using real-life data. The solution methodology concentrates only on the scheduling aspect of the problem and is concerned with the design of a good waiting strategy, i.e. a way of organising and distributing the waiting time along a dynamically constructed route.

The first strategy considered is the drive-first (DF) strategy; this requires a vehicle to drive as soon as it is feasible. This is the most common strategy and is the only appropriate strategy to use within the static problem. The second is the wait-first (WF) strategy which requires a vehicle to wait at its current location for as long as is feasible. Vehicles waiting at their starting locations results in more requests being known at the time they do leave, so there is more potential for better routes serving these locations. The disadvantage is that WF requires more vehicles than DF because WF has a tendency to concentrate long waiting times in the first part of the route, therefore a new vehicle may need to be added to feasibly service the requests arriving later in the scheduling horizon.

The above are the two extreme cases of waiting strategies; the other two strategies considered by Mitrovic-Minic and Laporte [2004] are combinations of these and require the concept of dynamically partitioning a route. A partition can be represented as a service zone and a route may be represented by a series of service zones. The dynamic waiting (DW) strategy constructs a schedule for a fixed route by driving within each service zone according to DF. When the vehicle finishes serving all locations in that zone, the vehicle uses WF, before leaving for the next zone.

Finally, the advanced dynamic waiting (ADW) strategy of Mitrovic-Minic and Laporte [2004] propagates the total waiting time available in the route along the entire route. Within each service zone, the scheduling is the same as in DW; however, the waiting

time at the last location of a service zone is taken to be a portion of the longest feasible waiting time. In their work, the ADW strategy generated the best solutions with respect to total route length and number of vehicles.

Waiting and buffering strategies for the DPDPTW are also investigated by Pureza and Laporte [2008]. Their work focuses on two new strategies which lie between the two extremes of *drive-first* (DF) and *wait-first* (WF), as defined in Mitrovic-Minic and Laporte [2004]. The first prescribes DF if the current and next locations are close to each other in both time and distance, otherwise WF is employed. In the second strategy, a portion of the maximum feasible waiting time replaces WF. It forces a vehicle to arrive as early as possible at the next planned location, but no earlier than the lower bound of the time windows. As a result, routes may include waiting periods but only after service is completed.

Yuen et al. [2009] also extend the work of Mitrovic-Minic and Laporte [2004], introducing the MDW strategy. This differs from the standard WF strategy in that a vehicle now departs from a location where service is complete such that service at the next location can begin immediately upon arrival. It is shown that MDW requires fewer vehicles and provides a shorter total travel distance compared to DF and WF in the instances they consider. The next section of this chapter overviews further examples of DVRPs.

5.4.5 Further Topics

Further variants of the DVRP and the DPDPTW which have been considered in the literature but are outside the scope of this thesis include stochastic variants of the problem. Sometimes, some information is known and a probability distribution can be estimated through the use of historical data.

A stochastic and dynamic model for the single-vehicle PDP is developed by Swihart and Papastavrou [1999]. It is assumed that new requests arrive according to a Poisson process. If it is generally known that new requests will arrive at a certain time, it may be worth modifying the objective function to allow for this. The dynamic vehicle routing and dispatching problem investigated by Ichoua et al. [2006] exploits probabilistic knowledge about future requests. Here, a vehicle can wait at a location if the probability that a request occurs in a predefined neighbourhood during a set time interval is greater than or equal to a given threshold.

Another development is the use of time-dependent travel times for more accurate schedules, as used in Chen et al. [2006]. This adds further difficulties to the problem. Sáez

et al. [2008] develop a family of solution algorithms which consider future demand and predict expected waiting and travel times for requests. More accurate schedules are also considered by Ghiani et al. [2009] who describe and assess anticipatory algorithms taking into account the fact that decisions made at an early stage of the scheduling horizon might affect the ability to make good decisions at a later stage.

The option of refusing a request if it is not feasible to insert it into the existing solution is explored by both Caramia et al. [2001] and Fabri and Recht [2006].

The allowance of vehicle diversion, while making the driver operations more complex, could be beneficial. Ichoua et al. [2000], devise a diversion algorithm for a DVRP and experimental results show that the modified algorithm is able to reduce the number of un-served requests and the total distance travelled when compared to the original heuristic. However, the allowance of diversion brings some technical difficulties, such as how the distances are calculated as well as the problem of estimating the time allocated for the improvement procedure.

The next section is concerned with how to evaluate the success of an algorithm in a dynamic environment.

5.4.6 Success of Algorithms for the DPDPTW

Evaluating the success of heuristics applied in a dynamic environment in the literature has generally been achieved by making comparisons with the results achieved in a static environment. The *value of information* is a measure of the effectiveness of a heuristic and it is calculated as a percentage difference in the cost from the solution obtained by applying the classical single-horizon based heuristics for the static problem compared to that achieved in a dynamic environment. This will be applied to assess the quality of our algorithm in Chapters 6 and 7.

Some interesting results achieved in the literature are as follows. Mitrovic-Minic et al. [2004] show the use of a double-horizon can improve solution costs when compared with classical single-horizon methods, but percentage improvement tends to go down as instances become larger.

Results show that solution quality for all 3 algorithms adopted by Pankratz [2005b] decreases as the instances become more dynamic. The GGA performs significantly better than H1 and H2, although this advantage diminishes with an increasing degree of dynamism. This is due to the fact that there is less slack time available in the routes for the GGA to improve the solution. An increasing degree of urgency results in more

requests having to be served shortly after they arrive so they quickly become fixed in a solution and cannot be re-planned.

Yang et al. [2004] state that more flexible revision plans consistently yield better results, this agrees with the results of Pankratz [2005b] who show their H2 heuristic is dominant over H1.

In Gendreau et al. [2006] it is found that neighbourhood search heuristics are more effective than simple dispatching rules based on insertion methods, even under the stringent time pressure conditions. However, when the request arrival rate is increased, the benefits associated with a sophisticated search framework diminish. With more requests per route, more opportunities for improvement are available; however, these cannot be exploited due to a lack of computational time between the occurrence of new requests.

In the case of Fabri and Recht [2006] it is found that, the fewer vehicles that are available and the more requests that arrive, the LS is able to achieve greater improvements to the initial solutions. This is due to the fact that there is a larger search space for the LS and hence more opportunities for improvement.

The next section will outline the measures of dynamism introduced in the literature, this is a characteristic introduced to distinguish between problem instances in a dynamic environment.

5.5 Measure of dynamism

Different problems or instances of different problems can have different levels of dynamism, which can be characterised according to two dimensions. The frequency of changes and the urgency of requests. The former is the rate at which new information becomes available, while the latter is the time gap between the disclosure of a new request and it's expected service time. From this observation three metrics have been proposed to measure the dynamism of a problem or instance.

Lund et al. [1996] define the *degree of dynamism* δ as the ratio between the number of dynamic requests n_d and the total number of requests n_{tot} as follows:

$$\delta = \frac{n_d}{n_{tot}} \tag{5.1}$$

Based on the fact that the time stamp of a request is also important Psaraftis [1988],

Psaraftis [1995] and Larsen [2001] proposed the *effective degree of dynamism* δ^e. The metric can be interpreted as the normalised average of the time stamps. Let T be the length of the scheduling horizon, R the set of requests, and t_i the disclosure time of request $i \in R$. Assuming that requests known beforehand have a disclosure time equal to zero, δ^e can be expressed as:

$$\delta^e = \frac{1}{n_{tot}} \sum i \in R \frac{t_i}{T} \tag{5.2}$$

To reflect the *level of urgency* of a request, Larsen [2001] extended the $effective\ degree\ of\ dynamism$ to problems with time windows. He defines the *reaction time* as the difference between the disclosure time t_i and the end of the corresponding time window, l_i, highlighting that longer reaction times means more flexibility to insert the request into the current routes. Thus the effective degree of dynamism measure is extended as follows:

$$\delta^e_{TW} = \frac{1}{n_{tot}} \sum_{i \in R} 1 - \frac{l_i - t_i}{T} \tag{5.3}$$

The next section will summarise the instances available for the DVRP and its variants.

5.6 Instances Available for the DVRP

We suggest, based on the literature review, that the amount of research that has presently been undertaken into the DPDPTW is limited.. Table 5.1 summarises the instances available for the DVRP and its variants, the information has been taken from Pankratz and Krypczyk [2009].

It can be seen that there are 4 main sets of instances available in the literature that fit our problem description. These are the instances of Gendreau et al. [1998], Mitrovic-Minic et al. [2004], Mitrovic-Minic and Laporte [2004] and Pankratz [2005b]. These sets of instances will be examined in more detail to determine which would be most suitable for use in our research.

The main advantage of the instances of Gendreau et al. [1998] is how closely they follow what is observed in a real-world environment in a pickup and delivery service based in Montreal. The main drawback of these instances is that only 15 instances are available by Gendreau et al. [1998] and it is felt that the instances therefore lack variation and depth and would not provide an interesting range of scenarios to investigate. Another

drawback is that Gendreau et al. [1998] fix the number of vehicles within their solutions, this differs to the problem specification and algorithms we have previously applied to the PDPTW in Chapters 3 and 4 . Hence, a direct comparison with the results provided by Gendreau et al. [1998] may not be possible.

Author	Problem type	Comment	Problem size
Gendreau et al. [1998]	DPDPTW	Fixed fleet size.	15 instances generated. Up to 200 requests.
Montemanni et al. [2003] and Montemanni et al. [2005]	DVRPTW		44 instances from Kilby et al. [1998]. 50-199 requests.
Attanasio et al. [2004]	DDARP	Information about request arrival was calculated by the program and not stored in the problem files.	26 instances. 20 instances from Cordeau and Laporte [2003], 24-144 requests. 6 instances from real life large scale problems.
Lackner [2004]	DVRPTW	Varying degrees of dynamism.	56 instances derived from Solomon [1987]. Up to 1000 requests.
Mitrovic-Minic and Laporte [2004]	DPDPTW		40 instances generated. 100, 300, 500, 1000 requests.
Mitrovic-Minic et al. [2004]	DPDPTW		90 instances generated. 100, 500, 1000 requests.
Branke et al. [2005]	DVRP		7 instances derived from the OR Library Beasley [1990]. Up to 1000 requests.
Pankratz [2005b]	DPDPTW	Instances with varying degrees of urgency and ex-ante knowledge.	5,600 instances derived from Li and Lim [2001]. 50-55 requests.
Fabri and Recht [2006]	DDARP		300 instances from Potini and Viola [2002]. 100, 400, 1000 requests.
Chen et al. [2006]	DVRPTW	Time dependent travel times.	56 instances derived from Solomon [1987]. 100 requests.

Table 5.1: Summary of the Instances available in the literature for the DVRP

With regards to the instances of Mitrovic-Minic et al. [2004] and Mitrovic-Minic and Laporte [2004], the data collected is from two medium-to-large courier companies operating in Vancouver. There are a range of instances with a varying number of requests and widths of time window. The only limitation here is that the problem does not require a vehicle to return to the depot at the end of the scheduling horizon. As we will see our algorithm can be easily adapted to take this difference into account making direct comparisons with the published results possible. The instances of Mitrovic-Minic

and Laporte [2004] are not freely available; hence these are not able to be investigated in this research.

The instances of Pankratz [2005b] have the largest variation of instances as they provide both varying degrees of urgency and dynamics to be investigated. The main advantage of these instances is that a direct comparison can be made to the results achieved in Chapter 4, for the static variant of the problem. A disadvantage is that all instances with varying degrees of dynamics have been generated with the highest degree of urgency. This may limit the opportunity to investigate improvement heuristics, as each new request arriving to the system is arriving at the latest possible time such that it can still be feasibly serviced. Therefore as soon as the request arrives it needs to be scheduled within a route.

In conclusion, the instances of both Mitrovic-Minic et al. [2004] and Pankratz [2005a] provide opportunities to investigate the DPDPTW and provide comparison with best known results. In particular, the results of Pankratz [2005a] will allow further research into varying degrees of dynamism and urgency of requests and those of Mitrovic-Minic et al. [2004] will allow further analysis into the insertion and improvement heuristics for the problem.

5.7 Chapter Summary

This chapter has introduced the research into the DVRP and its variants. It has detailed the research to date for the DPDPTW and overviews the methods proposed to solve the problem. It has described ways in which to evaluate solutions in a dynamic environment which can be applied in this research. An overview of the instances available for the DVRP has been summarised including for the first time an evaluation of those instances available for the DPDPTW.

It is apparent that many key decisions need to be made when adapting our algorithm to a dynamic environment and these will need to be investigated in more detail. These include deciding how to incorporate new requests, when to update the current solution and how the current solution should be updated.

With this in mind, the next chapter will look to begin our research into the dynamic problem for the instances of Pankratz [2005b]. A direct comparison can be made to the results achieved in Section 4.7.

Chapter 6

Adapting Our Algorithm to the DPDPTW

6.1 Introduction

The dynamic VRP (DVRP) is where planning methods need to react to dynamically revealed information. This is where not all information is known in advance and so schedules need to be updated during the scheduling horizon. Such problems are found in many transportation domains, such as pickup and delivery courier services and dial-a-ride services. See Gendreau and Potvin [1998] and Berbeglia et al. [2010] for further examples of real-time transportation problems.

Based on the definition by Pankratz [2005b], the dynamic PDPTW (DPDPTW) has the following change in comparison to its static counterpart, which has been outlined in Section 3.1. Requests are now not completely known in advance but become available during the scheduling horizon. Each request now has a time stamp associated with it equal to the time at which it becomes known to the system. The scheduling of a request can therefore not take place until after this corresponding time stamp has been reached in the scheduling horizon. The arrival of requests is the only source of dynamics that we will consider within this research. Others include real-time variations in travel times between locations and vehicle breakdowns (see Chen et al. [2006], Xiang et al. [2008] and Mu et al. [2010]).

Past research into the DPDPTW has considered splitting the problem into a series of sub-problems; this is carried out sequentially by taking each sub-problem as a static variant of the problem. These sub-problems can then be solved using a static algorithm, which can be updated at specified intervals, or in some cases, each time a new request

is received. The current schedule is then updated so the algorithm takes into account all information that has become known up until that time. The idea of updating the schedule over a rolling horizon framework was first introduced by Psaraftis [1988] and is one we will adopt in this thesis.

The rest of this chapter is structured as follows. Section 6.2 will provide information on the instances of Pankratz [2005b] to be investigated within this chapter. The main advantage of this set of instances is that they are generated from the static PDPTW instances of Li and Lim [2001] and adapted to a dynamic environment. Therefore a direct comparison can be made with the results achieved in Chapter 4 for the static PDPTW.

Section 6.4 looks to adapt the insertion methods explored in Section 2.8.1 for the static problem, both to construct an initial starting solution and to insert the requests that arrive dynamically throughout the planing period. Results for varying methods of insertion will be summarised in Section 6.5, both for instances with varying proportions of dynamic requests and varying urgency of dynamic requests.

Methods to improve on the solutions obtained by the insertion heuristics are explored in Section 6.6 and are based on those applied to the static PDPTW in Chapter 3 and Chapter 4. Comparisons with the results of both Pankratz [2005b] and those obtained by our static algorithm will be provided in Section 6.7. Finally, the chapter is concluded in Section 6.8.

6.2 DPDPTW Instances of Pankratz [2005b]

This section introduces the DPDPTW instances generated by Pankratz [2005b] that are investigated in this chapter. A total of 5,600 instances were generated for the DPDPTW by Pankratz [2005b] which are derived from the 56 static PDPTW instances of Li and Lim [2001] (see Section 3.3). The requests that arrive during the scheduling horizon will be referred to as dynamic requests and those known prior to the beginning of the scheduling horizon as static requests.

Two sets of instances were created by Pankratz [2005b], each with varying properties. They contain exactly the same information as those generated by Li and Lim [2001] such as number of requests, location of requests and time windows, but contain an initial column of data containing the time stamp of each request. This is generated in two ways based on the two different sets of instances. In both cases if a request is selected to be a dynamic request it is allocated a time stamp. This is a proportion of

the time left available to feasibly service that request based on its time window and location from the depot.

The first set contains instances that have dynamic requests with varying degrees of urgency. The urgency of a dynamic request refers to the amount of time left available to feasibly service that request once it has become known to the system. This set of instances is referred to as P1 and has instances ranging from 10% urgent to 100% urgent. To calculate the urgency of a request, the latest possible arrival time, such that the request can still be feasibly serviced by a new vehicle, needs to be determined.

Following the notation from Section 3.2, for each request r, let the latest possible arrival time be denoted as t_r^{latest}. Let v_i and v_j be the pickup and delivery locations of request r respectively. Then l_i and l_j are the latest times that service can take place at each of these locations and s_i and s_j are the service times at each location. Again, v_0 denotes the depot and t_{ij} is the travel distance from location v_i to location v_j, where distance is equal to time.

Therefore t_r^{latest} can be defined as:

$$t_r^{latest} := \min\{l_i, l_j - t_{ij} - s_i\} - t_{0i} \tag{6.1}$$

Let a be the degree of urgency, varying from 0.1 to 1 in steps of 0.1, then each request was allocated a time stamp, t_r, where $t_r = a \times t_r^{latest}$. Therefore 10 instances for each of the 56 static PDPTW instances were generated, giving a total of 560 instances generated in the set P1. These instances have no requests known in advance, as all requests are allocated a time stamp as above. However for some requests, $t_r^{latest} = 0$, therefore they are allocated a time stamp equal to 0, and can be treated as a static request.

The second set of instances, P2, have varying proportions of dynamic requests. These range from instances having 10% of requests being known in advance of the scheduling horizon to 90% of the requests being known in advance; here all have the highest degree of urgency. These were generated by increasing the proportion of requests known in advance, q, from 0.1 to 0.9 in steps of 0.1.

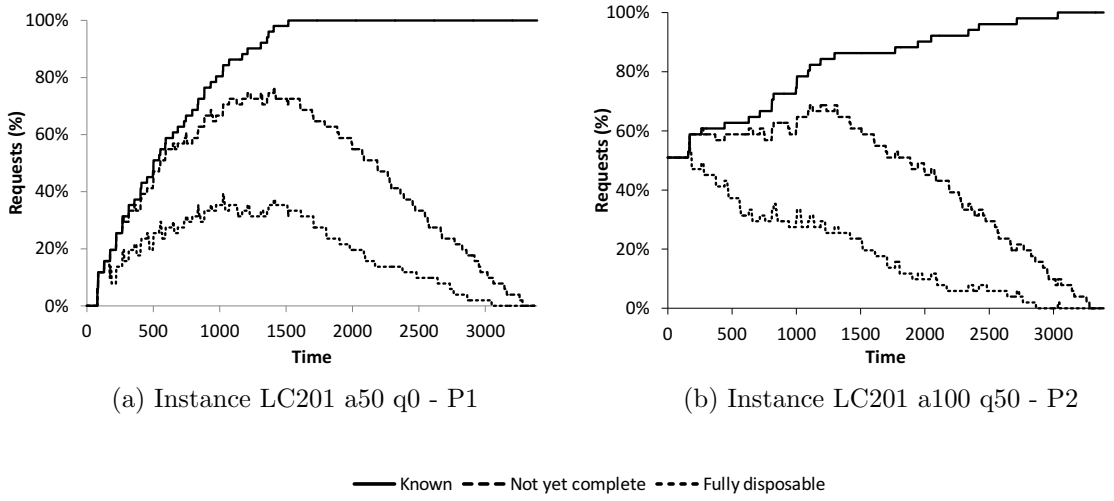
In order to reduce the risk of stochastic bias in deciding which requests should be static ($t_r = 0$) and which should be dynamic ($t_r = t_r^{latest}$), Pankratz [2005b] generated 10 dynamic instances for each q. Therefore a total of 5,040 instances were generated in set P2, however it was felt that this number of instances is not practical for use in this research. Therefore only one of the 10 randomly generated for each instance will be

examined, therefore a total of 504 instances. This is similar to the number analysed for each degree of urgency in set P1. In Section 7.7 it is shown that selecting a subset of these requests still generates results which are comparable to those of Pankratz [2005b]. Summary information for each set of instances is provided in Table 6.1.

Set	Degree of urgency (a)	Proportion of requests known in advance (q)	Number of DPDPTW instances generated
P1	0.1 to 1 in steps of 0.1	0	560, 1 for each instance, for each a
P2	1	0.1 to 0.9 in steps of 0.1	5040, 10 for each instance, for each q

Table 6.1: The DPDPTW Instances of Pankratz [2005a]

The characteristics of each set of instances are shown in Figure 6.1. The instance 'LC201 a50 q0' shown in Figure 6.1a is taken from the set P1 and has a medium degree of urgency at 50% and has 0% of requests known in advance. The instance 'LC201 a100 q50' in Figure 6.1b is taken from the set P2 and half of the requests are known at the beginning of the scheduling horizon, with these requests having the highest degree of urgency, at 100%. Both figures show the percentage of requests known to the system over the scheduling horizon, and of these, the proportion of requests not yet completed, and the proportion of requests which are fully disposable.



(a) Instance LC201 a50 q0 - P1

(b) Instance LC201 a100 q50 - P2

Figure 6.1: Comparison of instances from set P1 and P2

Note that the percentage of requests not yet complete includes all requests whose delivery time window has not passed. The percentage of fully disposable requests includes all requests whose pickup time windows has not yet passed, hence, the request is still 'fully disposable' in the solution.

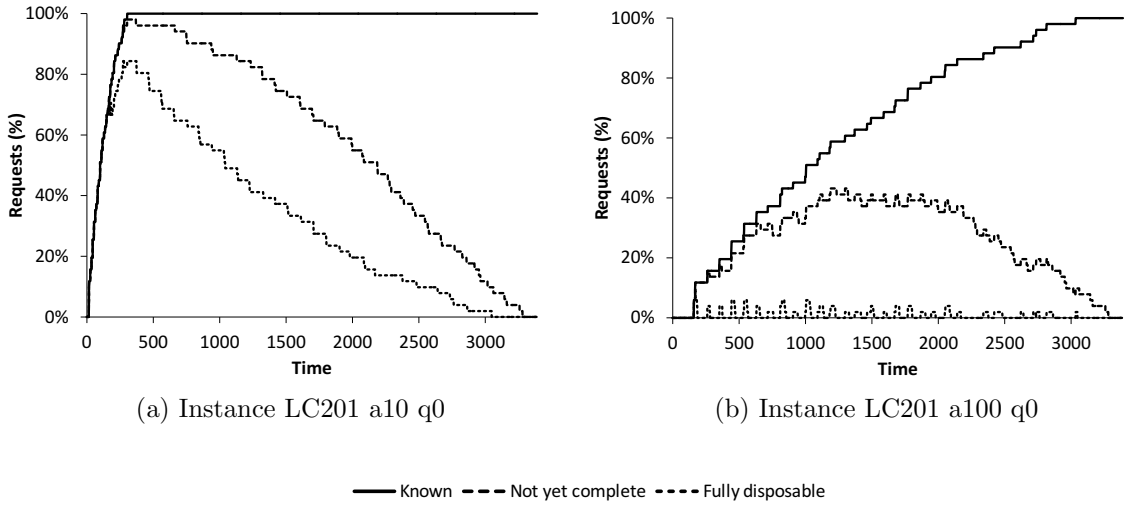
It can be seen, under the conditions of P1 shown in Figure 6.1a, that the number of fully disposable requests first increases and then decreases, with an average of around 20% throughout the scheduling horizon. However, under the conditions of P2 shown in Figure 6.1b, the number of fully disposable requests starts at its maximum, at 50%, and then decreases throughout the remainder of the scheduling horizon.

The main objective when considering the instances in set P1 will be how best to incorporate the dynamic requests and how this may change under varying degrees of urgency. It will therefore be the insertion of the dynamic requests and the improvement heuristics that will play an important role here.

For the case of set P2, only a portion of the requests will be dynamic, but will be arriving with the highest degree of urgency. Therefore each dynamic request will need to be inserted into the solution immediately and will become fixed to the vehicle to which it is assigned. This could limit the opportunities for both dynamic insertion and improvement heuristics. The initial solution of the static requests is therefore important in this case, with regards to how it can best incorporate the urgent requests arriving dynamically.

Looking at both sets of instances in more detail, the extremes for both cases will now be considered. The characteristics of the extreme instances within set P1 are shown in Figure 6.2. It is clear that in the case where no requests are known in advance, as shown in Figure 6.2b, there is a limited number of fully disposable requests throughout the scheduling horizon. This is due to the fact that requests are arriving at t_r^{latest} and therefore have to be assigned to a vehicle where service can begin immediately. Therefore the line on the graph line is simply showing the number of requests that arrive at each time instant during the scheduling horizon.

From Figure 6.2a the number of requests not yet completed increases to begin with and then starts to decrease again. This is due to the fact that, although requests arrive at the highest degree of urgency, some of these requests will only have an urgent pickup time window and not an urgent delivery time window. These will therefore not be completed immediately, but throughout the scheduling horizon. Therefore some opportunity could be available for improving the solution by re-ordering the delivery locations within a route.



(a) Instance LC201 a10 q0

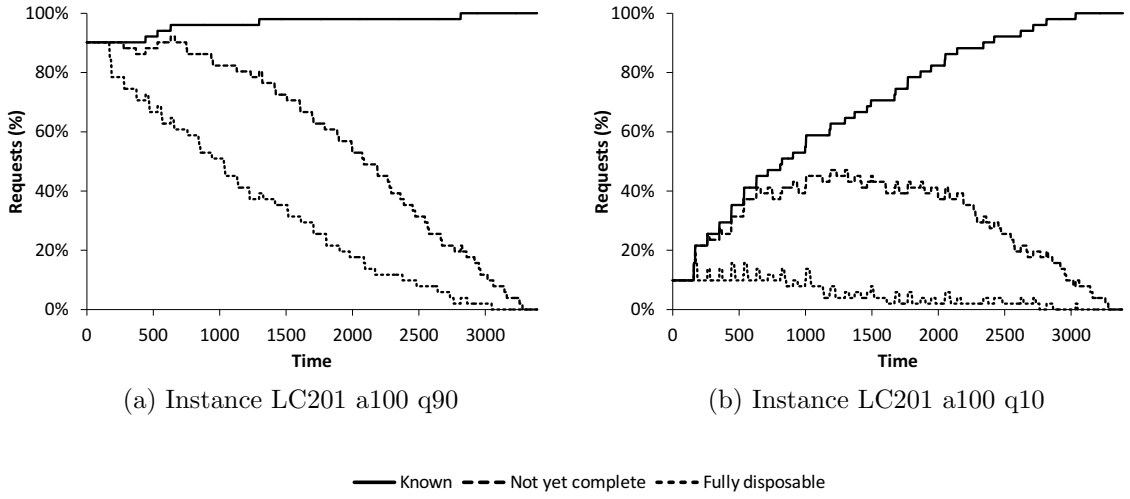
(b) Instance LC201 a100 q0

Figure 6.2: Comparison for varying Urgency - P1

With regards to Figure 6.2a this shows the case where the dynamic requests have the lowest degree of urgency, meaning that all requests arrive early on in the scheduling horizon. This then decreases consistently throughout the scheduling horizon as requests are serviced as their time windows are reached. All of the planning for this case will be completed early on in the scheduling horizon, as all information will quickly become known. This could result in limited opportunities to improve the solution over time.

For the examples shown in Figure 6.3, all requests have the highest degree of urgency, this means that all dynamic requests will need to be inserted immediately into the solution after their arrival. These requests will then be fixed to the vehicle to which they are assigned and it will not be possible to re-insert them into another route. However, the position and time of service for their delivery location may still be improved. With regards to the static requests, they can be inserted into a new vehicle until their value of t_r^{latest} is reached, so there may still be opportunities for improvement here.

It is clear from Figure 6.3b that, once again, if the dynamic requests are arriving throughout the scheduling horizon with the highest degree of urgency, there is a limited number of fully disposable requests. Hence there is likely to be little opportunity for improvement again. Also, as no requests are known in advance, there is no opportunity for the initial insertion heuristics to create a good initial solution. However, for the case shown in Figure 6.3a, where there is a high proportion of requests known in advance, there is a high percentage of requests that are not yet complete or fully disposable at this point in time. Therefore a good initial starting solution could become critical with little opportunity for improvement then being available over time.



(a) Instance LC201 a100 q90

(b) Instance LC201 a100 q10

Figure 6.3: Comparison for varying Proportions of Dynamic Requests - P2

It is clear that for the instances with a high degree of urgency and a high proportion of dynamic requests that the opportunities for improvement within a solution, especially through *intra-route* moves could be limited. However, to investigate the arrival of requests in a dynamic environment and the behaviour of heuristic methods to improve these solutions, the instances with a high proportion of dynamic requests but a low degree of urgency appear to provide the most interesting points of analysis.

The next section will look to re-introduce the insertion heuristics covered in Section 2.8.1 for the static PDPTW. The insertion heuristics will be applied to generate an initial solution for the static requests and then will be adapted to insert the dynamic requests.

6.3 Constructing Initial Solutions

The procedure to be followed in adapting our algorithm to a dynamic environment is outlined in Figure 6.4, it shows that at the start of the scheduling horizon an initial solution is constructed comprising of the static requests. The first step in solving the DPDPTW is therefore to decide how to assign the static requests to vehicles, if some are known, at the beginning of the scheduling horizon.

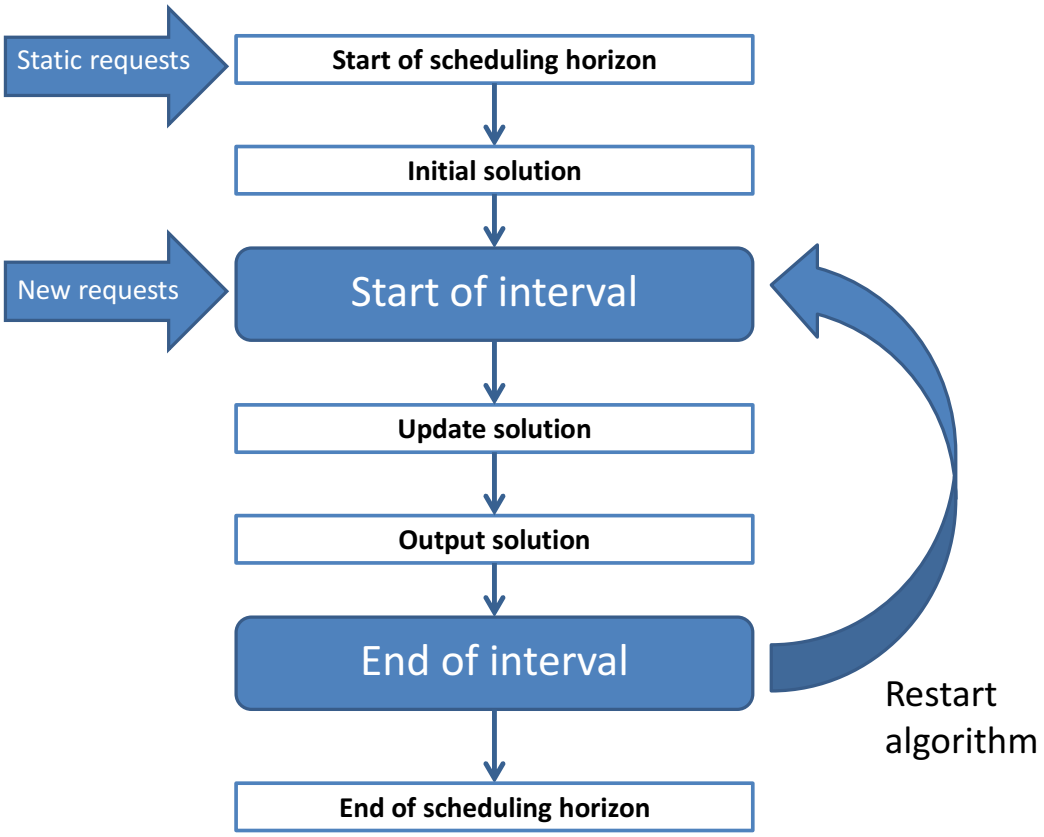


Figure 6.4: Structure for solving the DPDPTW

Recall that in our static algorithm, an initial feasible solution was constructed using a combination of a random and greedy heuristic. Results showed that the *random* insertion method outperformed others from the literature including the *greedy* insertion method of Nanry and Barnes [2000] and the *slack* insertion method of Pankratz [2005b]. These 3 insertion methods will now be be applied to construct an initial solution for the static requests. We will refer to the random heuristic applied in Section 3.4 once again as *random*, the greedy insertion heuristic of Nanry and Barnes [2000] as *greedy* and the slack insertion heuristic of Pankratz [2005b] as *slack*.

From Figure 6.4, the next step in adapting our algorithm to a dynamic environment is deciding how best to incorporate the dynamic requests, this will be considered in the following section. For the case of Pankratz [2005b], the solution is updated each time a new request arrives, therefore an interval is simply the time between the occurrence of each new request.

6.4 Dynamic Insertion Heuristics

There are many ways to deal with the insertion of requests that arrive during the scheduling horizon. Pankratz [2005b] compares the results for three methods. The results for their genetic algorithm (GA) are compared against two insertion heuristics.

For information, a location in a route is considered to be fixed at time t, if the vehicle servicing that location has already done so, or has already left the proceeding location at time t (i.e. is currently en-route to the location). A request is considered completed at time t if both the pickup location and the delivery location are fixed in a route at time t, whereas it is called active, if at time t its pickup location is fixed but its delivery location is not yet fixed. For this research, a pickup location is also considered fixed if the location can no longer be serviced by the introduction of a new vehicle, i.e. it can only be feasibly serviced by its current vehicle or another in close proximity.

The first of the dynamic insertion heuristics considered by Pankratz [2005b] is denoted H1. It inserts each new request immediately upon its arrival and this is carried out without fundamentally changing the current solution. The second dynamic insertion heuristic, H2, completes a total revision of the current solution each time a new request arrives. This excludes those requests that are already fixed to the current position within a route. For both of these methods, at each new insertion the requests are inserted again in order of t_r^{latest} (see Section 6.2).

For the GA of Pankratz [2005b], every time a new request arrives, the route position of all locations which have already been served at time t, are fixed. All requests which are still free for re-planning, i.e. those which are not fixed nor active, are deleted. As a result, a truncated solution is obtained which contains the irreversible parts of the original best solution. The GA then re-inserts the deleted requests along with the new requests, attempting to preserve as much of the information, regarding the grouping of requests, originally encoded in the solution.

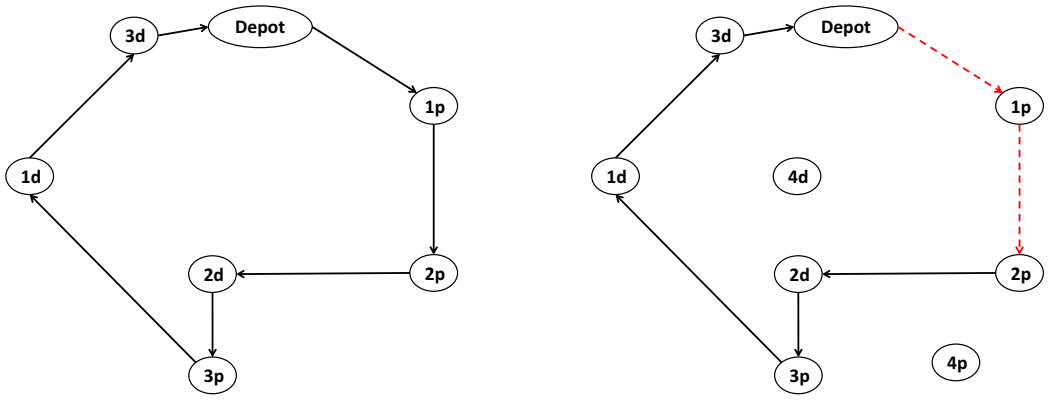
Based on the above dynamic insertion methods of Pankratz [2005b], three methods of inserting dynamic requests will be investigated in this research. However, three criterion will also be considered for the re-insertion of the requests, and not just the *slack* method as was the case for Pankratz [2005b].

The first of the dynamic insertion methods will be a simple insertion method which is the same as H1. Here, each request is inserted immediately upon its arrival into the feasible position, with the minimal increase in total distance travelled, in the already constructed solution. No change is made to the ordering of the requests in the existing

solution.

The second method of dynamic insertion is the same as H2, being the re-insertion of all non-fixed requests (NFR). At each interval, this heuristic first removes all requests which are not fixed within a route of the current solution. Each one of these non-fixed requests can then be re-inserted along with the new requests, as in the simple insertion method.

The third method, which extends the H2 method of Pankratz [2005b], is the re-insertion of all non-fixed locations (NFL). It is known that a request is fixed to a route if its pickup location has already been serviced (or is currently being serviced), however the delivery location may still be free for re-planning within that route. Therefore the delivery location of that request is fixed to a route, but is not fixed to a position within that route. For this example, at each interval, all locations that are not fixed within a route are removed. All locations are then re-inserted along with the new requests. However to ensure the constraints of the problem are still met, the delivery locations of fixed requests must be returned to the same route from which they were removed, but not necessarily to the same position.



(a) Before the arrival of a new request

(b) After the arrival of a new request

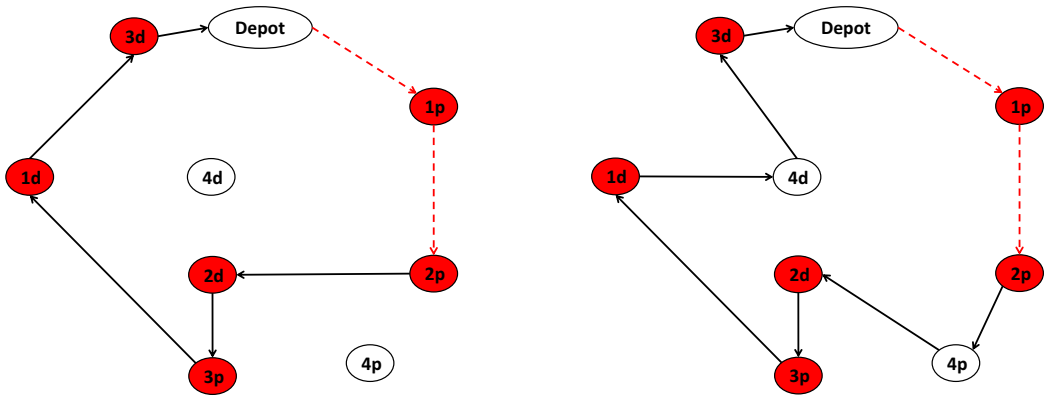
Figure 6.5: A simple solution during the scheduling horizon

Figure 6.5 shows a snapshot of a solution both at the beginning of the scheduling horizon and then after the arrival of a dynamic request, at time t. Figures 6.6, 6.7 and 6.8 subsequently show the effects of applying the 3 dynamic insertion heuristics introduced in this section to incorporate the dynamic requests into the existing solution.

The solution prior to the arrival of the new request has 3 requests all serviced by a single vehicle. At time t, when the new request arrives, the vehicle is currently servicing the pickup location of request 2, hence this location and anything prior to it are considered

fixed. This also means that both the delivery locations of request 1 and 2, are fixed to this route. A dashed red line between two locations indicates the section of the route is fixed.

Figure 6.6 shows how the application of the simple insertion method incorporates the new request. Here the request is inserted into the best feasible position in the already constructed solution, found in Figure 6.6a. For this case all locations are considered fixed, as no change can be made to the current ordering of the locations in the route. These fixed locations are therefore highlighted in red. The solution after the dynamic request has been incorporated using the simple insertion method can be found in Figure 6.6b.

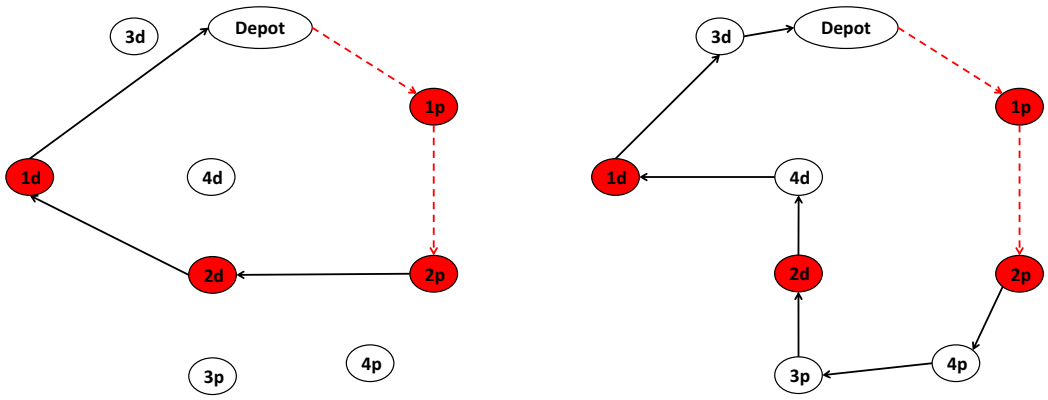


(a) Before insertion

(b) After insertion

Figure 6.6: Simple Insertion Heuristic

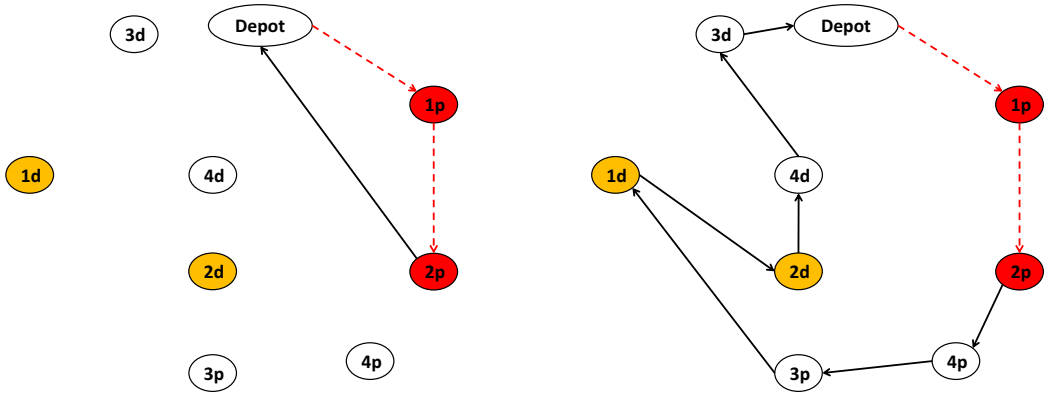
Figure 6.7 provides an example of applying the non-fixed request insertion method to incorporate the dynamic request. This method first removes all the non-fixed requests from the solution. As the pickup location of requests 1 and 2 have been serviced or are currently being serviced, these requests are considered fixed. Therefore, request 3 is the only non-fixed request and is therefore removed from the solution, as shown in Figure 6.7a. When re-inserting request 3 and inserting request 4, based on the insertion criterion chosen, this could allow a change in the current ordering of the route. The solution formed after the dynamic request has been incorporated using the non-fixed request insertion is shown in Figure 6.7b and differs from that obtained by the simple insertion method.



(a) Before insertion (b) After insertion

Figure 6.7: Non-fixed Request Insertion Heuristic

Finally, Figure 6.8 provides an example of a solution obtained after incorporating the dynamic request using the non-fixed location insertion method. Here, all non-fixed requests and the locations that are still active are removed from the solution. For this example this again includes request 3 as this not fixed to the route, but it also includes the delivery locations of requests 1 and 2, as these have not yet been serviced. These locations are highlighted in orange and are shown in Figure 6.8a. This might allow further re-ordering of the locations still to be serviced within the route, as shown in the solution achieved in Figure 6.8b.



(a) Before insertion (b) After insertion

Figure 6.8: Non-fixed Location Insertion Heuristic

For each of these cases, the dynamic request and any of the removed requests can be inserted in any route, including a new route. As before, for this research the insertion that results in the smallest increase in cost is accepted. It should be noted that for the

case of the non-fixed locations method, where the delivery location of a request is not fixed but the pickup location is, the delivery location would only be able to be inserted into the same route, to ensure feasibility. Therefore it must be checked before inserting a new request into a route, if all remaining active delivery locations to be re-inserted into that route can still feasibly be inserted along with the new request.

It is clear that inserting one request into a single route at a particular point in time under different insertion methods creates varying solutions. The next section will present the results for each of these insertion methods using each of the 3 insertion criterion detailed in Section 6.3.

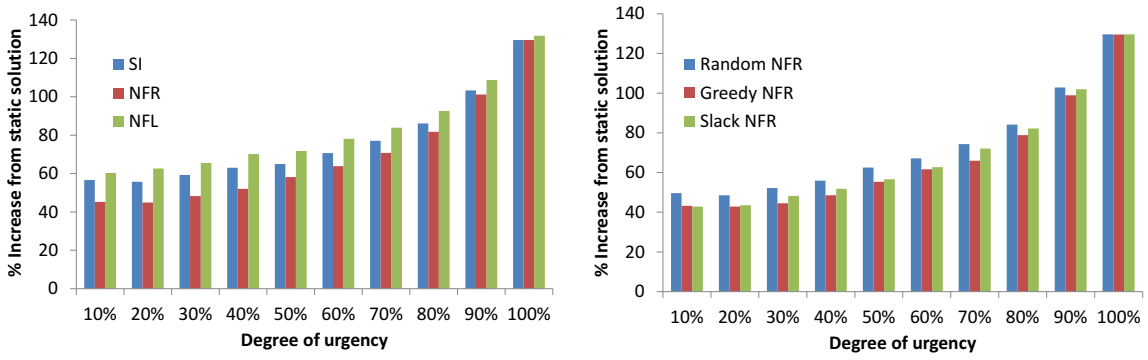
6.5 Results for the Dynamic Insertion Heuristics

This section will compare the results for the 3 methods of dynamic insertion detailed in Section 6.4, using the 3 criterion for insertion, based on the most successful insertion methods for the static variant of the problem (see Section 2.8.1).

Figure 6.9 provides results for the instances from the set P1 described in Section 6.2. Here SI denotes the simple insertion method, NFR, the non-fixed request insertion method and NFL, the non-fixed location insertion method.

Figure 6.9a provides results for each of the 3 dynamic methods of insertion, for each degree of urgency and the result is the average for the 3 insertion criterion. All figures in this section report the average percentage increase in the total distance travelled over the 56 instances, compared to the best result for the static variant of the problem achieved by our algorithm in Section 4.7.

As expected, as the degree of urgency of the dynamic requests increases, so does the additional percentage increase of the solution. It can be seen that it is the method of inserting the non-fixed requests that achieves the most promising results, at each degree of urgency. This agrees with the result of Pankratz [2005b] who found that the H2 insertion method outperformed the H1 insertion method, when investigating the urgency of requests. This could be due to the fact that it allows some alteration to the ordering of the locations in a route, but does not allow all of the non-fixed part of a route to be changed, this results in preserving some of the original ordering of the requests, which was shown by Pankratz [2005b] to improve the results.

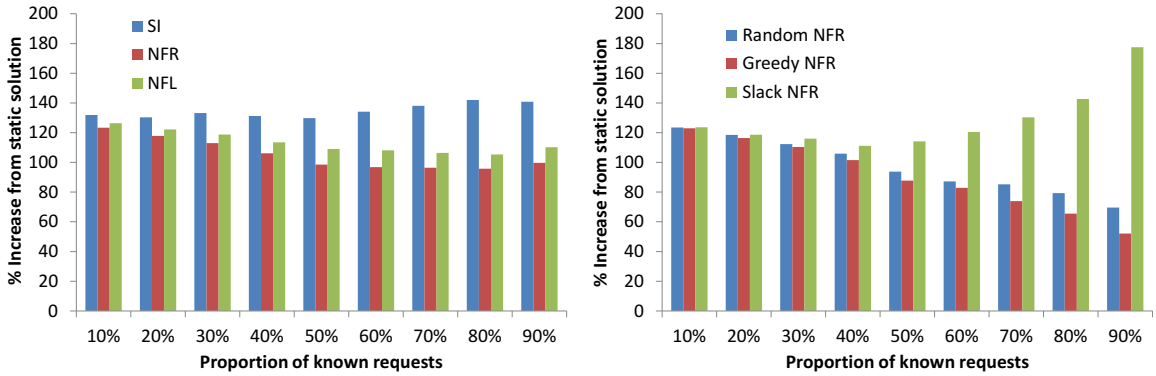


(a) Average results for 3 criterion (b) NFR for each criterion

Figure 6.9: Dynamic Insertion Methods for the P1 set of instances

Figure 6.9b provides the results for the 3 criterion for insertion, whilst applying the non-fixed request dynamic insertion method. It is shown that whilst applying the dynamic insertion method of inserting the non-fixed request, which provides the best results overall, it is the *greedy* insertion criterion that achieves the most promising result. This is to be expected where no improvement heuristic is applied and was seen in Section 3.8 for the static variant of the problem.

Each of these methods is investigated in Figure 6.10 for the instances of the set P2, as described in Section 6.2. Figure 6.10a provides results for each of the 3 dynamic methods of insertion for each increasing proportion of static requests. The result again is the average, over the 3 criterion for insertion.



(a) Average results for 3 criterion (b) NFR for each criterion

Figure 6.10: Dynamic Insertion Methods for the P2 set of instances

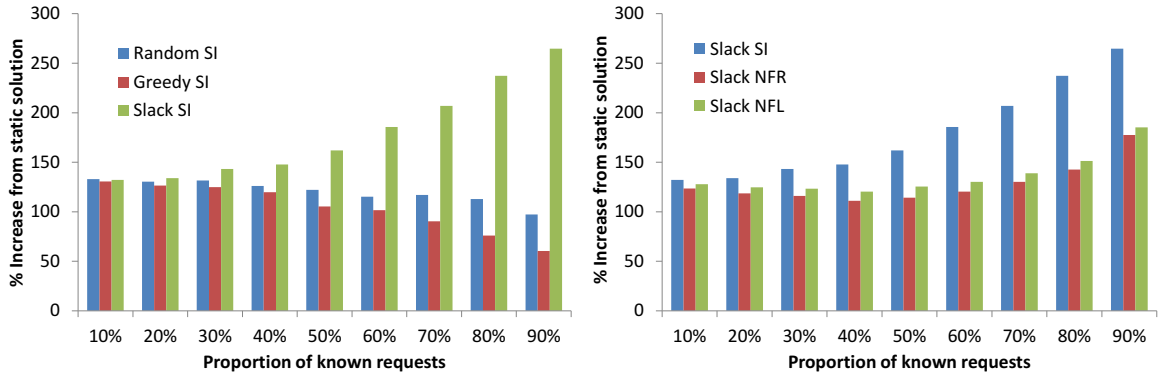
It is the insertion of all non-fixed requests that again provides the best results, as shown in Figure 6.10a. It was found by Pankratz [2005b] that the H2 method outperformed

H1 when also considering the proportion of dynamic requests, hence our results agree once more.

Figure 6.10b provides the results for the 3 criterion for insertion whilst applying the non-fixed request dynamic insertion method. It is shown that the *greedy* insertion criterion again achieves the most promising result.

As expected, as the proportion of static requests increases, so the cost of the solution decreases, for both the case of the non-fixed requests and the non-fixed locations insertion methods. However, this is not the case for the simple insertion method and the *slack* criterion for insertion. We now look to investigate both the simple insertion method and the *slack* criterion for insertion in more detail, to determine the cause for this.

Figure 6.11a shows that when considering each of the 3 criterion for insertion separately, for the cases of the simple insertion method, varying results were achieved. Figure 6.11 provides a summary of how the 3 criterion for the simple insertion method differ.



(a) Simple Insertion

(b) *Slack* criterion of insertion

Figure 6.11: Further breakdown of Methods for the P2 set of instances

It is clear that it is the simple insertion method under the *slack* criterion for insertion that is skewing the overall results. It can be seen that for these two methods combined, there is a significant increase in the results for a high proportion of known requests. The *slack* insertion criterion inserts the most urgent requests first and, for these instances, all dynamic requests are arriving at the highest degree of urgency. The simple insertion method allows no alteration to the current ordering within a route when inserting the dynamic requests. Therefore, for high proportions of known requests, a small proportion of the requests are dynamic and arrive with the latest degree of urgency so have to immediately be serviced by a route. As no change can be made to the current ordering of the route, these are inserted into a new route, ultimately increasing the

cost of the solution.

It has been shown previously in the literature that more flexible revisions consistently yield better results than simple incremental revisions (see Yang et al. [2004]). However, it has also been shown that preserving some information from the original route at the re-insertion phase provides promising results, as shown by the GA in Pankratz [2005a]. Therefore, this could provide the evidence as to why the method of re-insertion based on inserting all non-fixed locations did not perform as well as the non-fixed requests.

Overall it can be seen for both sets of instances and for the varying criteria for insertion that the dynamic insertion method of non-fixed requests provides the lowest percentage increase in total travel distance, for all cases. This will need to be investigated further to determine if this is still the case when an improvement heuristic is added, as previous results show that this was not the case for the static problem.

The next section will look to improve on the results achieved in this section by attempting to adapt both the tabu search heuristic and the branch and bound heuristic described in Chapter 4 to the dynamic problem. The varying criteria for insertion will continue to be analysed to determine how each performs after the improvement phase.

6.6 Improvement Methods in a Dynamic Environment

This section will adapt 2 of the improvement methods studied previously for the static PDPTW to the dynamic variant of the problem introduced in this chapter. This includes the tabu search heuristic introduced in Section 4.2 and the branch and bound heuristic introduced in Section 4.5. They will be analysed under both varying degrees of urgency and varying proportions of dynamic requests.

6.6.1 Tabu Search Heuristic

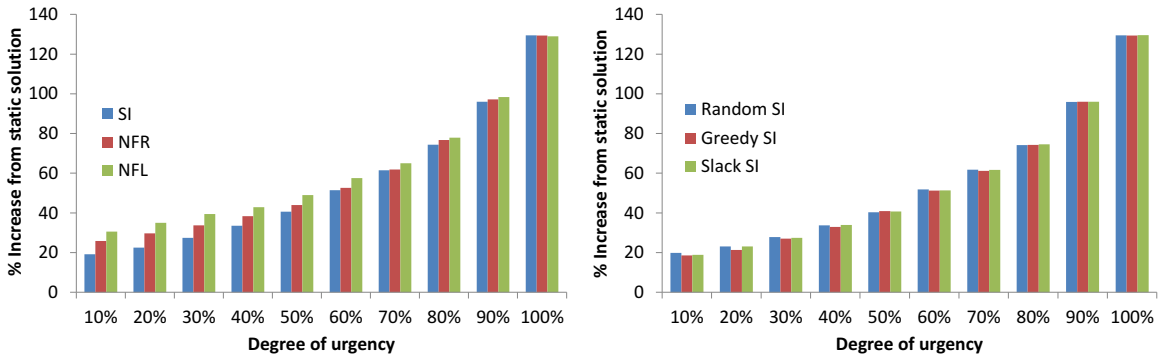
The tabu search heuristic outlined in Section 4.4 will now be adapted to a dynamic environment. There are three main criterion which need to be determined when applying the tabu search heuristic in this research, as outlined in Section 4.3. These are: the tabu attribute (the characteristic to be stored within the tabu list), the tabu tenure (the length of the tabu list, meaning how many iterations an attribute of a move remains in the tabu list), and the stopping criterion (how many iterations need to be

performed, without an improvement to the best found solution, before the search is stopped). The search is also stopped if there remains no feasible move to be made to the existing solution.

The attribute to be stored in the tabu list will again be the direct and indirect delivery edges obtained from the resulting move (see Section 4.6). This attribute achieved the most promising results for the static variant of the problem and preliminary results showed its success in a dynamic environment. This also allows a more meaningful comparison between results achieved by the static algorithm and those achieved in the remainder of this chapter.

The tabu tenure and stopping criterion still need to be determined. For the static variant of the problem, these were both equal to the number of requests present in the instance, but for the dynamic variant of the problem, this is unknown. Tabu tenures and stopping criterion proportional to the number of available requests were investigated. Preliminary results showed that a tabu tenure and stopping criterion equal to the number of available non-fixed locations provided good results. Overall, there was very little difference between the results investigated. This could be due to the reduction in the number of available moves at each iteration due to the decrease in the number of available requests. Therefore these parameters will be adopted in this section.

Figure 6.12 provides average results for each of the 3 dynamic insertion methods, for each degree of urgency. The result is the average over the 3 criterion for insertion. For all figures in this section the result reported is again the average precentage increase in the total distance travelled over the 56 instances, compared to the best result for the static variant of the problem, achieved by our algorithm in Section 4.7.



(a) Dynamic Insertion Methods

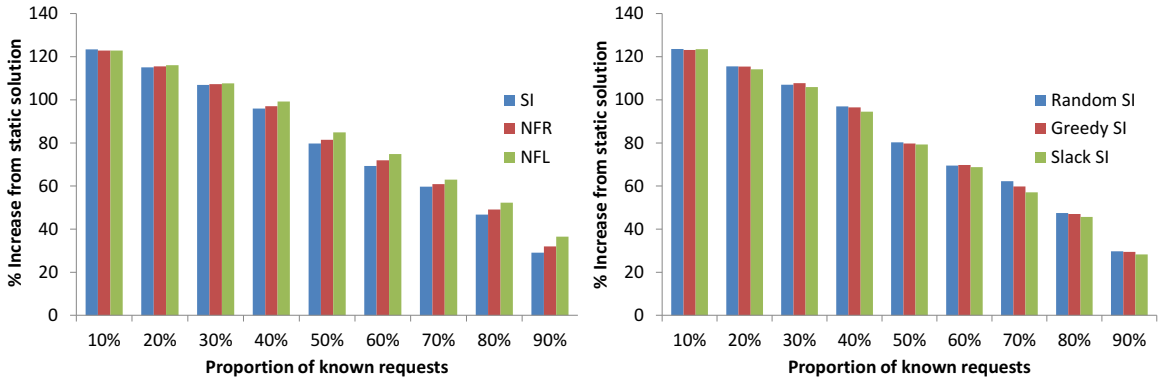
(b) Simple Insertion

Figure 6.12: Tabu Search Heuristic for the P1 set of instances

From Figure 6.12a it can be seen that the simple insertion method provides the overall lowest total travel distance at each increasing degree of urgency. It is for the lowest degrees of urgency that the simple insertion method provides the biggest gain compared to the other two methods.

Investigating the simple insertion method further, Figure 6.12b provides the results for each of the criterion for insertion. It can be seen that the overall results are very similar; however it appears that it is the *greedy* method that produces the overall lowest percentage increase from the static solution.

We will now investigate each of these methods for the set P2, again described in Section 6.2. Figure 6.13a shows results for each of the 3 dynamic methods of insertion for each increasing proportion of known requests, the result is again the average over the 3 insertion criterion.



(a) Dynamic Insertion Methods

(b) Simple Insertion

Figure 6.13: Tabu Search Heuristic for the P2 set of instances

From Figure 6.13a it can be seen that it is again the simple insertion method that provides the overall lowest total travel distances. However, for the lowest proportion of known requests, both the insertion method of re-inserting the non-fixed requests and the non-fixed locations achieve better solutions. Investigating the simple insertion method further, in Figure 6.13b, the *slack* method provides the most promising results.

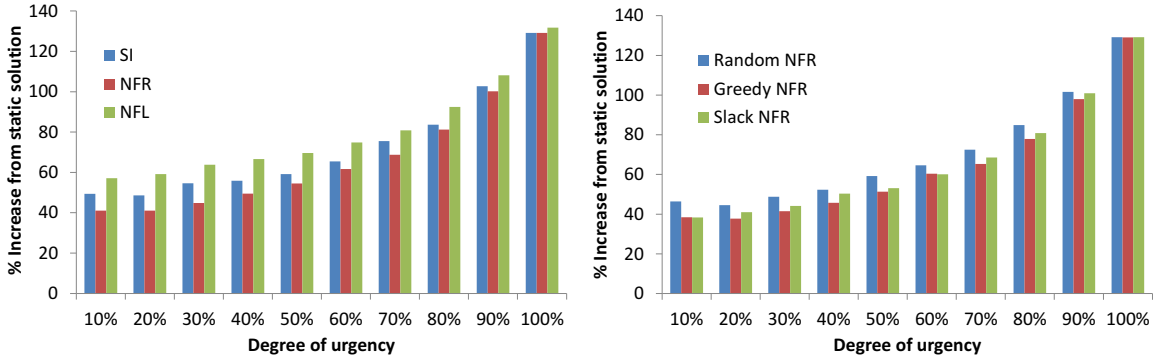
The next section will look to investigate the results of applying the branch and bound heuristic outlined in Section 4.5 for the static variant of the problem to the dynamic problem.

6.6.2 Branch and Bound Heuristic

The branch and bound heuristic introduced in Section 4.5 attempts to improve the final solutions achieved by our algorithm. It takes a single route or subset of a route, depending on the number of services, and attempts to improve the ordering of the locations using a branch and bound technique.

This method will now be adapted to the dynamic problem, where it will look to improve the ordering of all non-fixed locations in each route or subset of a route. It will be applied at each iteration, after the arrival of the new requests and their insertion in the existing solution via one of the dynamic insertion methods. As in Section 4.5, the maximum number of locations in each sub-section is 14.

It will now be investigated as to what improvement can be made to the solutions under the varying scenarios by applying the branch and bound heuristic along with each of the dynamic insertion methods and varying criteria for insertion. Figure 6.14a provides results for each of the 3 dynamic methods of insertion, for each degree of urgency. The result is the average for the 3 insertion criterion.



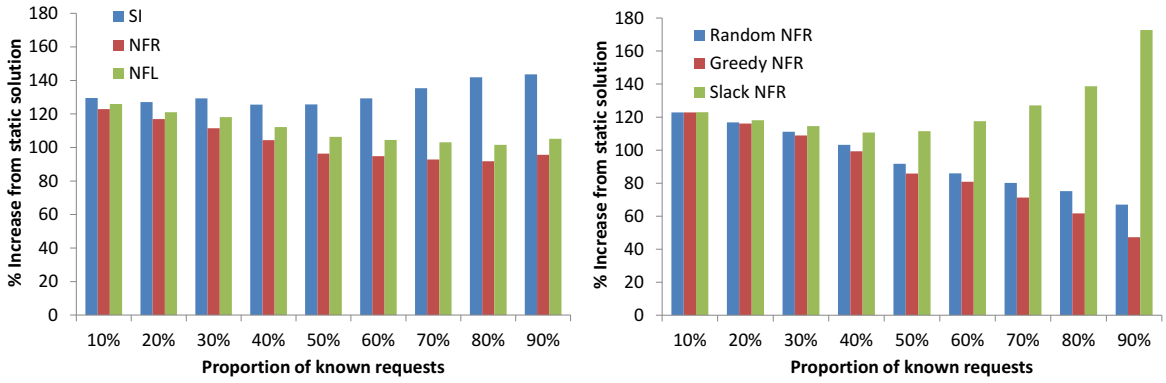
(a) Dynamic Insertion Methods

(b) NFR Insertion

Figure 6.14: Branch and Bound Heuristic for the P1 set of instances

It can be seen that the method of insertion of the non-fixed requests provides the most promising results overall, however these are not as good as those provided by the tabu search heuristic. Figure 6.14b shows the 3 methods of insertion for each degree of urgency under the non-fixed request criterion for insertion and it can be seen that it is the *greedy* insertion method that provides the overall best results.

We will now investigate each of these methods for the set P2, Figure 6.15a shows results for each of the 3 dynamic methods of insertion for each increasing proportion of static requests. The result again is the average, over the 3 criterion for insertion.



(a) Dynamic Insertion Methods

(b) Simple Insertion

Figure 6.15: Branch and Bound Heuristic for the P2 set of instances

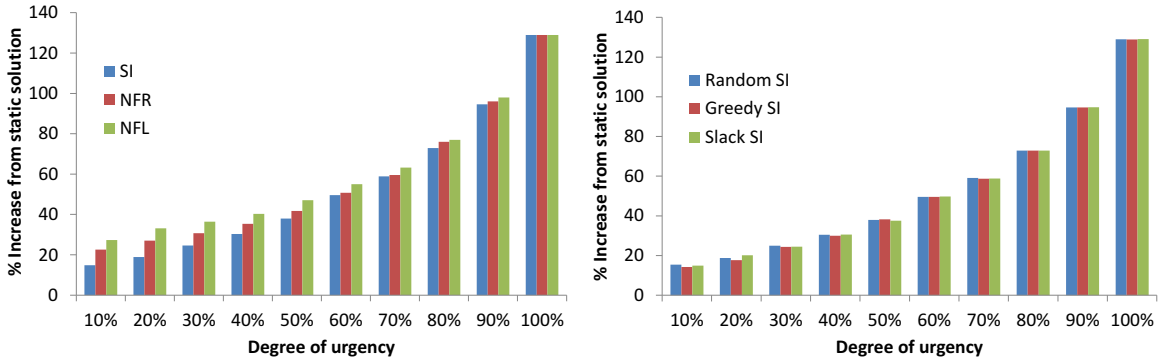
After examining the results in Figure 6.15a, it can be seen that, once again, the dynamic insertion method of re-inserting all non-fixed requests provides the overall lowest total distances, when paired with the branch and bound heuristic. Again these results are not as promising as those achieved by the tabu search heuristic.

To investigate this further Figure 6.15b provides the results for the non-fixed requests method of insertion and it can be seen that is it the *greedy* criterion that provides the best results once again. A similar trend can be seen again with the *slack* criterion as was the case prior to the improvement phase. This trend was similar for both the simple insertion method and the method of inserting all non-fixed locations.

It is clear that the results for applying the branch and bound heuristic are not as good as those of the tabu search heuristic, this could be because it does not allow any change to the grouping of requests, it only looks to improve the ordering of requests within a route, or sub-section of that route. However, we will now investigate the results of combining these two methods.

6.6.3 Combining Improvement Heuristics

Figure 6.16a provides results for each of the 3 dynamic methods of insertion, for each degree of urgency and is the average for the 3 insertion criterion. It can be seen that the simple method of insertion provides the overall most promising results and these slightly improve on those for the tabu search heuristic alone.

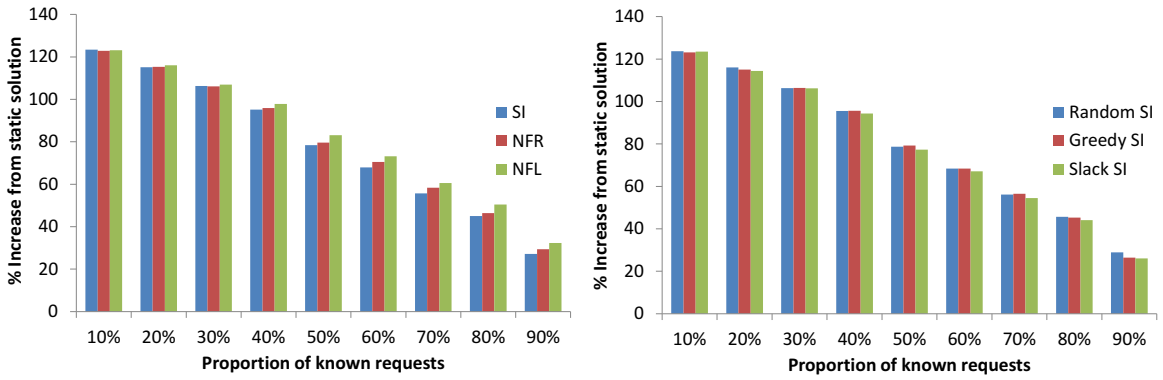


(a) Dynamic Insertion Methods (b) Simple Insertion

Figure 6.16: Combining Improvement Heuristics for the P1 set of instances

Investigating this further, Figure 6.16b shows the 3 criterion for insertion for each degree of urgency, under the simple insertion method. It can be seen that the *greedy* insertion criterion provides the overall best results. As requests become fixed over time, there are fewer opportunities to improve the solution at each iteration. It could therefore be the case that the solution achieved after the initial construction phase, now has more of an effect on the final solution achieved. This could explain why the *greedy* heuristic achieves the most promising results for both sets of instances as it was shown this provided the best initial results.

We will now investigate each of these methods for the set P2, Figure 6.17a shows results for each of the 3 dynamic methods of insertion for each increasing proportion of known requests. The result again is the average, over the 3 criterion for insertion and it can be seen once again that it is the simple insertion method that provides the most promising results compared to those of the original static algorithm.



(a) Dynamic Insertion Methods (b) Simple Insertion

Figure 6.17: Combining Improvement Heuristics for the P2 set of instances

Investigating this further, Figure 6.17b shows the 3 criterion for insertion for each proportion of known requests under the simple insertion method. It can be seen that the *slack* simple insertion methods provides the overall best results. The results again improve on those of applying the tabu search heuristic without the branch and bound heuristic.

The next section will summarise the results achieved in this section and will investigate the computational times of our algorithm. Comparisons will then be made to the results obtained in Pankratz [2005b].

6.7 Summary of Results

The best results achieved when applying the improvement heuristics outlined in Section 6.6 will now be summarised for the 2 sets of instances. It was found that using the tabu search heuristic followed by the branch and bound heuristic provided the best overall results for both sets of instances. The basic procedure for our algorithm in a dynamic environment is outlined in Algorithm 16.

Algorithm 16 DynamicAlgorithm

```
1: Run RandomInsertion on static requests
2: while (No new requests arrive) do
3:     Wait
4: Run SimpleInsert
5: Run TabuMove
6: Run BranchBound
```

For both set P1 and P2, the best 3 methods used the simple insertion method, which inserts the dynamic requests into the solution without making any alteration to the current ordering of the locations in the route. Therefore each case only differs by the chosen criteria for insertion being *random*, *greedy* or *slack*.

A summary of results are provided in Table 6.2 and 6.3, where the result is the average percentage increase in the total distance travelled over the 56 instances, compared to the best result for the static variant of the problem, achieved by our algorithm in Section 4.7. For set P1 results are provided for each degree of urgency and for set P2 results are provided for each proportion of known requests.

Degree of urgency	Rand SI Tabu + B&B	Greedy SI Tabu + B&B	Slack SI Tabu + B&B
10%	15.45	14.24	14.93
20%	18.80	17.67	20.11
30%	24.93	24.38	24.47
40%	30.47	29.97	30.57
50%	37.94	38.28	37.56
60%	49.55	49.55	49.79
70%	59.12	58.73	58.85
80%	72.84	72.89	72.90
90%	94.59	94.59	94.71
100%	129.00	128.89	129.04

Table 6.2: Average % increase for the Best 3 Methods for the P1 set of instances by each degree of urgency

It can be seen in Table 6.2 that the results for the *greedy* insertion criterion are slightly lower than the other two methods; however there is little difference between them. For at least one degree of urgency, each of the *random* and *slack* criterion have achieved the overall lowest percentage increase in the total travel distance.

Performing a one-way analysis of variance for repeated measures between each of the 3 sets of results above, a p-value of 0.142 is obtained. Therefore there is no significant difference between the results achieved by any of the insertion criteria at 5% significance (or even at 10%).

From Table 6.3 it can be seen that the results of set P2, for the best 3 methods are again similar, with the most promising results coming from the *slack* insertion criterion. Performing a one-way analysis of variance for repeated measures between each of the 3 sets of results above, a p-value of 0.02 is obtained. Therefore there is a significant difference between the results. Pairwise comparisons reveal that the *slack* insertion criterion is significantly different from the other two methods, at 5% significance, and it is the *slack* insertion criterion which achieved significantly lower results.

From the results achieved in Tables 6.2 and 6.3 it can be seen that there was no significant difference between the results achieved for the *random* criterion and the *greedy* criterion for insertion. This supports preliminary investigations that showed there were limited feasible insertion positions for the dynamic requests at each interval, therefore little variation achieved in solutions obtained after 100 runs of the algorithm

when applying the *random* criterion, compared to that of a single run, which is what is reported in this chapter.

Proportion of dynamic requests	Rand SI Tabu + B&B	Greedy SI Tabu + B&B	Slack SI Tabu + B&B
10%	123.74	123.06	123.53
20%	116.06	114.72	114.45
30%	106.33	106.61	106.21
40%	95.57	96.26	94.38
50%	78.75	78.91	77.31
60%	68.41	70.35	67.10
70%	56.12	58.62	54.51
80%	45.69	45.10	44.10
90%	28.94	25.64	26.02

Table 6.3: Average % increase in TD for the Best 3 Methods for the P2 set of instances by varying proportions of known requests

Comparing the extremities of both cases of urgency and dynamics in Tables 6.2 and 6.3, it can be seen that the results for the lowest degree of urgency (10%) improve on those with the highest proportion of known request (90%). For the results of the lowest proportion of known requests it can be seen that it is the results from the set P1 that improve on those from P2. Therefore it would seem reasonable to conclude that the instances with a high proportion of dynamic requests seem to be the most difficult for our algorithms.

Table 6.4 reports the average time taken to update the solution after the arrival of a dynamic request, by each increasing degree of urgency for the instances in the set P1.

Degree of urgency	Rand SI Tabu + B&B	Greedy SI Tabu + B&B	Slack SI Tabu + B&B
10%	0.2143	0.2404	0.2344
20%	0.1305	0.1292	0.1322
30%	0.0889	0.0915	0.0911
40%	0.0429	0.0407	0.0413
50%	0.0262	0.0274	0.0255
60%	0.0131	0.0136	0.0128
70%	0.0061	0.0062	0.0061
80%	0.0024	0.0024	0.0023
90%	0.0011	0.0010	0.0010
100%	0.0006	0.0006	0.0006

Table 6.4: CT required by the Best 3 Methods for the P1 set of instances by each degree of urgency (seconds)

Table 6.5 reports the average computational time required to update the solution after the arrival of a new request, for each proportion of known requests in the set P2.

Proportion of dynamic requests	Rand SI Tabu + B&B	Greedy SI Tabu + B&B	Slack SI Tabu + B&B
10%	0.0008	0.0008	0.0008
20%	0.0018	0.0015	0.0014
30%	0.0030	0.0035	0.0029
40%	0.0069	0.0078	0.0080
50%	0.0155	0.0170	0.0134
60%	0.0403	0.0556	0.0269
70%	0.0800	0.0853	0.0506
80%	0.1454	0.1687	0.0913
90%	0.3714	0.4770	0.2144

Table 6.5: CT required by the Best 3 Methods for the P2 set of instances by varying proportions of known requests (seconds)

Looking at these tables, it is clear for both sets of instances that the computational times are very low and therefore this method is practical for use in a real-time environment for both varying degrees of urgency and varying proportions of dynamic requests.

It is the instances with a low degree of urgency or a high proportion of known requests which have the highest computational times. This is due to the larger number of requests arriving at a particular point in time in these instances, which then need to be incorporated into the solution leading to greater opportunities for improvement. For instances with a high proportion of known requests, the largest number of requests is at the beginning of the scheduling horizon and for the instances with a low degree of urgency, this is early on in the scheduling horizon.

Pankratz [2005a] recommends that a planned revision should take less than 60 seconds. Clearly, our results more than meet this criterion.

The best method overall from the results in Tables 6.2 and 6.3 will now be examined in more detail for each of the 6 characteristics of instances. For comparison reasons, and due to the fact that there is no significant difference between the results for the case of set P1, we choose the *slack* insertion method combined with the tabu search heuristic and the branch and bound method to be analysed further.

Figure 6.18 provides the results for the set P1 and Figure 6.19 for the set P2. The result again is the average percentage increase in the total distance travelled over the instances in each set, compared to the best result for the static variant of the problem, achieved by our algorithm in Section 4.7.

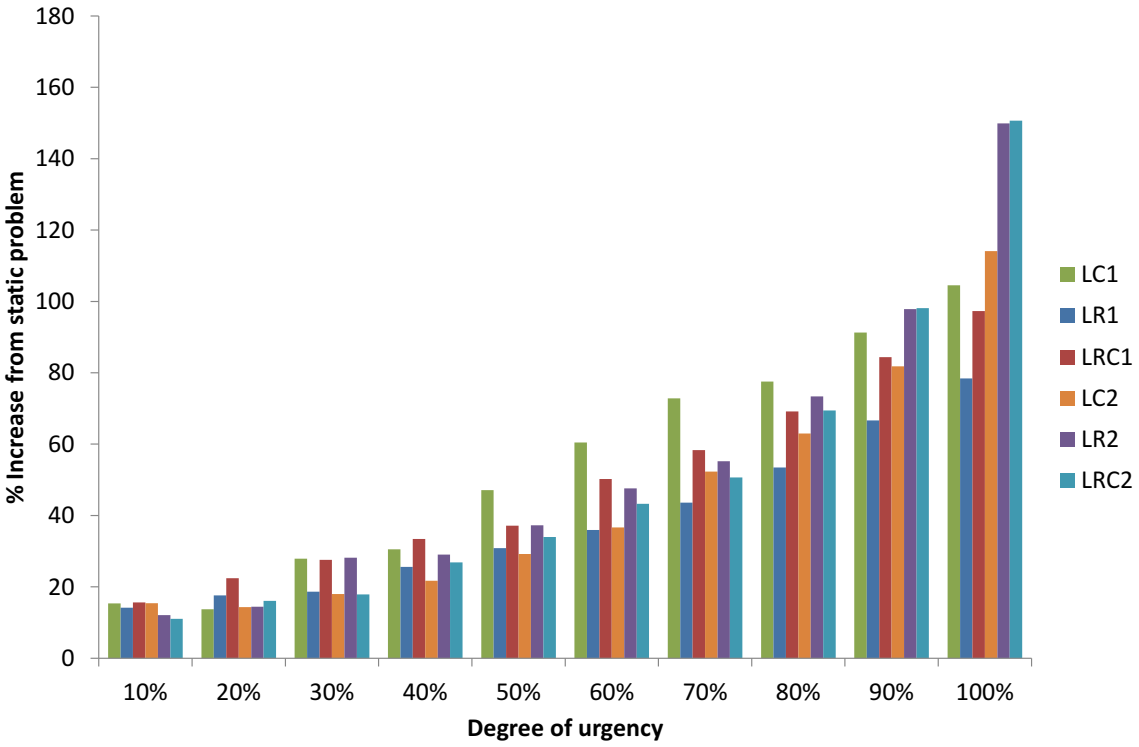


Figure 6.18: Breakdown by instance for the P1 set of instances

Comparing Figures 6.18 and 6.19 it is seen that instances with a longer scheduling horizon perform worse under varying degrees of dynamics than an increasing degree of urgency. This is due to the fact that requests are spread over a longer scheduling horizon. Therefore, at each insertion, there are a higher number of fixed requests which limits opportunity for improvement. This agrees with the result found in Pankratz [2005b].

It is also seen that the instances with clustered requests, in particular those with a short scheduling horizon are harder to solve at each degree of urgency. For both cases it appears it is the random instances with a short scheduling horizon which provide the minimum increase from the static solution.

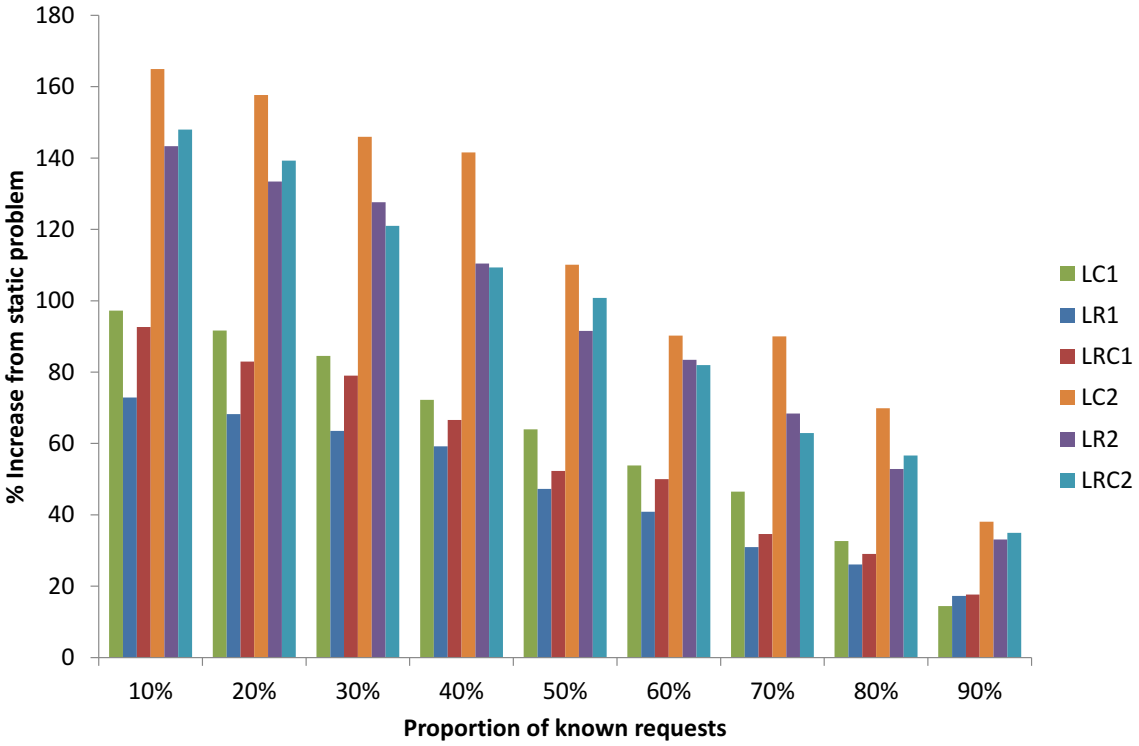


Figure 6.19: Breakdown by instance for the P2 set of instances

A comparison will now be made with the results achieved by Pankratz [2005b] for both their H1 and H2 insertion heuristics and their GGA. The best overall method chosen from the results above is the *slack* simple insertion paired with both the tabu search heuristic and the branch and bound heuristic. Figure 6.20 provides the comparison for the set P1, whilst Figure 6.21 provides the comparison for the set P2.

To ensure a direct comparison can be made, the graphs are formatted in the same manner as those presented in Pankratz [2005a]. The relative solution quality for our algorithm, in this case, is with regards to the static results achieved in Pankratz [2005a].

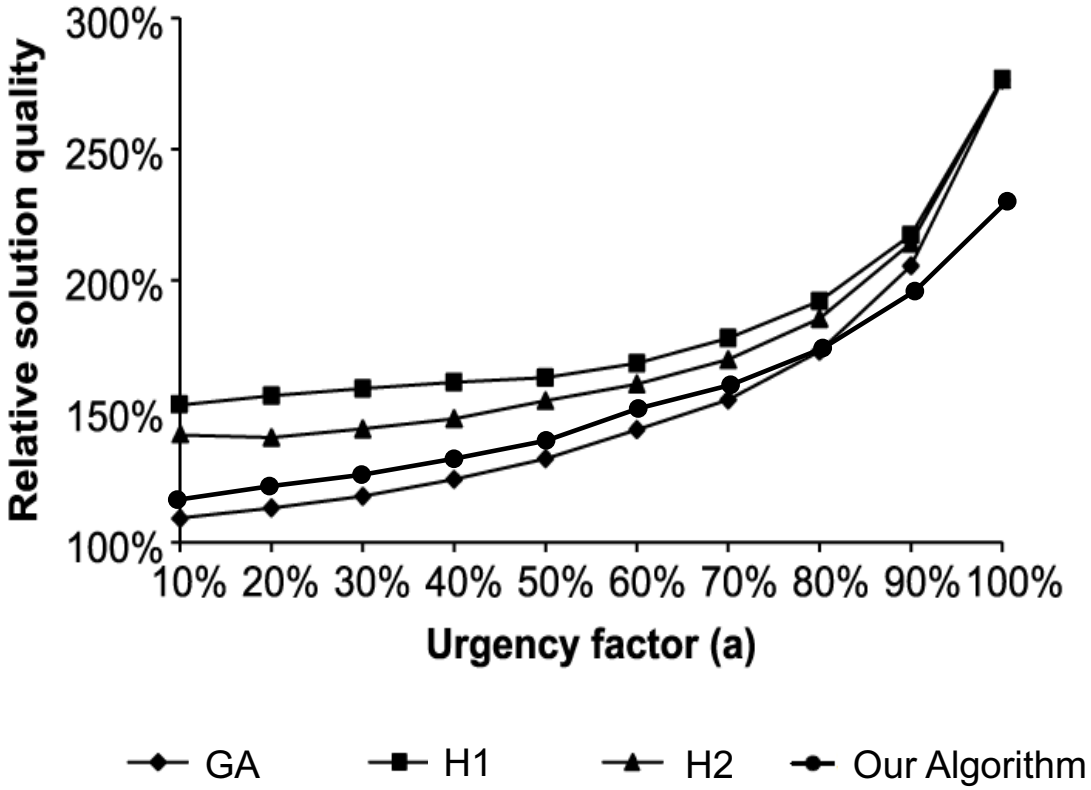


Figure 6.20: Comparison to Pankratz [2005b] for the P1 set of instances

From the results in Figure 6.20, for the P1 set of instances, it can be seen that the GGA of Pankratz [2005a] achieved the best results for up to an urgency of 80%, then our algorithm provides the lowest overall relative solution quality, improving on the results for both 90% and 100% urgency.

The improvement of the GA over our algorithm could be accounted for by the fact that it concentrates on the grouping aspect of the requests, rather than the ordering of the requests within routes. For the lowest degrees of urgency, a larger number of requests arrive early on in the scheduling horizon and need to be assigned to routes. Therefore, correctly grouping these requests is important at this stage. Later on in the scheduling horizon, when fewer requests arrive and more requests are fixed to a route, our algorithm which looks to improve the ordering of locations *within* routes appears to provide better solutions.

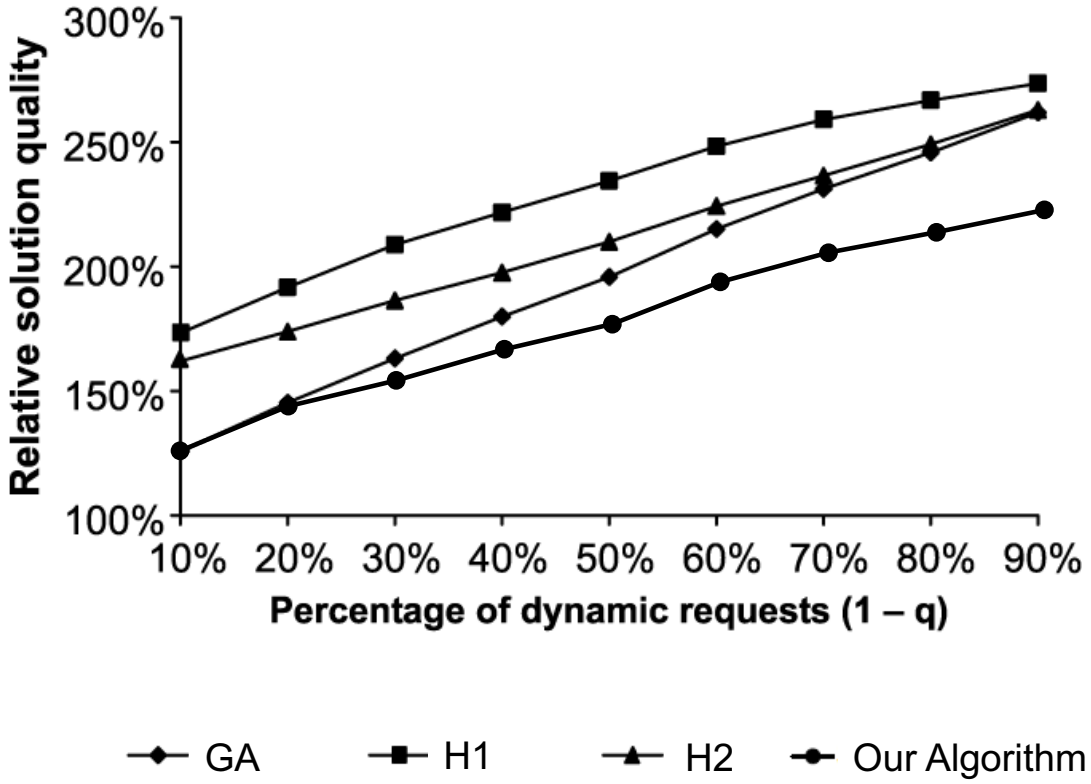


Figure 6.21: Comparison to Pankratz [2005b] for the P2 set of instances

From the results presented in Figure 6.21 it can be seen that our algorithm outperforms the results for all methods of Pankratz [2005b], especially for the instances where there are over 50% dynamic requests. This shows our algorithm works well for instances with requests arriving at a high degree of urgency, both for the instances of set P1 and the instances of set P2, as all instances in set P2 had the highest degree of urgency. However, the results could still be improved for requests arriving with a low degree of urgency.

It should be noted that the results for the methods of Pankratz [2005b] are averaged over 10 instances for each of the 56 static instances, where for our results, we use the average over one set of the 56 instances. Figure 6.22 shows a box plot for the average results obtained for each of the 10 randomly generated set of instances for the proportion of known requests equal to 10%. It can be seen that the results obtained by each set are similar.

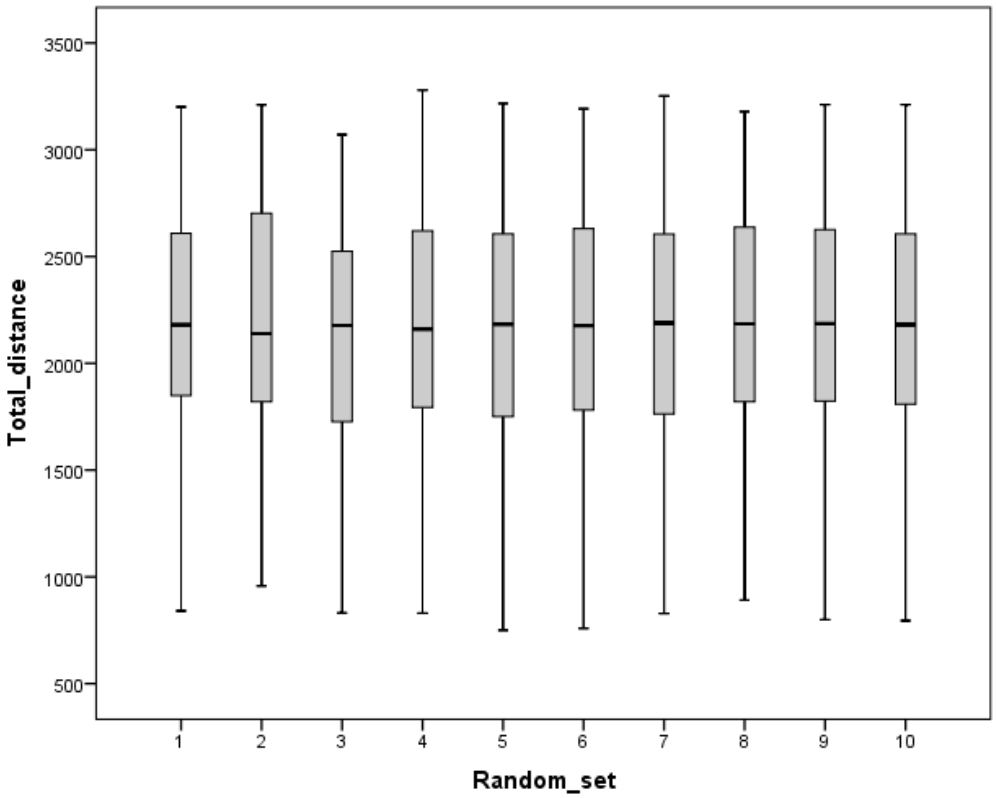


Figure 6.22: Results each Random Set with $q = 10\%$ for the P2 set of instances

A one-way analysis of variance (ANOVA) is performed on the 10 independent sets of instances giving a p-value of 1.000 ($F = 0.09$), showing no significant difference between the average result obtained in each case. The average total travel distance obtained from the instances in the first set, which has been used in our analysis so far, is 2187.99. The average results obtained over all 10 sets of instance, as used by Pankratz [2005b], is 2185.74, a difference of 0.10%. We therefore believe our results to be comparable to those of Pankratz [2005b] and a similar result is expected for all proportions of known requests.

The next section will conclude the chapter and provide suggestions for further research into the DPDPTW for investigation in the following chapter.

6.8 Chapter Summary

This chapter has looked to investigate ways in which to adapt the algorithm introduced in Section 4.7 to the DPDPTW. This was investigated using the instances of Pankratz

[2005a] which were originally generated from the 56 static PDPTW instances of Li and Lim [2001] (see Section 3.3).

Section 6.4 introduced three insertion heuristics both for constructing an initial solution at the beginning of the scheduling horizon for the requests known in advance and to update the solution during the scheduling horizon, for the requests that arrive dynamically. These are developed from those found in the literature and from those developed in Section 3.4.

Section 6.6 investigated methods of improving the solutions based on those studied for the static variant of the problem in Sections 4.2 and 4.5. It was shown that a simple *slack* insertion method when paired with the tabu search heuristic and the branch and bound heuristic provided the most promising results.

The initial investigations performed for the varying insertion methods and improvement heuristics provide insights into the characteristics of the problem which have not before been explored, these include which is the best method of insertion and what should be inserted at the varying degrees of urgency and proportions of dynamic requests.

It was found that a simple insertion of all new requests at each interval improved on a more destructive method of removing non-fixed locations from routes when an improvement phase is to be applied. The main reason for this is that it allowed for a greater improvement to be made by the tabu search heuristic and the branch and bound method resulting in a better final solution. It was found that a *greedy* insertion method provided the most promising initial results, but this allowed for little changes to be made during the improvement phase, hence the final solution was worse than for the *slack* insertion method. The improvement of the simple *slack* insertion method is greatest at the lowest degree of urgency and the highest proportion of known requests where there is the greatest opportunity for the heuristics to improve the solution.

The final results provided in Section 6.7 showed that our algorithm produced reasonable results for the instances of set P1, but produced improved results under a high degree of urgency. However, it is for the set of instances of P2 that there is greatest improvement from the results of Pankratz [2005b]. This shows that our algorithm performs well when the proportion of dynamic requests is high and when requests are arriving with the highest degree of urgency.

There are limitations to the instances provided in this chapter with regards to the opportunity for improvement during the scheduling horizon. The next chapter will therefore apply the methods developed to instances more comparable to those found in a real-life courier service.

Chapter 7

Investigating the DPDPTW

7.1 Introduction

This chapter will continue to investigate the DPDPTW using the instances of Mitrovic-Minic et al. [2004], which were generated based on real-life data for a courier service. The aim of the chapter is to firstly explore the behaviour of our algorithm during the scheduling horizon, by observing the changes made to the solution after incorporating the dynamic requests. Secondly, it is hoped furthering our research to more realistic problems will aid in validating the results achieved in Chapter 6.

The chapter is structured as follows. The instances of Mitrovic-Minic et al. [2004] are introduced in Section 7.2 with an overview of how they were created. Section 7.3 investigates the varying methods of inserting the dynamic requests that were introduced in Section 6.4. Section 7.4 then looks at applying the improvement heuristics from Section 6.6 to validate the results achieved in Section 6.7.

Section 7.5 studies the behaviour of the improvement heuristics during the scheduling horizon. As an attempt to further investigate our algorithm over time, varying intervals at which to re-start our algorithm are investigated in Section 7.6.

The final results achieved by our algorithm are compared both to the results achieved in the literature of Mitrovic-Minic et al. [2004] (see Section 7.7) and to those obtained for the static variant (see Section 7.8). The chapter is concluded in Section 7.9.

7.2 DPDPTW Instances of Mitrovic-Minic et al. [2004]

This section introduces the instances to be examined in this chapter. We chose these instances specifically because the requests were generated based on real-life data (collected in two medium-to-large courier services in Vancouver, Canada). The 'Rnd8' instances are the first set of instances used in the study by Mitrovic-Minic et al. [2004]. Here, instances have 100, 500 and 1000 requests with 30 instances for each problem size, giving a total of 90 instances for this set. For the final results provided at the end of this chapter in Sections 7.7 and 7.8 a further set of instances of Mitrovic-Minic et al. [2004], the 'Rnd9' instances, are introduced to further support our findings.

The area covered by the courier service is 60 km x 60 km, with a few delivery locations (around 6%) out of the service area (these have negative coordinates). It is assumed by Mitrovic-Minic et al. [2004] that vehicle speed is constant at 60km/h, hence time is equal to distance with 1 km = 1 minute. The depot is located at (20km, 30km). For this case service time at each location is considered zero and the load of each request is also assumed to be zero, i.e. there are no capacity constraints. The requests made to the courier service consist of letters and small parcels and therefore a restriction on the vehicle capacity is not a concern for the service, they also consider service time to be negligible. The total scheduling horizon is 10 hours.

The number of vehicles is assumed unlimited in the problem instances; however the methods of Mitrovic-Minic et al. [2004] initialise the solution with 20, 60 and 80 vehicles for the instances with 100, 500 and 1000 requests respectively. An unbounded number of vehicles were chosen in this case as it was found to be consistent with practice, since a large pool of private drivers could be used by the service. Therefore Mitrovic-Minic et al. [2004] do not attempt to minimise the number of vehicles in their algorithm and our approach is consistent with this. The objective considered by Mitrovic-Minic et al. [2004] is to minimise the total distance travelled, therefore a direct comparison can be made to the results achieved by our algorithm.

In each problem instance, time windows are generated such that their distribution emulates real-world requests. The opening of each pickup time window is equal to the time at which the request becomes known to the system and the end of the delivery time window is determined by the request 'type'. A '1 hour' request means that the entire request has to be served within 1 hour from the time at which it becomes known to the system. For a '2 hour' request this has to be serviced within 2 hours and for a '4 hour' request, within 4 hours. A '1 hour' request therefore needs to be 'generated' within the

first 9 hours of the scheduling horizon for it to feasibly be serviced. The positions of the pickup and delivery locations of a 1 hour request are randomly generated such that the 'direct travel time' between the two locations are at most 30 minutes. Also the 'total travel time' from the depot to the pickup location and to the delivery location is at most 45 minutes. A summary of this information for each 'type' of request is provided in Table 7.1.

Type of request	Proportion	Generated	Direct travel time	Total travel time
1 hour	28%	9 hours	30 minutes	45 minutes
2 hour	30%	8 hours	90 minutes	105 minutes
4 hour	42%	6 hours	180 minutes	210 minutes

Table 7.1: Distribution of Requests in Rnd8 Instances of Mitrovic-Minic et al. [2004]

For these instances, requests appear uniformly during the whole service period and no requests are known in advance. Mitrovic-Minic et al. [2004] consider re-starting their algorithm for 40 intervals during the scheduling horizon. This results in a 15 minute interval between each re-start in which new requests arriving to the service are accumulated; this differs from the approach of Pankratz [2005b] who re-start their algorithm each time a new request arrived. The process adopted by Mitrovic-Minic et al. [2004] may be more practical in real-life as it may be unrealistic to re-start the algorithm every time a new request arrives during periods of high demand. The number of requests arriving within each time interval, for an instance with 100 requests, is shown in Figure 7.1.

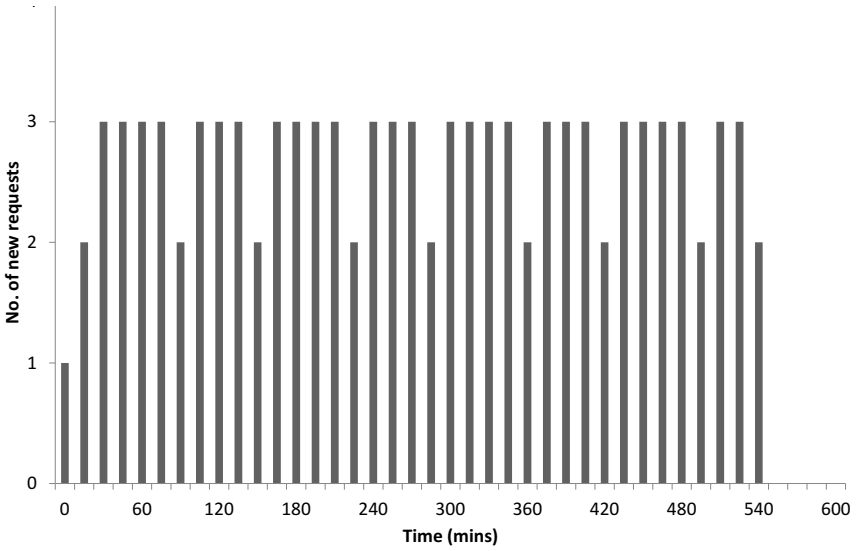


Figure 7.1: Arrival of dynamic requests

Figure 7.1 shows that a maximum of 3 requests are received per interval, with a minimum of 1 request in the first interval. It is clear that no requests arrive during the last hour of the scheduling horizon, as stated above.

It is stated by Mitrovic-Minic et al. [2004] that the solution constructed is one of open routes; therefore the final cost of returning to the depot for each route is not included. This could have been incorporated to stop bias in selecting requests close to the depot at the start of the scheduling horizon. For comparison this approach will also be adopted here.

The next section looks to produce initial solutions for the instances by applying the dynamic insertion methods introduced in Section 6.4.

7.3 Insertion Heuristics

This section will investigate the dynamic insertion heuristics introduced in Section 6.4 for the 'Rnd8' instances of Mitrovic-Minic et al. [2004]. For this case, no requests are known in advance; therefore the initial construction phase is no longer needed.

For information, a location in a route is considered to be fixed at time t, if the vehicle servicing that location has already done so, or has already left the proceeding location at time t (i.e. is currently en-route to the location). In our case, a pickup location is also considered fixed if it can no longer be serviced by the introduction of a new vehicle - it can only be feasibly serviced by its current vehicle or another in close proximity.

Recall that the dynamic insertion methods introduced in Section 6.4 include a simple insertion (SI) method (where each new request is inserted into the feasible position, with the minimal increase in total distance travelled, in the already constructed solution), the re-insertion of all non-fixed requests (NFR) (where each new request, and all requests which are not fixed within a route of the current solution, are inserted as in simple insertion), and the re-insertion of all non-fixed locations (NFL) (where each new request and all non-fixed locations are inserted as in simple insertion). The difference for the case of re-inserting non-fixed locations is that the delivery location of a request may not be fixed to a route, but the pickup location of that request may have already been serviced or be fixed. For this case, the delivery location is removed from the route but it must be returned to the same route, not necessarily in the same position.

Here, the three criterion for insertion introduced in Section 6.3 will be investigated once more; these are the *random* insertion, *greedy* insertion and *slack* insertion criterion. To better understand the difference between each of the insertion methods, a summary

of the number of services available at each interval will be provided. The number of locations to be inserted at each interval for an example instance, for each of the dynamic insertion methods is shown in Figure 7.2.

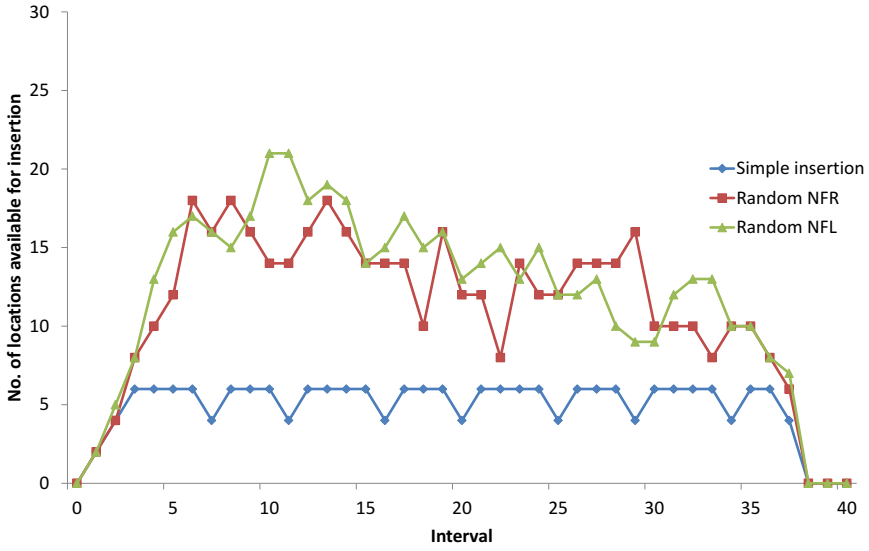


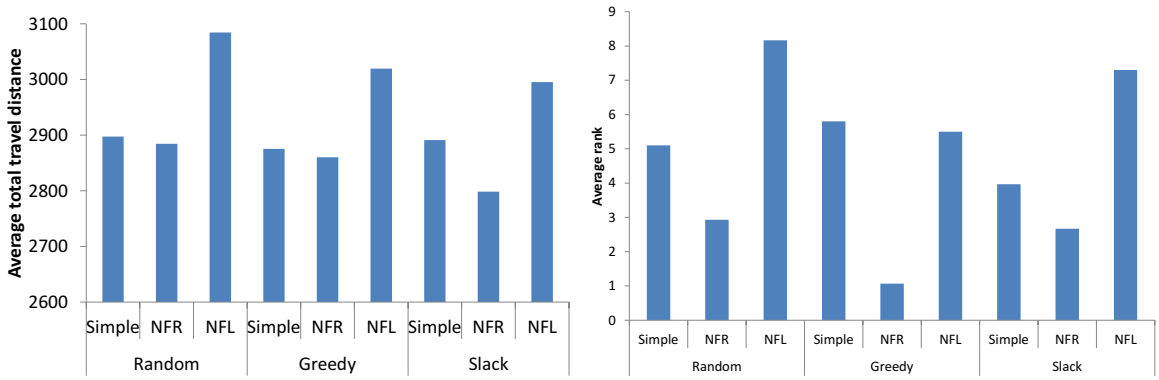
Figure 7.2: Available locations at each interval - $Rnd8\_10h\_100\_000$

We can see that for the case of the simple insertion, this is simply double the number of requests (the pickup and delivery location) that have arrived since the last re-start of the algorithm at each interval. For the case of the non-fixed requests, on average this is more than double the simple insertion and for the case of the non-fixed locations this is again increased.

Early on in the scheduling horizon, the number of non-fixed locations is higher than the non-fixed requests, which is what would be expected as, for some requests, their pickup location will be fixed but their delivery location may still require servicing. However, towards the end of the scheduling horizon, the number of non-fixed requests is higher than the number of non-fixed locations at some intervals. This indicates that a higher proportion of requests are now fixed in the solution created by this method than inserting the non-fixed locations methods. This could be because the solutions created early on in the scheduling horizon by the non-fixed location insertion may provide an improved arrangement in the ordering of locations, due to a higher number of locations to be inserted. Therefore a solution could be created which is able to service a higher number of requests early on in the scheduling horizon, hence, a higher proportion of requests become fixed. This could then result in a lower number of insertion opportunities later on in the scheduling horizon. This result is similar for

each of the insertion criteria.

The results of applying each of these dynamic insertion methods, by each criterion for insertion, is investigated for the 30 instances with 100 requests. Figure 7.3 provides the average results over each of the instances.



(a) Average total distance travelled

(b) Average rank

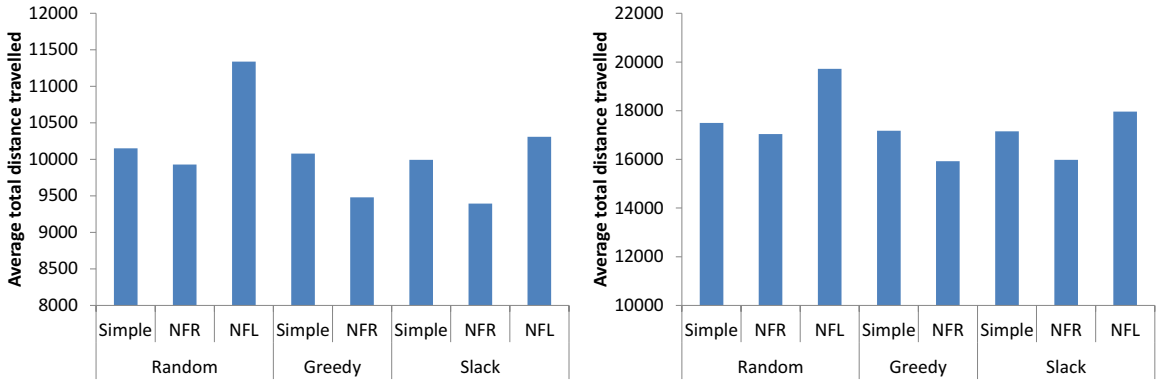
Figure 7.3: Dynamic insertion methods for each criterion for insertion

From Figure 7.3a it can be seen that the non-fixed requests insertion method provides the best results on average, in particular, for the *slack* criterion. Investigating how each method performs for each of the individual instances, the 9 methods are ranked in order of the total travel distance achieved, 1 being the minimum. Figure 7.3b shows the average of the ranks over the 30 instances for each of the insertion methods for each criterion.

It can be seen that the results in Figure 7.3b are reasonably comparable to those in Figure 7.3a. However, for the case of the *greedy* non-fixed requests, this achieves a lower average rank than the *slack* simple insertion. This shows that although the *slack* simple insertion achieved an overall lower total travel distance, the *greedy* criterion for the insertion of non-fixed requests provides more consistent results over the 30 instances. The non-fixed request insertion method clearly outperforms the other two insertion methods for the instances examined here.

Summary results are now provided for the instances with both 500 and 1000 requests. The *greedy* criterion for insertion with the dynamic insertion of all non-fixed locations, required a significant increase in computational time, due to the larger neighbourhood being explored at each insertion at each interval. It was thought that this method was therefore not appropriate for use in a real-time setting, hence is discarded in this analysis for the instance with 500 and 1000 instances.

Figure 7.4 provides average results for the instances with 500 and 1000 requests respectively, again the results are based on the average total travel distance achieved over all instances of that size.



(a) 500 requests

(b) 1000 requests

Figure 7.4: Total distance travelled for each dynamic insertion method for each criterion for insertion

It can be seen that the results agree with those in Figure 7.3 for the case of 100 requests. Therefore the dynamic insertion methods and criterion for insertion appear to provide consistent results over varying numbers of dynamic requests.

To note, when applying the *random* criterion for insertion the result reported is for a single run of the heuristic. Preliminary investigations showed that due to the small number of requests available at each insertion, there was very little variability in the results achieved over 100 runs of the method, therefore we will continue to use the solution obtained after a single run.

The next section will investigate the improvement heuristics previously considered in Section 6.6.

7.4 Improvement Heuristics

This section will re-introduce the improvement heuristics investigated for the DPDPTW in Section 6.6, namely the tabu search heuristic and the branch and bound heuristic, to investigate the improvement made to the solutions over time for the instances of Mitrovic-Minic et al. [2004]. The tabu search heuristic applies the same setting as in Section 6.6.

Figure 7.5 provides the average total travel distance, over the 30 instances with 100

requests, achieved when applying each of the improvement methods. Results are provided for each of the dynamic insertion methods, by each criterion for insertion (see Section 7.3).

For the case of 'Tabu', the tabu search heuristic is applied as an improvement phase at each interval after the dynamic insertion methods have been applied. For 'BB', the branch and bound heuristic is applied to the solutions achieved after the insertion methods as an improvement phase in place of the tabu search heuristic. The results are also provided for combining the tabu search heuristic and the branch and bound methods, 'Tabu + BB', as was the case in Section 6.6, where it was found combining these two methods in the improvement phase improved the results achieved.

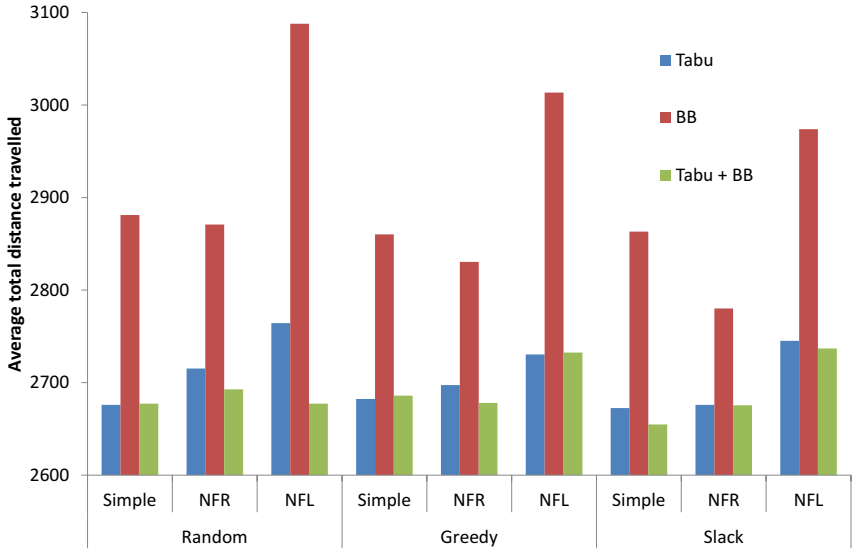


Figure 7.5: Summary Results of Dynamic Insertion Methods

For 'Tabu' the simple insertion method achieves the most promising results, specifically for the *slack* criterion. A reason for this could be that there is a larger neighbourhood of feasible moves available when requests are inserted using the simple criterion, as no attempt is made at this stage to re-order the locations in routes to accommodate the new requests. This could result in the tabu search heuristic having a larger number of opportunities for improvement.

It is clear that the results for 'BB' are worse than those of 'Tabu'. This time the insertion of all non-fixed requests which achieves the most promising results. This could be due to the fact that the branch and bound heuristic only looks to improve the ordering of locations within routes and is not able to improve the grouping of requests to routes. For this case, it would seem the final solutions achieved could be heavily

dependent on the initial solution constructed and it was known from Section 7.3 that the insertion of all non-fixed requests provided the best results.

A one-way analysis of variance for repeated measure has been performed to compare the results achieved by the 'Tabu + BB' method for the varying dynamic insertion methods and criteria for insertion. Results showed that at 5% significance the slack insertion of the non-fixed locations was significantly worse than the best 5 methods and that the slack simple insertion method was significantly better than the worst two methods. Comparing the 'Tabu method to 'Tabu+BB using a paired samples t-test, there was a significant difference in the results achieved at the 5% level for the random insertion of all non-fixed locations and greedy insertion of all non-fixed requests. However, there was no significant difference in the results for the other 7 variations, including that of the simple slack insertion which achieved the most promising results.

The results in Figure 7.5 show that once again combining the tabu search heuristic with the branch and bound heuristic at each interval provides the lowest total travel distance. This is comparable to the results achieved for the instances of Pankratz [2005b] investigated in Chapter 6, where results are provided in Section 6.7. This shows our algorithm is consistent over variations in types of instances.

It is interesting to note that for the case of the *greedy* insertion method, 'Tabu' provides better results than that of 'Tabu + BB'. This could be linked to the fact that the *greedy* criterion provides the initial solutions with the minimal total travel distance as it finds the best feasible move of all requests at each iteration. Therefore the ordering of locations to routes is better prior to the tabu search heuristic than for the other methods, resulting in better solutions being obtained early on in the scheduling horizon. However, this may lead to more requests becoming fixed early on in the scheduling horizon and fewer improvements being feasible later on in the scheduling horizon, resulting in a worse final solution.

The next section will investigate the improvement heuristic during the scheduling horizon to investigate how each method differs.

7.5 Comparisons between Improvement Methods

This section looks to compare the cost of a solution at each interval of the scheduling horizon for a representative instance with 100 requests. Figure 7.6 shows the total distance travelled in the solution at each interval for the case where only the tabu search heuristic is applied (Tabu) and for the case where both the tabu search heuristic

and the branch and bound heuristic are applied (Tabu + BB). These two methods have been chosen as they achieved the most promising results in Section 7.4.

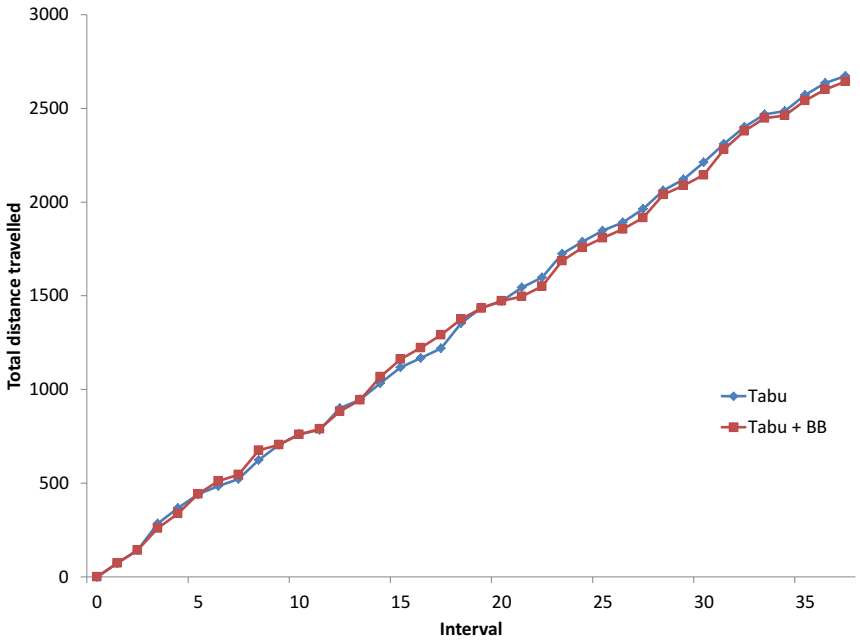
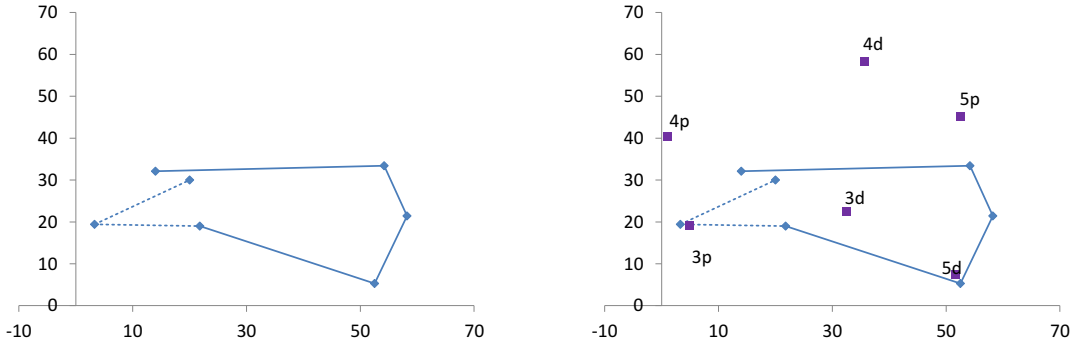


Figure 7.6: Comparison of Dynamic Improvement Methods - Rnd8_000

From the results provided in Figure 7.6 it can be seen that, for the particular instance considered, 'Tabu + BB' achieves the best results at the end of the scheduling horizon. However, the results vary throughout the scheduling horizon. To explain what is happening here, we will now compare the solutions obtained early on in the scheduling horizon at intervals 1, 2, 3 and 4, corresponding to times 15 minutes, 30 minutes, 45 minutes and 60 minutes respectively.



(a) Solution at Interval 1

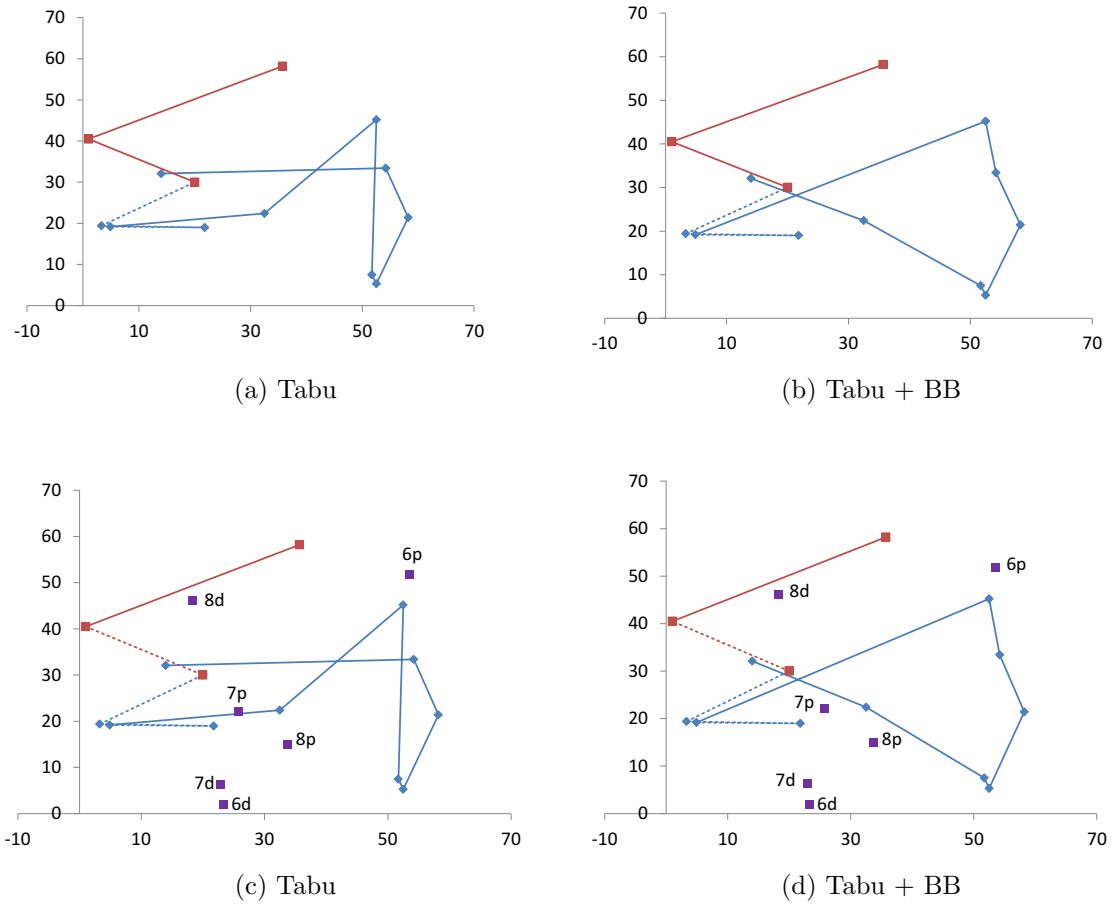
(b) New Requests at Interval 2

Figure 7.7: Starting Solution at Interval 1 and New Requests at Interval 2

Figure 7.7a shows the solution obtained at the end of the first interval, 15 minutes. The current solution for both cases is the same and consists of 3 requests and a total distance travelled of 141.85.

Figure 7.7b indicates the requests that are accumulated prior to the next re-start of the algorithm at time = 30 minutes. In total 3 new requests have arrived since the last update of the solution. These are labelled as requests 3, 4 and 5, with a 'p' denoting a pickup location and a 'd' for the delivery location.

Figures 7.8a and 7.8b show the solution obtained after these new requests have been incorporated for both cases of the improvement heuristics.



(a) Tabu

(b) Tabu + BB

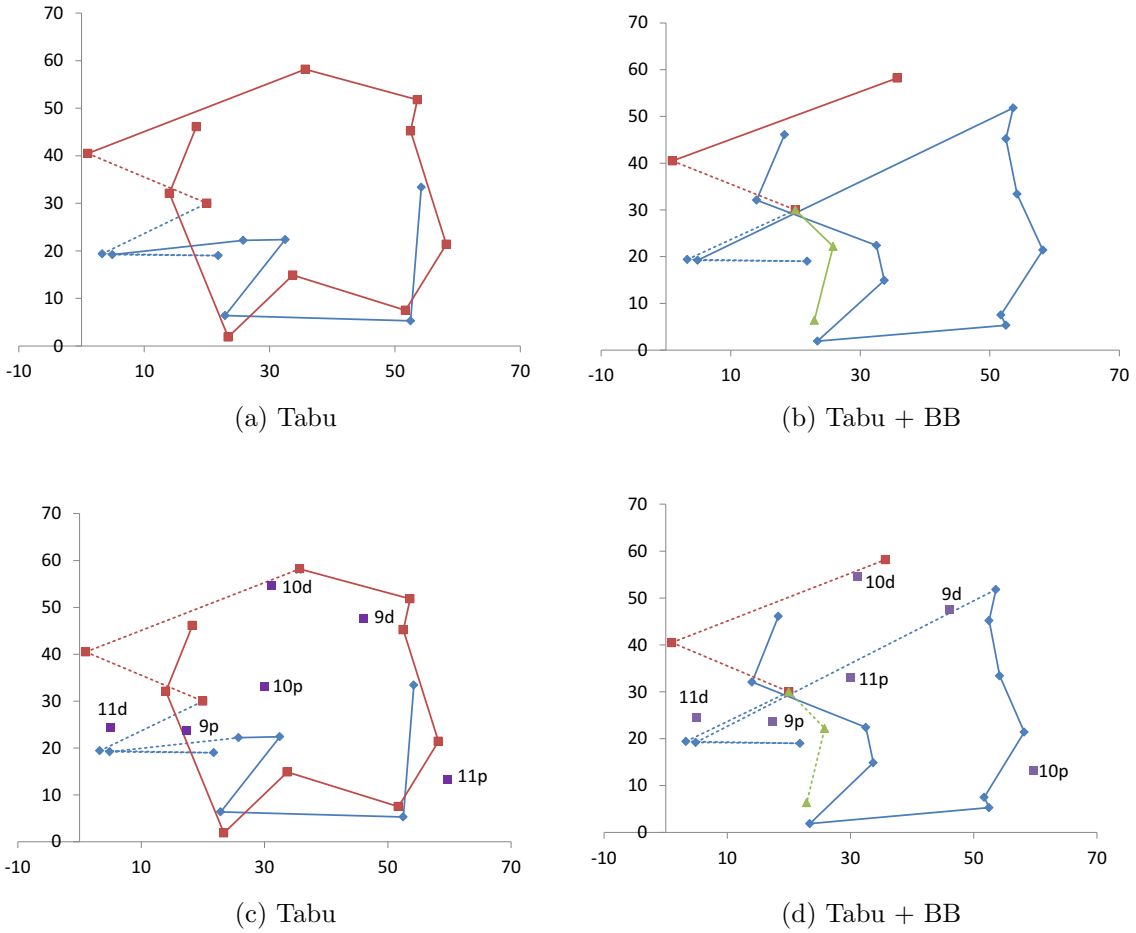
(c) Tabu

(d) Tabu + BB

Figure 7.8: Solution at Interval 2 and New Requests at Interval 3

From Figures 7.8a and 7.8b it can be that the solution produced for each method differs. For the case of applying 'Tabu + BB', this further improves the current ordering of the locations in the route. The cost of the solution by 'Tabu' is 283.96 and for 'Tabu + BB' is 259.27. A total saving of 24.69 in total travel distance is achieved.

At the third interval (time = 45 minutes), 3 new requests again arrive, these are labelled as requests 6, 7 and 8 and are shown in Figures 7.8c and 7.8d. Figures 7.9a and 7.9b provide the solutions obtained again for applying 'Tabu' and 'Tabu + BB'.



(a) Tabu

(b) Tabu + BB

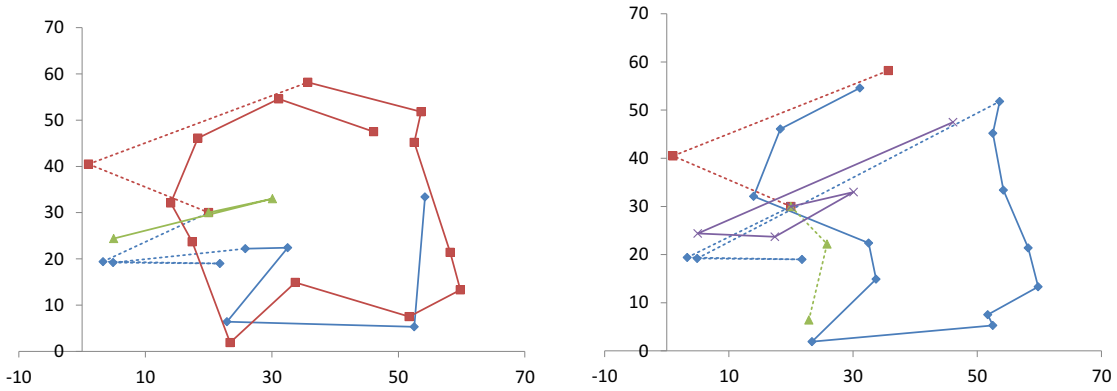
(c) Tabu

(d) Tabu + BB

Figure 7.9: Solution at Interval 3 and New Requests at Interval 4

From Figures 7.9a and 7.9b it can be seen that the 2 solutions produced now vary in the number of vehicles they require. The total travel distance in the solution where 'Tabu' is applied is 367.94 and for the case of 'Tabu + BB' is 338.20.

Finally, the requests arriving at 60 minutes are shown in the solutions provided in Figures 7.9c and 7.9d and Figures 7.10a and 7.10b provide the solutions obtained again for the 2 cases after the requests have been incorporated into the solution.



(a) Tabu

(b) Tabu + BB

Figure 7.10: Solution at Interval 4

These results show that a further addition of a new vehicle in the solution obtained by applying both the tabu search heuristic and the branch and bound heuristic. The total distance travelled for the solution provided in Figure 7.10a is 439.88 and for the solution in Figure 7.10b is 442.28.

This example highlights the large variation in the solutions achieved over a small period of the total scheduling horizon for the two different improvement phases. It is shown that the best method at a particular interval does not necessarily still achieve the best solution at later intervals. It also highlights that the best solution achieved varies between the methods early on in the interval and that it is heavily dependent on what requests are received within the next interval.

From Figure 7.6 it is known that the best solution at the end of the scheduling horizon was achieved by an improvement phase consisting of both the tabu search heuristic and the branch and bound heuristic, however this has varied over the entire scheduling horizon.

The next section will examine the number of intervals at which to re-start the algorithm to incorporate the dynamic requests.

7.6 Investigating the Number of Intervals

As previously stated, for the method of Mitrovic-Minic et al. [2004], their algorithm is re-started at 15 minutes intervals throughout the 10 hour scheduling horizon, giving a total of 40 intervals. It will be investigated as to whether an improvement can

be made to the solutions achieved by our algorithm by varying the time between re-starts of our algorithm. On the one hand, long intervals will allow a higher number of requests to be accumulated, leading to a higher number of opportunities for insertion and improvement, on the other hand this could lead to no longer feasibly servicing all requests.

Our algorithm comprises of the best found method achieved in Section 7.4, the simple dynamic insertion using the *slack* criterion and an improvement phase consisting of the tabu search heuristic and the branch and bound heuristic.

A range of intervals were analysed and it was found that, for less than 35 intervals an infeasible solution, with regards to servicing all requests, was achieved for at least one instance out of the 30 instances with 100 requests. This was due to a request arriving during an interval which due to the state of the current solution would need to have been inserted before the end of the interval for it to be feasibly serviced.

A summary of results for the average total travel distance achieved (TD), the standard deviation over the 30 instances (SD) and the average rank over all instances for varying numbers of intervals which provided the most promising results are provided in Table 7.2. Solutions are ranked 1 to 4, with a rank of 1 indicating that the solution achieved the minimum total travel distance of the 4 cases considered and a rank of 4 indicating that the solution achieved the highest total travel distance of the cases considered.

Number of intervals	TD	SD	Average rank
35	2672.70	135.38	2.47
40	2654.79	119.90	2.23
45	2671.69	141.95	2.50
50	2682.68	118.95	2.80

Table 7.2: Average TD, SD and Rank achieved for varying numbers of Intervals for the RND8 Instances

It can be seen that the 40 intervals suggested by Mitrovic-Minic et al. [2004] provides the most promising results overall, followed by 35 intervals and similarly 45 intervals. For the case of comparing 35 intervals with 40 intervals although the average total travel distance over all instances was less for the case of 40 intervals, for some instances a lower total travel distance is achieved for the case of 35 intervals. The same applies when comparing the case of 40 intervals against 45. It is also supported by the high standard deviation reported for the average result.

The results provided for each instance differ with regards to the number of intervals that gives the overall lowest total travel distance, this is supported by examining the average ranks provided in Table 7.2 where it can be seen that these are similar.

A one-way analysis of variance was performed for repeated measures to determine if there was a significant difference between the results achieved for the varying intervals. It found that there is no significant difference between the results achieved for each interval at 5% significance, with a p-value of 0.538 ($F = 0.728$). We will therefore continue to investigate our algorithm, incorporating the dynamic requests at each 15 minute interval to ensure results are comparable to those of Mitrovic-Minic et al. [2004].

The next section will summarise the results for our algorithm and comparisons are made with best known solutions from the literature.

7.7 Summary of Results

This section will compare the best results achieved by our algorithm for the instances of Mitrovic-Minic et al. [2004] who apply a double-horizon based heuristic with the idea that better management of slack time in the distant future may help to improve the overall solutions obtained.

For the algorithm of Mitrovic-Minic et al. [2004] an insertion heuristic is applied at every 15 minute interval similar to the *slack* criterion for insertion found to achieve the best results in Section 7.4. This time the insertion criterion is applied to all new requests accumulated since the last re-start followed by the re-insertion of all scheduled requests whose pickup location has not been serviced, similar to the non-fixed request insertion method. A tabu search heuristic then runs in the period between these intervals and runs close to 15 minutes. The tabu search heuristic of Mitrovic-Minic et al. [2004] is a simplified version of the method introduced by Gendreau et al. [1998] which uses neighbourhoods defined by means of ejection chains. It has been extended to apply a greedy approach when finding the best insertion of a request within a route as is the case with our dynamic insertion heuristics, rather than an approximation algorithm as in Gendreau et al. [1998]. See Section 5.4.3 for more details on this.

The results of our algorithm are therefore directly comparable to those of Mitrovic-Minic et al. [2004] and the procedure followed is similar to that outlined in Algorithm 16, except the algorithm is now re-started at set intervals, rather than each time a new request arrives. Table 7.3 provides a comparison for the instances with 100 requests, Table 7.4 for the instances with 500 requests and Table 7.5 for the instances with 1000

requests.

	Mitrovic-Minic et al.	Our algorithm	% Decrease
Rnd8_10h_100_000	2656.41	2642.97	1%
Rnd8_10h_100_001	2700.60	2605.27	4%
Rnd8_10h_100_002	2774.64	2797.31	-1%
Rnd8_10h_100_003	2853.89	2695.67	6%
Rnd8_10h_100_004	2787.88	2727.12	2%
Rnd8_10h_100_005	2965.55	2790.13	6%
Rnd8_10h_100_006	2631.34	2596.22	1%
Rnd8_10h_100_007	2674.47	2725.43	-2%
Rnd8_10h_100_008	2888.39	2726.56	6%
Rnd8_10h_100_009	2978.87	2778.56	7%
Rnd8_10h_100_010	2576.58	2523.94	2%
Rnd8_10h_100_011	2812.76	2638.94	6%
Rnd8_10h_100_012	2677.90	2658.15	1%
Rnd8_10h_100_013	2703.01	2594.72	4%
Rnd8_10h_100_014	3016.79	2740.93	9%
Rnd8_10h_100_015	2759.91	2684.19	3%
Rnd8_10h_100_016	2694.01	2539.09	6%
Rnd8_10h_100_017	2894.00	2698.98	7%
Rnd8_10h_100_018	2696.56	2703.61	0%
Rnd8_10h_100_019	2537.65	2508.39	1%
Rnd8_10h_100_020	2819.49	2561.16	9%
Rnd8_10h_100_021	2704.22	2693.24	0%
Rnd8_10h_100_022	2860.85	2848.82	0%
Rnd8_10h_100_023	2479.15	2388.92	4%
Rnd8_10h_100_024	2894.79	2816.63	3%
Rnd8_10h_100_025	2543.57	2396.13	6%
Rnd8_10h_100_026	2889.89	2788.19	4%
Rnd8_10h_100_027	2780.39	2629.16	5%
Rnd8_10h_100_028	2653.85	2454.65	8%
Rnd8_10h_100_029	2763.60	2690.5	3%
Average	2755.70	2654.79	4%

Table 7.3: Comparison of TD of Our Algorithm to that of Mitrovic-Minic et al. [2004] for 100 requests for the RND8 Instances

From the results provided in Table 7.3, on average an improvement of 4% is achieved in the instances with 100 requests. A lower total travel distance is achieved in 27 out of a total of 30 instances. Applying a matched paired samples t-test to the difference between these results, a p-value of < 0.000 (t=6.574) is obtained, indicating a significant difference.

	Mitrovic-Minic et al.	Our algorithm	% Decrease
Rnd8_10h_100_000	10053.62	8739.43	13%
Rnd8_10h_100_001	9699.48	8349.54	14%
Rnd8_10h_100_002	9608.40	8202.41	15%
Rnd8_10h_100_003	9807.06	8350.71	15%
Rnd8_10h_100_004	10176.05	8832.41	13%
Rnd8_10h_100_005	10133.55	8733.24	14%
Rnd8_10h_100_006	10045.82	8485.01	16%
Rnd8_10h_100_007	9978.97	8753.23	12%
Rnd8_10h_100_008	9651.25	8513.46	12%
Rnd8_10h_100_009	9707.42	8865.12	9%
Rnd8_10h_100_010	9200.16	8458.44	8%
Rnd8_10h_100_011	9710.40	8586.22	12%
Rnd8_10h_100_012	9748.16	8600.62	12%
Rnd8_10h_100_013	9961.84	8380.88	16%
Rnd8_10h_100_014	9560.35	8390.46	12%
Rnd8_10h_100_015	9296.75	8448.59	9%
Rnd8_10h_100_016	9784.43	8500.53	13%
Rnd8_10h_100_017	9917.51	8411.73	15%
Rnd8_10h_100_018	9729.92	8554.13	12%
Rnd8_10h_100_019	9721.48	8297.99	15%
Rnd8_10h_100_020	10118.79	8742.17	14%
Rnd8_10h_100_021	9458.99	8742.4	8%
Rnd8_10h_100_022	10126.10	8739.42	14%
Rnd8_10h_100_023	9879.78	8533.37	14%
Rnd8_10h_100_024	9313.77	8572.88	8%
Rnd8_10h_100_025	9637.84	8323.2	14%
Rnd8_10h_100_026	10349.09	8684.06	16%
Rnd8_10h_100_027	9925.99	8411.79	15%
Rnd8_10h_100_028	9823.70	8572.82	13%
Rnd8_10h_100_029	9997.84	8066.93	19%
Average	9804.15	8528.11	13%

Table 7.4: Comparison of TD of Our Algorithm to that of Mitrovic-Minic et al. [2004] for 100 requests for the RND8 Instances

Comparing the results of our algorithm to those obtained by Mitrovic-Minic et al. [2004] for 500 requests, an overall average improvement of 13% is achieved. This further improves on the comparison of the instances with 100 requests and a solution has been achieved with a lower total travel distance for all instances with 500 requests. Applying a matched paired samples t-test on the difference between the results a p-value of < 0.000 (t=24.658) is achieved, therefore again a significant difference testing at 5%.

Table 7.5 provides results for the instances with 1000 requests. Detailed results for all

instances for Mitrovic-Minic et al. [2004] are not available, hence only the average value is compared. The results of our algorithm for each instance can be found in Appendix B, Table 7.6. Again a clear improvement can be seen, this time an average percentage decrease of 19% in total distance travelled.

	Mitrovic-Minic et al.	Our algorithm	% Decrease
Average	17610.45	14188.15	19%

Table 7.5: Comparison of Average TD of Our Algorithm to that of Mitrovic-Minic et al. [2004] for 1000 requests for the RND8 Instances

Another set of instances were generated by Mitrovic-Minic et al. [2004] to compare the results of their algorithms. The two sets differ only in the distribution and width of the time windows assigned to the requests. The distribution of requests in the second set of instances is: 10% 1 hour requests, 20% 2 hour requests, 30% 4 hour requests, 30% 6 hour requests and 10% 8 hour requests; therefore instances in the second set have a wider range and longer duration of time windows than those in the first set. There are again instances with 100, 500 and 1000 requests, there being 30 instances of each size.

A summary of the results achieved by our algorithm for the second set of instances is provided in Table 7.6, where the average total travel distance achieved over all instances by our algorithm for 100 and 500 requests is stated. Detailed results for each instance are not available in the case of the second set of instances for Mitrovic-Minic et al. [2004], only the average value is provided, and no results are given at all for the case of 1000 requests.

	Mitrovic-Minic et al.	Our algorithm	% Decrease
100	2518.54	2410.92	4%
500	9104.13	7591.82	17%

Table 7.6: Comparison of Average TD of Our Algorithm to that of Mitrovic-Minic et al. [2004] for 100 and 500 requests for the RND9 Instances

As can be seen from the results provided in Table 4, our algorithm improves on Mitrovic-Minic et al. [2004]. The percentage decrease is greater than that achieved for the first set of instances in Table 7.5 again showing the consistency of our approach. It should be noted that the computational times of our algorithm at each interval during the scheduling horizon for 100 requests are comparable to those previously achieved in Section 6.7, for 500 requests the time computational time required

at each interval is ≈ 2 seconds and for 1000 is ≈ 30 seconds. Therefore our algorithm remains appropriate for use in a real-time environment.

7.8 Comparisons to the Static Problem

A comparison will now be made to the results achieved if all requests had been known in advance. This will provide an insight into how our dynamic algorithm is performing compared to what would have been achieved if all information had been known prior to the beginning of the scheduling horizon. Table 7.7 provides results for the first set of instances and Table 7.8 for the second set. Comparisons are made against the results achieved by Mitrovic-Minic et al. [2004] for 100 and 500 requests. No results were provided for 1000 request for either set of instance.

	Static		Dynamic	
	Mitrovic-Minic et al.	Our algorithm	Mitrovic-Minic et al.	Our algorithm
100	2325.39	2098.59	2755.70	2654.79
500	8769.96	7002.00	9804.15	8528.11

Table 7.7: Comparison of Average TD of Our Algorithm to that of Mitrovic-Minic et al. [2004] for 100 and 500 requests for the Static problem and the Rnd8 Instances

As expected the results for the static variant of our algorithm improve on those where the requests arrive dynamically throughout the scheduling horizon. For the instances with 100 requests the percentage increase from the static solution compared to the dynamic is ≈ 27%. For the case of 500 requests the dynamic problem increased the average total distance travelled by ≈ 22%.

Our dynamic algorithm improves on the solutions achieved by Mitrovic-Minic et al. [2004] when taking all information known in advance, here the percentage increase in the total travel distance is 19% and 12% for 100 and 500 requests respectively. This could show the weakness of the approach by Mitrovic-Minic et al. [2004] to handling larger numbers of requests at each interval, hence the poor solutions achieved when all information is known in advance.

	Static		Dynamic	
	Mitrovic-Minic et al.	Our algorithm	Mitrovic-Minic et al.	Our algorithm
100	2143.16	1856.02	2518.54	2410.92
500	8022.20	6197.66	9104.13	7591.82

Table 7.8: Comparison of Average TD of Our Algorithm to that of Mitrovic-Minic et al. [2004] for 100 and 500 requests for the Static problem and the Rnd9 Instances

For the second set of instances provided in Table 7.8 similar results are achieved. This time for 100 requests the percentage increase in cost for the dynamic solution compare to the case where all requests are known prior to the beginning of the scheduling horizon is $\approx 29\%$ and for 500 requests is $\approx 22\%$. The results of Mitrovic-Minic et al. [2004] are $\approx 16\%$ and $\approx 14\%$ greater than the static solution for 100 and 500 requests respectively.

7.9 Chapter Summary

To conclude the chapter it can be seen that our algorithm developed in Chapter 6 continued to perform consistently well on the slightly different problem formulation (and problem instances) of Mitrovic-Minic et al. [2004].

Again, a simple insertion of the dynamic requests under a *slack* insertion criterion and an improvement phase consisting of the tabu search heuristic and branch and bound heuristic provided the most promising results.

Indeed, improvements in results were achieved in instances with 100, 500 and 1000 requests against those provided in Mitrovic-Minic et al. [2004] over two sets of instances. Comparisons to the results achieved if all information had been known in advance show that once again our algorithm outperforms that of Mitrovic-Minic et al. [2004] for both sets of instances.

The next chapter looks to further apply the research carried out so far in this thesis to a real-life variant of the problem, in particular a health courier service.

Chapter 8

The Health Courier Service

8.1 Introduction

A study by Landry and Philippe [2004] showed that healthcare organisations often overlook the role of logistics and that logistics related activities account for approximately 46% of a hospital's total budget. Therefore, a better allocation of resources, could result in a significant saving.

This chapter applies the research undertaken so far in this thesis to a real-life variant of the problem found in a healthcare organisation. The problem to be considered is that of the Welsh Ambulance Service Trust (WAST) health courier service (HCS), which provides services to support local communities, health alliances, local health groups and general practitioners (GPs), through the transportation of items such as mail, specimens and blood.

Two problems are faced by the service. Firstly, the scheduling of static requests for schedules completed daily (this is a variant of the PDPTW introduced in Chapter 3). Secondly, the scheduling of real-time priority requests received through a 24/7 service (this is a variant of the DPDPTW introduced in Chapter 6).

The literature related to OR methods applied to transportation in healthcare is limited. The majority of the research available concentrates on the transportation of patients, rather than goods, indicating that further research into this area could be beneficial. An overview of the literature will be provided in Section 8.2.

The remainder of this chapter is structured as follows. An introduction to the HCS is outlined in Section 8.3 and a description of the problem is provided in Section 8.4. Preliminary investigations are then performed using existing data from the HCS. The

process of generating the data required for the travel distances and travel times is outlined in Section 8.5 and a number of the static daily schedules of the HCS are analysed in Section 8.6, along with the adaptations made to our algorithm to incorporate this.

The real-time 24/7 service is investigated in Section 8.7, where an analysis is performed on existing data. The chapter is concluded in Section 8.8 where ideas for further research into this problem are provided.

The aim of this chapter is to provide initial analysis to determine if it is possible for the HCS to service a higher proportion of the dynamic requests it receives than is currently achieved.

8.2 Relevant Literature

As mentioned in Section 8.1, there is limited research available for the transportation of goods for a healthcare organisation and in particular for a real-time variant of the problem. Most of the literature in this field investigates the transportation of patients, rather than goods, and looks at a problem where all information is known in advance. This section will review the literature that is available in the hope of gaining useful insights into how the specific constraints faced by a healthcare organisation, can be incorporated into our algorithm.

The relevant literature with regards to the transportation of patients within a hospital is reviewed first. The basic problem can be viewed as a dial-a-ride problem (DARP) (see Section 2.7). The main difference between this and the PDPTW is that people are transported instead of goods. The objective function therefore is usually a combination of minimising the transportation costs and maximising patient satisfaction. In a hospital context, additional data is provided for each request such as the varying modes of transport taken by patients (e.g. a wheelchair or a stretcher) or the priority of a request.

A recent study into the dynamic transportation of patients in hospitals was conducted by Beaudry et al. [2010]. It aims to provide an efficient and reliable transport service to patients between several locations in a hospital campus. For this case, transportation is provided by ambulances which can usually be shared by several patients simultaneously. A two-phase heuristic is proposed which starts with a simple insertion method and ends with a tabu search heuristic. The algorithm employs a DF strategy (see Section 5.4.4), meaning that a vehicle drives from a location at its earliest departure time. In this case, to prevent an empty vehicle from waiting at a pickup location, it is sent back to the

depot. This is the case in many real-life transportation services and allows the driver to complete ad-hoc tasks at the depot. This is also the case for the HCS considered.

The dynamic problem of transporting patients between hospitals is looked at by Kergosien et al. [2010], in particular, for the hospital complex of Tours in France. For this variant of the problem, an ambulance central station is used to plan the transportation of patients between care units which require a vehicle. Specific constraints of this problem include that each request requires a specific type of vehicle and a vehicle can only transport one patient at a time. Therefore, once a patient is picked up, the vehicle must travel immediately to that patients delivery location. There is the possibility to seek the help of private ambulance companies if necessary, at an additional cost, so serving all requests is not a hard constraint. This is also the case for the HCS considered and will be discussed further in Section 8.7.

For the case of Kergosien et al. [2010], a priority is assigned to each request, where its time window depends on the priority. This is something that can be incorporated into the problem studied in this research. Here, there is no unique depot, patients move between two points with several depots and diversions are also allowed. The general algorithm introduced is a tabu search heuristic based on that of Gendreau et al. [1999].

The static problem of routing patients within a hospital is considered by Turan et al. [2011]. Here, patients have to be transported between different units for a fixed appointment by a porter. An optimisation model is developed to solve the problem explicitly which is then extended to improve both patient satisfaction and the use of hospital resources. The first extension to the model ensures that one porter is assigned to one patient, as it is considered beneficial for the patient if the same porter can escort them on both journeys. The second extension looks to temporarily send porters back to their home depot to be assigned other tasks if their waiting time exceeds a specified value. This extension is similar to that of Beaudry et al. [2010].

A DARP for the transportation of patients within a healthcare organisation by Melachrinoudis et al. [2007], and again in Melachrinoudis and Min [2011], focuses on the centre for addictive behaviour, health and recovery services inc. (CAB) based in Massachusetts. The service provides transportation of scheduled trips for detoxification, intermediate medical care, halfway house interviews, medical/psychological appointments, homeless sheltering, and discharges. The CAB provides treatment and rehabilitation through 9 treatment centres scattered around the Boston Metropolitan area and, prior to the research, no communication existed among the different centres with regards to their transportation services. The aim of the research is to create routes between centres through a centralised dispatching system, where a vehicle from one

centre, can service patients from other centres. A tabu search heuristic, which applies a simple shift operator, improves initial results and shows routes can be improved by using a centralised dispatching system.

To conclude the review into the transportation of patients in a healthcare environment we now consider a similar problem studied more frequently in the literature, the transportation of the handicapped and elderly. The problem faced by the Copenhagen Fire-Fighting Service was studied by Madsen et al. [1995] and involves both multiple capacities and multiple objectives. Toth and Vigo [1997] suggest a parallel insertion heuristic along with a tabu search heuristic for the problem. They consider two modes of transport, either where patients require seating, or are seated in a wheelchair. Handicapped people's transport in Berlin is studied by Borndörfer et al. [1997], and in particular the dial-a-ride system known as Telebus. This is solved via a branch-and-cut algorithm. A GGA is implemented by Rekiek et al. [2006] for a handicapped persons transportation problem in the city of Brussels, Belgium. Finally, a more recent study for the Austrian Red Cross was considered by Parragh [2011]. Both heterogeneous requests and vehicles are introduced and the problem is solved via a branch-and-cut algorithm and a VNS, initially designed for the standard DARP by Parragh [2009].

There is limited literature available on the transportation of goods within or between locations in a healthcare organisation. An example of scheduling pickup and delivery requests for both patients and goods within a hospital is considered by Fiegl and Pontow [2009]. This is based on the Natters State Hospital in Austria and consists of transporting patients, all types of medical items e.g. records, forms, medicines, laboratory samples and goods such as mail and waste. Two types of request exist: ad-hoc requests that arrive in real-time and standard requests that have to be executed daily or weekly at a given time. This is similar to the HCS to be considered here. The most significant difference in this problem to a standard routing problem is that the distance travelled within a hospital, in comparison, is very small. Therefore a porter, or a vehicle in routing terms, is available again in a short time after service and can react quickly to changes that may occur due to the arrival of new requests.

An example of scheduling a static pickup and delivery problem within a health maintenance organisation is considered by Shang and Cuff [1996]. In this case, vehicles are leased to transport patients' records, equipment and supplies between locations. Transfers can occur at any location, for any item, and between any two vehicles. The approach taken constructs mini-routes, which are small subsets of requests that look like they should be serviced by the same vehicle. For example, if two requests have the same pickup location, they can be picked up simultaneously. The best mini-route is chosen and if an available vehicle exists, the best mini-route will be assigned to the

available vehicle. If all vehicles are occupied, an insertion procedure is used to identify the vehicle that can service the mini-route with minimal cost.

Finally, a computer based planning system, Opti-*TRANS*©, that supports all phases of transportation in a hospital was designed by Hanne et al. [2009]. Here, vehicles either represent ambulances or staff on foot. The system is configured to search for fastest routes and can take into account traffic conditions at different times of the day. This may be something to consider for the HCS. Opti-*TRANS*© includes several optimisation routines that can be combined depending on the time available for planning. These include a load balancing strategy, which assigns a new request to the vehicle with the earliest availability for service to begin, among those with the smallest workload and a best selection strategy, which identifies the best feasible route, then inserts a request using varying criteria. An evolutionary algorithm (EA) controls the assignment of requests to vehicles, where mutations are performed by randomly assigning a new vehicle to a request and the selection criteria is based on the objective function. For periods of peak demand, the load balancing strategy combined with the best insert criterion provides the most promising solutions; the EA is more suitable for periods of low demand.

This review highlights the limited literature available for applying the static and dynamic PDPTW to a healthcare environment. A description of the problem faced by the HCS to be considered is provided in the next section.

8.3 The Health Courier Service

The WAST HCS which is to be investigated in this research, provides a non-patient transport service to the National Health Service (NHS) Trusts and other non-patient transport stake holders across Wales.

The re-organisation of NHS Wales, which came into effect on October 1st 2009 created single local health organisations responsible for delivering all healthcare services within a geographical area. NHS Wales now delivers services through seven Health Boards and three NHS Trusts in Wales.

An overview of the pickup and delivery services provided includes:

- Specimen collection
- Compliant urgent blood & blood products transportation
- CSSD (sterile equipment) transportation
- Laundry services
- Pathology services (laboratory samples)
- Bio mechanical engineering transportation
- Clinical waste services (including dental waste)
- Needle exchange services
- Sorting, and the delivery/distribution of internal NHS mail & notes
- Welsh Government civil contingency disaster management services
- Pharmacy distribution
- Controlled drug distribution
- Staff transportation
- Emergency hospital equipment transportation
- Nuclear medicine & radioactive waste transportation
- Movement of dangerous goods on behalf of NHS Wales

The services demanded and those provided differs depending on the Health Board requirements in each area. Figure 8.1 shows the 7 Health Boards in Wales and the current service delivery operated in each area is provided in Table 8.1. A full HCS includes the transportation of specimens, blood, blood products, pharmacy, notes and mail and 24/7 indicates the service provides a 24/7 service for the transportation of priority requests.

Health Board	Type of service operated
Cardiff & Vale	Full HCS, 24/7 & CSSD
Aneurin Bevan	Full HCS, CSSD
Abertawe Bro Morgannwg	Full HCS, controlled drug distribution
Betsi Cadwaladr	Full HCS, needle exchange & dental waste
Hywel Dda	Full HCS & CSSD (Pembrokeshire), specimens (Ceredigion)
Cwm Taf	Laundry transport

Table 8.1: Current Areas of Service Delivery for WAST HCS

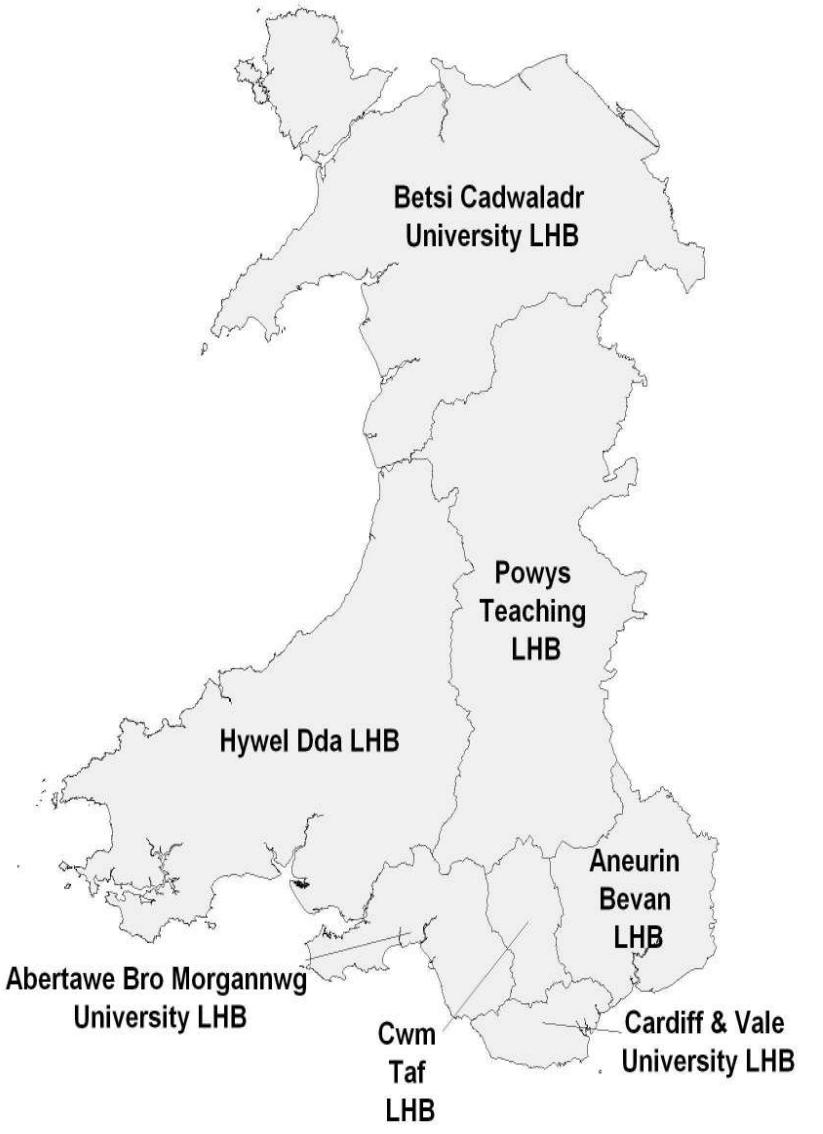


Figure 8.1: Local Health Boards in Wales

For this research we will concentrate on the services undertaken in the Cardiff and Vale University Health Board, who provide a full HCS. The service also provides assistance to the Central Sterile Service Department (CSSD), for the transportation of sterilised medical devices, equipment and consumables. The Cardiff and Vale University Health Board is one of the largest NHS organisations in Wales and provides health services to a population of around 465,700 people. It is the only Health Board in Wales whose HCS provides a 24/7 service for an immediate response to urgent specimen transport and emergency blood bank responses.

Taxis have been used in the past to carry these, and whilst at the moment it is legal to convey small amounts, it is potentially very damaging if these were to go missing in a

taxi cab. There are also clinical governance risks. The 24/7 service is able to manage these and other goods including blood covered by legislation.

The next section will provide a detailed description of the problem to be considered - in particular, the constraints specific to the HCS.

8.4 Problem Description

There are two main requirements encountered by the HCS considered. Firstly, the static schedules of the service which are carried out daily. These are mainly required to service the hospitals, GPs, health centres and medical practices, by transporting items such as mail and specimens across the area. This functionality accounts for approximately 80% of the service's total workload. These routes service approximately 160 locations per day, throughout both Cardiff and the Vale of Glamorgan. Most of the schedules are repeated routes, visiting major hospitals many times during the day.

A summary of the current fixed schedules undertaken by the HCS is provided in Appendix D. There are 16 schedules dedicated specifically to individual tasks such as the transportation of sterile instruments or laundry. There are also 13 schedules for varying courier service routes which are to be investigated in Section 8.6.

The second requirement is its 24/7 service. Based on the information that we have, this function of the HCS currently services around 17 requests per day, mainly those with a high priority. However this estimate is based on the limited data that we have available and is thought to be much higher with a large amount of variation between numbers of requests on consecutive days. The service has 1 vehicle whose sole use is for the 24/7 requests; it is a multi-purpose cab which allows carriage of goods, as well as 4 passenger seats for staff and patient transport. Vehicles have mobile communications, hence they are controlled via the 24/7 call centre, in order to deal with the real-time requests and changes to schedules.

At present we are told by the service that the 24/7 vehicle cannot feasibly service all requests that arrive. This is mainly due to the fact that if multiple urgent requests arrive at the same time they cannot be serviced by the same vehicle. It is often the case that priority requests are sent via taxi, or lost to other courier services. It is envisaged by the service that, through better scheduling techniques, all requests not feasibly serviced by this vehicle can be inserted into the current static schedules. At present this is not achieved.

In this thesis an initial analysis will therefore first be performed to determine if it

is possible for the HCS to service a higher proportion of the dynamic requests than currently. Investigations will be performed for a number of static schedules currently executed by the service. Here, the problem can be seen as a static PDPTW and methods applied in both Chapters 3 and 4 can be incorporated. The aim is to both improve the current schedules and to investigate the slack time currently available within these schedules. (A better understanding of the slack time could lead to better opportunities to insert the dynamic requests arriving to the 24/7 service.)

Following this, the requests arriving to the 24/7 service over a given period will be analysed. Summary statistics will be provided and scenarios will be tested to determine how many requests the 24/7 vehicle can feasibly service in a given period.

The basic problem faced by the HCS can be viewed as a PDPTW, but it is in fact considerably more complicated, due to the healthcare-specific constraints. These constraints will now be outlined, highlighting the differences to the static and dynamic problems previously defined in Sections 3.2 and 6.1.

For this case, distance is no longer equal to time, and both are no longer symmetric; this is due to the fact they are generated from real-life routes between locations (see Section 8.5). For example, they take into account one way streets. Our algorithm is currently able to incorporate this.

For the dynamic requests, servicing all requests is no longer a constraint. There now exists the possibility to use private companies such as taxis to service the dynamic requests, at an additional cost. All requests for the static schedules have to be serviced, however.

Each request arriving to the 24/7 service is given a priority indicating its degree of urgency. The order of service priorities is:

1. Blood components
2. Speciments/tissue/foetus
3. Isotopes
4. Drugs
5. Surgical instruments
6. Medical record/x-rays
7. Staff/patients
8. Mail

This will be used to define a time window for both the pickup and delivery location. Time windows will also be generated for the requests in the current fixed schedules.

Time windows are considered hard, as previously assumed in this research. We consider this to be a reasonable assumption, for example, an operation cannot be delayed due to the late arrival of blood or results. These will be incorporated when generating the instances for both problems in Sections 8.6 and 8.7.

For a courier service route, each vehicle is available during a given period of the day, with one or more scheduled service interruptions of fixed duration. These 'break' periods (such as a lunch break) need to be taken at the depot. To incorporate this into our algorithm these can be inserted as a 'dummy' request into each route, with the pickup location and the delivery location being the depot and the service time being equal to the specified break period. A time window will be assigned to the request, identifying the deviation permitted from the specified break time.

For this problem, there is a heterogeneous fleet of vehicles, as vehicles can be specialised for carrying particular equipment, have alternative ways of loading, have different transportation modes and have a different capacity. As we are to consider only the courier service routes for the static schedules, and not the routes which require a specialised vehicle, it can be assumed that the vehicles are homogeneous and all schedules will begin and end at the depot. Although, considering a heterogeneous fleet of vehicles could be incorporated into our algorithm.

The objective function consists of minimising fleet operating costs including total travel time, total travel distance, the number of vehicles, staffing costs and vehicle inactivity periods. For the courier service routes considered, the operating costs, with regards to the number of vehicles and staffing costs, are fixed. It is the travel times, travel distances and vehicle inactivity periods which can be controlled. Once again, the objective will therefore be to minimise the total distance travelled by the vehicles, as it is thought that this is strongly correlated with the other two objectives. Another option here could be to use a weighted sum objective function of a number of terms, or to use multi-objective optimisation.

The service does not currently consider transfer opportunities between vehicles. Based on the current literature (see Shang and Cuff [1996]) it is thought that this could lead to an improvement to the service, but this is outside the scope of this research.

The next section will discuss the generation of the data for the travel times and distances to be generated to investigate the problem further.

8.5 Generating the Travel Times and Travel Distances

There are in total 133 locations serviced by the Cardiff and Vale University Health Board HCS including 9 hospitals, 17 health centres and numerous medical practices and GP surgeries. Other locations include Cardiff North Renal Dialysis Unit, University of Wales Institute Cardiff Podiatry department, the Welsh Blood Service and other continuing care and treatment centres.

To investigate the HCS further, both travel times and travel distances between each set of locations are obtained. The travel time (minutes) and the travel distance (kilometres) are obtained using a tool developed by Knight et al. [2012], for an ambulance location problem for WAST. The tool utilizes Google Maps Javascript API functionality and allows the user to easily navigate locations to be serviced directly within the interface, returning the travel time and travel distance matrix via geocoding. Travel times are rounded to the nearest whole integer and travel distances are rounded to 2 decimal places. More information can be found in the online user guide by Smith et al. [2011].

To calculate travel times, Google Maps uses speed limits provided by its data providers which generally use information from road signs or public records. When Google Maps plots directions, it breaks down the trip into individual segments, indicating how long you will travel on each road, street or motorway. It multiplies the amount of time on each segment by the speed limit for that segment, thus, if travelling for 75 miles on a road with a 50mph speed limit, it calculates a 90 minute time for that segment. It then adds up the travel times for all segments to generate the estimated trip time. Thus, the trip time assumes driving at the posted speed limit at all times. In practical terms this is not often achieved, hence, Google's trip time estimates need to be validated further. This model could be extended to allow the travel times to be generated based on the time of day, however, this is outside the scope of this research so has not been utilised further at this point.

Figure 8.2 provides an example of the output produced by the Google Maps tool showing the locations to be serviced by the HCS. It can be seen that a high number of the locations are clustered around the centre of Cardiff.

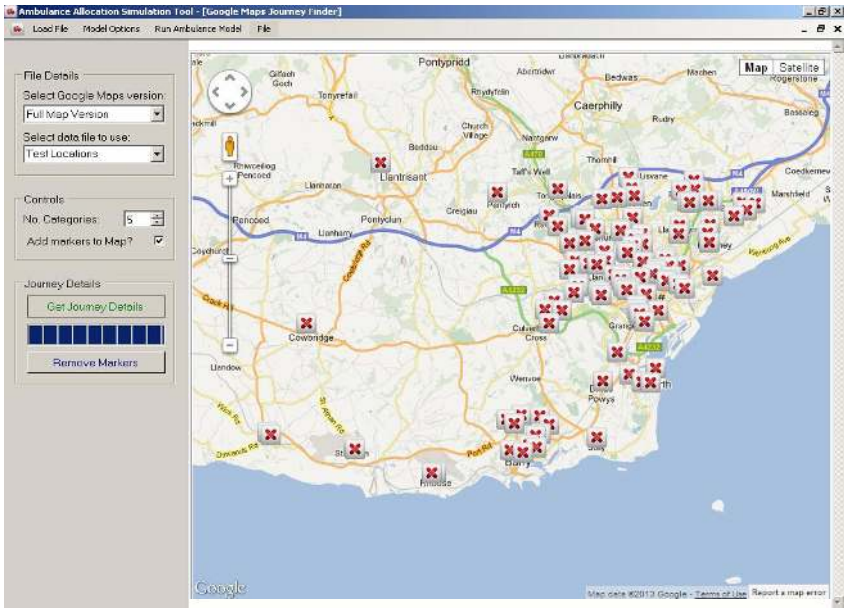


Figure 8.2: Output from the Google Maps tool

To validate the times and distances generated by the tool, the next section will investigate schedules for four of the known courier service routes currently executed by the service.

8.6 Investigating the Fixed Schedules

To validate the travel times and travel distances to be used by our algorithm, four of the static schedules used by the HCS for courier service routes will be investigated. The aim is to re-create the existing schedules followed by the service to identify any differences between the data that has been generated by the Google Maps tool and what is currently being observed.

The schedules followed at present have been developed over time by the drivers who carry out the same routes on a daily basis. Therefore, each driver has a good knowledge of the fastest routes between locations and also experience in avoiding congested routes, at specific times of the day. The driver has factored this information, including the corresponding time it takes to complete service at each location, to accurately provide the arrival times currently executed in the existing schedules.

The main issues faced in converting the existing schedules into PDPTW instances is that service times are not provided for each request. The time of arrival at each

location and a list of the services that are to be carried out at that location is all that is provided. A copy of the existing schedules followed by the service to be investigated in this section can be found in Appendix E.

At some locations, drivers may have to visit a specific area at that location. For example, at a hospital, a driver may have to visit the post room, specimen reception or theatre reception, all of which could be located in different areas. Service times will therefore differ depending on the details of the service. In the majority of cases, there are multiple pickup or delivery requests serviced at a single location and this will also affect the service time. If two services need to take place at the same department, then it is thought this should not increase the service time from a single service. However, if two services need to take place at two different departments, then this will require an additional service time.

The four static schedules that we investigate vary by duration, number of requests and type of request. They have been selected as they represent the varying alternative shifts for the courier service routes. Therefore each schedule represents a single courier service route carried out by one vehicle. When making a comparison between the times produced by the Google Maps tool and the existing schedules, we will compare the arrival times at each unique site visited, rather than pickup and delivery times for each request. Each of the schedules will now be discussed in more detail.

Schedule S2 consists of transporting all items for the HCS to and from the hospitals and major surgeries. The shift duration is 8:30am to 5:00pm, Monday to Friday. This is the most common shift pattern, with a 1 hour break period taken at the depot. In total there are 37 pickup and delivery requests, i.e. 78 locations to be visited, with two 'dummy' requests created to represent the start and end time of the shift and the break period. There are 29 unique sites to be visited on this route where the arrival times at each location will be compared to those found in the existing schedule.

Schedule S5 is a half-day shift which starts at 8:30am and finishes at 1:30pm on a Saturday. It is a repeated route of a number of hospitals consisting of only 7 requests, resulting in periods of inactivity during the route. For this schedule, during substantial breaks between services, the driver must be available to carry out ad-hoc duties which would need to take place at the depot. Having investigated this further, 3 occurrences have been identified where a significant gap in the schedule would allow a return to the depot and sufficient time to carry out such tasks. These are inserted into the schedule as 'dummy' requests with a service time equal to 45 minutes. These are the only breaks to be taken by this vehicle as, for a half-day shift, the driver does not take a scheduled break. There are in total 15 unique sites to be considered for this route, these include

those of the requests and the depot for the scheduled breaks.

Schedule S6 is similar to S2 with a shift duration of 8:30am to 5:00pm, Monday to Friday. This route transports a variety of items between all hospitals and major locations. There are in total 26 requests serviced by this schedule, with a 1 hour break period taken at the depot. This is inserted again as a 'dummy' request along with a request for the duration of the shift. There are 32 unique sites to be considered for this route.

Schedule S7 is a half-day shift visiting 28 GP surgeries, clinics, health centres and medical practices collecting mail and specimens to be delivered to a single hospital and the depot. The schedule starts at 11:00am and finishes at 15:30pm, Monday to Friday. There is again no break period in this route and there are 31 unique sites to be compared. Summary information for each schedule is available in Table 8.2.

Schedule	Duration	Requests	Type
S2	08:30 - 17:00	37	Transport mail, specimens, nurse bank and finance boxes in a repeated route around priority locations
S5	08:30 - 13:30	7	Transport specimens and drug boxes in a repeated route to 5 of the major hospitals
S6	08:30 - 17:00	27	Transport mail, specimens, nurse bank and finance boxes to and from all hospitals and major surgeries
S7	11:00 - 15:30	29	Numerous pickups from GP surgeries, clinics, health centres and medical centres all for delivery to a hospital and the depot

Table 8.2: Summary Information for 4 HCS Fixed Schedules

To validate the travel times and travel distances obtained, the requests are serviced in their existing order. Therefore no time windows need to be assigned; a location is simply serviced as soon as it is feasible. Preliminary investigations show that allocating a service time of 4 minutes to each 'different' service at a location provided acceptable results.

The difference between the arrival times at each unique site are now compared to those found in the existing schedules. Figure 8.3 provides the results for schedule S2, Figure 8.4 provides the results for schedule S5 and Figures 8.5 and 8.6, provide results for schedules S6 and S7 respectively. A positive difference identifies that our arrival time is later than in the existing schedule and a negative difference identifies that our arrival time is earlier than in the existing schedule.

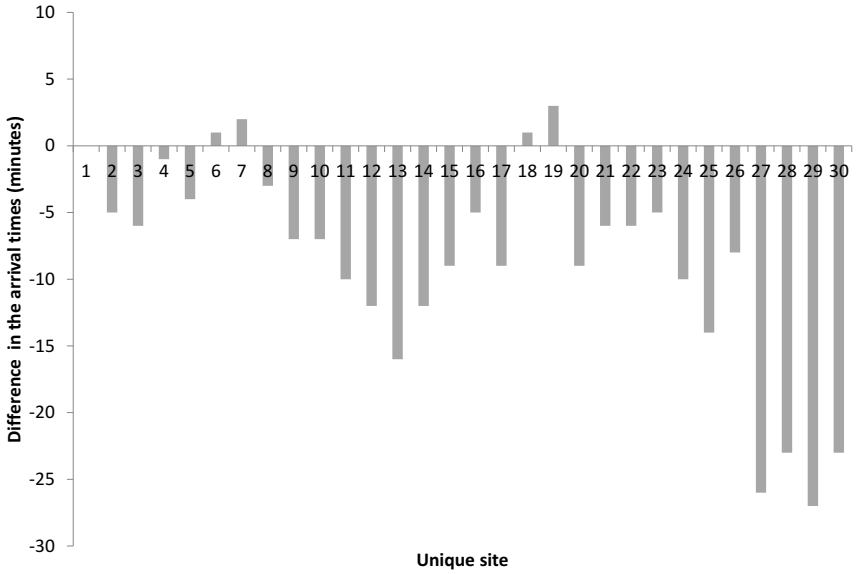


Figure 8.3: Difference between the arrival times - S2

Figure 8.3 shows that the results provided for schedule S2 vary throughout the day. During the beginning of the schedule the arrival times are less than 6 minutes earlier than the existing times, reaching a maximum of 16 minutes early at the 13th site visited, before returning to virtually no difference around the 18th site. However, then the arrival times produced become earlier than those in the existing schedule, once again, reaching a maximum of 27 minutes early at site 29. It is noted that the largest difference in times occur during the middle and end of the route. This could be due to underestimating the service time and variations in travel times.

It could be that, during the morning rush hour, the travel times between sites are greater than those estimated; however, these are compensated by underestimating the service time required. For the later periods of the day, a lower service time is then not compensated as there is no longer a significant increase in travel time.

For the results of schedule S5, there is less variation in the arrival times than in S2 this could be due to the smaller number of requests that are serviced. Except for one major clinic, this route only services hospitals, and could show that the service time applied is more consistent for servicing hospitals than for the other sites. The schedule reaches a maximum of 9 minutes late compared to the existing schedule, but quickly returns to what is closely observed in the existing schedule.

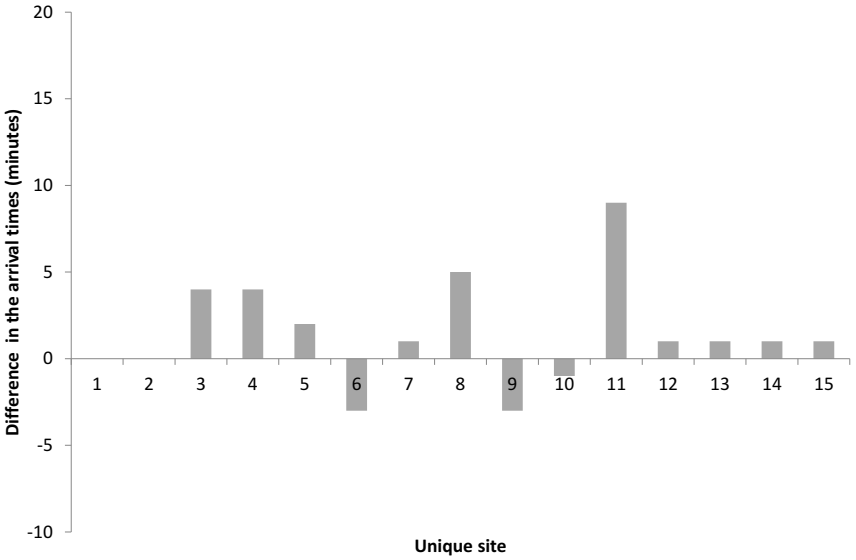


Figure 8.4: Difference between the arrival times - S5

It is clear from Figure 8.5 that the travel times and service times applied to schedule S6 underestimate the time needed to service the sites towards the second half of the route, similar to S2. The arrival times reach a maximum of 49 minutes earlier than the existing schedules. There are once again differences in the arrival times throughout the day - the schedule is relatively accurate for the first part of the route and the difference increases towards the end of the day.

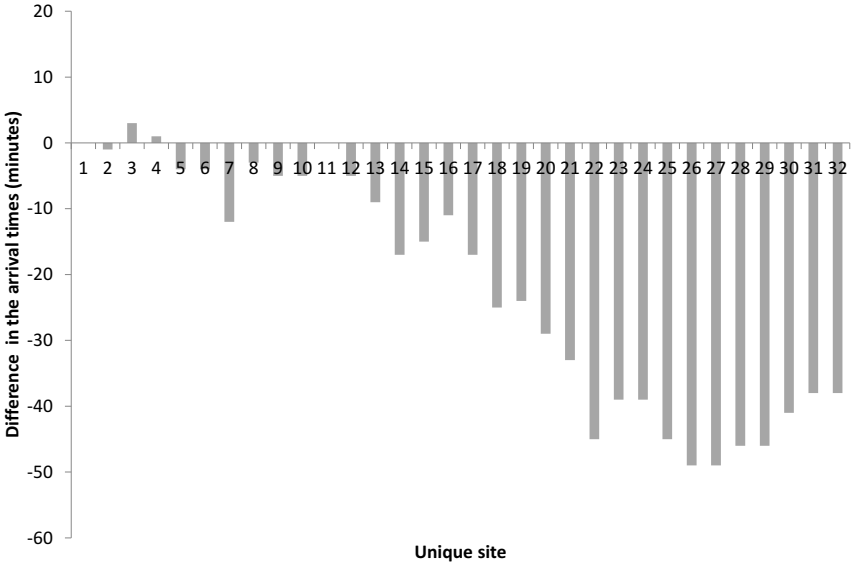


Figure 8.5: Difference between the arrival times - S6

Finally, looking at the arrival times for schedule S7, it is clear that these are different

to those of schedules S2 and S6, as they are generally later than what is expected. This could be accounted for by the fact that this is only a half-day shift taking place during the middle section of the day when travel times may not be so varied. It could be that, servicing smaller sites such as GP surgeries, does not require as long a service time as servicing a hospital. This could be accounted for by the fact that, when servicing a hospital a driver might have to visit a specific department, which would add an additional travel time once at the site.

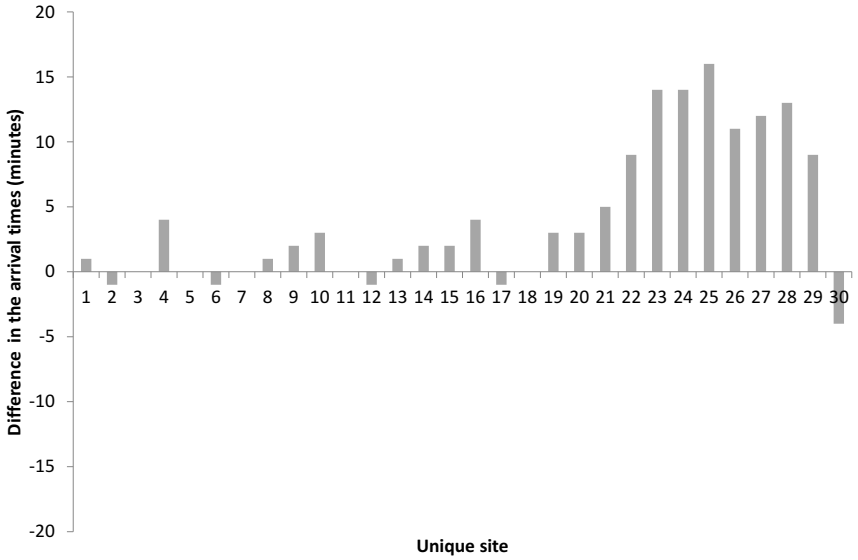


Figure 8.6: Difference between the arrival times - S7

It can be seen that there are many difficulties in estimating the travel times and service times accurately to reproduce the schedules currently followed by the service. It is thought that varying travel times during the day can have an effect on the schedules produced and that service times vary depending on the site to be serviced and the number of items to be serviced at each site. It appears that servicing a GP surgery requires less time than servicing a hospital for instance. It is also thought that the service may factor slack time into its existing schedules, to allow for any changes that may occur such as a road closure or a vehicle breakdown. This could account for the differences in the schedules.

The results for the total travel times (minutes), travel distance (kilometres) and available time (minutes) for each of the schedules are provided in Table 8.3. The available time is calculated based on the difference between the duration of the schedule and the total of the travel times and service times (including break periods). This should allow us to better understand where the dynamic requests can be inserted into these schedules.

Schedule	Duration	Travel time	Distance	Service time	Free time
S2	510	207	136.18	284	19
S5	300	124	88.05	175	1
S6	510	225	138.64	240	45
S7	270	116	69.83	150	4

Table 8.3: Initial Results for the HCS Fixed Schedules

The information provided in Table 8.3 shows that there are definitely differences in the time available in each of the schedules. In particular, there is 37 minutes of available time in the schedule for S6, it is not clear if this time could be utilised by the 24/7 service.

We now investigate whether any improvement can be achieved by re-ordering the locations in the existing routes. In this case, the width of the time windows assigned to each request needs to be determined, since it is clear that some requests are more flexible than others. Initially all requests are assigned a time window of 60 minutes, i.e. service may begin 30 minutes before or after the actual time from the existing schedules. The *slack* insertion heuristic defined in Section 3.4 will be utilised to insert the requests into the route and the branch and bound heuristic defined in Section 4.5 will attempt to improve the ordering of the locations. This has been chosen, as we are considering the case of a single vehicle for each schedule.

Generally, multiple services that take place at the hospitals are fixed (it is known that these need to be visited multiple times throughout the day). Based on this information, there is therefore little improvement to be made with schedules S2, S5 and S6. For schedule S5, a slight improvement is achieved without altering the grouping of services at sites. The total travel time is reduced by 1 minute and the total travel distance by 1.41km, not a significant improvement. No improvement is made to S2 and S6, however, there are already 3 'dummy' break periods in S2, indicating 45 minutes spent at the depot, which could be utilised by the 24/7 service.

Schedule S7 consists of visiting a set of locations which all have two general delivery sites - being a single hospital and the depot. Therefore, these locations can be serviced in any order, provided all pickup locations are visited before making a delivery. This could be controlled via time windows, by setting the opening time window at the delivery sites, after the closing time of all pickup locations. Applying this to S7, our algorithm provides a solution with a distance of 58.78km, a reduction of 11.05km and a total time of 92 minutes, a saving of 24 minutes (over 20%) of total time travelled.

This results in 28 minutes of spare time available, where the driver could complete ad-hoc tasks. Table 8.4 summarises the improvements.

Schedule	Duration	Travel time	Distance	Service time	Free time
S2	510	206	134.77	284	20
S5	300	124	88.05	175	1
S6	510	225	138.64	240	45
S7	270	92	58.78	150	28

Table 8.4: Summary Results for the HCS Fixed Schedules after Improvement

Table 8.4 shows improvements can be made to the routes where services are not grouped together at sites (S7). The available time created in these routes could then allow the driver to return to the depot to complete ad-hoc tasks. This could result in using the time currently available when servicing the major hospitals (S2), to service the dynamic requests. This would be beneficial as it is known the requests arriving to the 24/7, are priority requests whose pickup location, delivery location or both are generally a major hospital. Further work is needed to better predict the service times for the requests and to improve estimates of the travel distances and travel times during different periods of the day.

This initial analysis shows that the courier routes currently carried out by the service could be utilised to incorporate dynamic requests. The dynamic requests received by the 24/7 service will be investigated in the next section.

8.7 Investigating the Dynamic 24/7 Service

The service is interested in expanding its capabilities with regards to its 24/7 service. In the future it would like to offer the service to other neighbouring Health Boards. It is therefore interested in analysing the current demand it receives, to investigate the potential for expansion.

The data to be investigated consists of 6 days of records for the period 01/02/2011 to 06/02/2011, labelled D1 - D6. This was a Tuesday to a Sunday, capturing both weekday and weekend demand. Initial analysis is performed to identify if all the requests, arriving on each of these 6 days, can be serviced by one route (the 24/7 vehicle). To do this, time windows need to be assigned to each request. It is known that the highest priority request is for blood and this has a maximum 2 hour time window. Therefore,

each request is initially assigned a maximum time window of 2 hours. Service can begin at the pickup location immediately after the request is received.

Summary information for each of the 6 days is provided in Table 8.5. 'Requests', is the number of requests received and 'Arrival' provides the arrival rate per hour. 'Time', represents the total travel time and 'Distance', the total travel distance. As previously for the static schedules, a service time of 4 minutes is assigned to each location and the total service time is represented by 'Service'. Finally, '% Available' represents the percentage of available time in the schedule.

Based on these 6 days of data obtained, the 24/7 service is able to feasibly service all requests using its 24/7 vehicle. However, this is only a small sample of data and variation between the numbers of requests arriving to the service is thought to be high, hence this could not be representative of the service as a whole. Looking at the number of requests, an average of 17 requests are received per day (for the period of data considered) giving an average arrival rate of 0.69 per hour. It can be seen that there is a lower demand on weekends, with an average of 14.5 requests received, compared to an average of 17.75 on a weekday. On average, the percentage of free time available is 64%, showing the vehicle is being utilised only 36% of the time.

	Requests	Arrival	Time	Distance	Service	% Available
D1	21	0.88	456	344.57	168	57%
D2	12	0.50	293	226.98	96	73%
D3	20	0.83	418	317.75	160	60%
D4	18	0.75	452	352.31	156	58%
D5	14	0.58	332	247.38	112	69%
D6	15	0.63	384	285.02	120	65%
Avg	16.67	0.69	389	295.67	135.33	64%

Table 8.5: Initial Results for the HCS 24/7 Service

To better understand the idle times of the vehicle, Figure 8.7 shows the periods of inactivity of the vehicle during the D1 shift, carried out on a weekday. Figure 8.8 shows the periods of inactivity during the D5 shift, a day on the weekend.

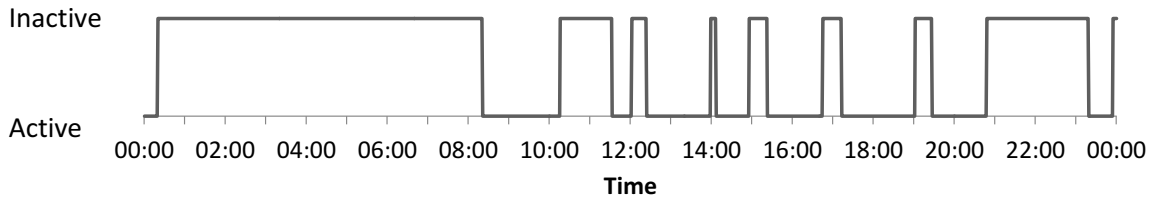


Figure 8.7: Waiting periods of the 24/7 vehicle on a weekday

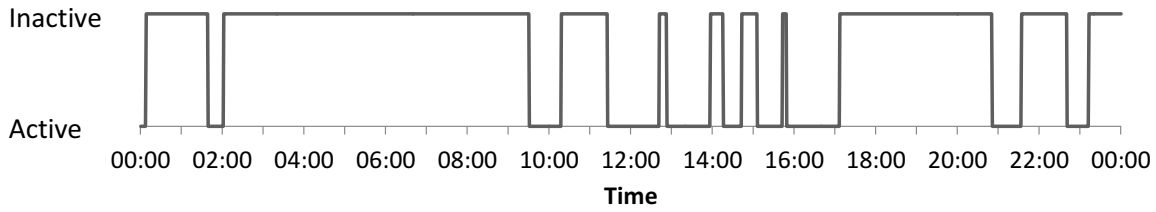


Figure 8.8: Waiting periods for the 24/7 vehicle for a day on the weekend

From both figures it appears that the vehicle waits for a long period during the early hours of the day, before a request is received. This is what would be expected because only emergency procedures are carried out during this period. There are longer periods of waiting during the latter half of the shift, again consistent with what would be expected. Comparing both figures, we observe longer periods of idle time in Figure 8.8, due to the fewer number of requests that arrive on average.

From the results provided, it would appear that the service is able to increase its capacity and potentially service a higher proportion of requests. To investigate this further, a larger set of requests arriving in a single 24 hour period needs to be considered. At present, the requests arrive randomly to the service and it is known that there is no correlation between requests arriving on each day. To produce a larger set of requests to investigate, the requests from each individual day will be combined.

Table 8.6 provides summary results. Here the '% serviced' indicates the percentage of requests which can be serviced by the 24/7 vehicle (all other variables are as in Table 8.5). Combinations of days have been considered to provide the best range for the number of requests. The algorithm in this case applies the dynamic simple *slack* insertion heuristic, as outlined in Section 6.4, along with the branch and bound heuristic adapted to a dynamic environment to attempt to improve the results.

	Requests	Arrival	Time	Distance	Service	% Available	% Serviced
D5+D6	29	1.21	541	414.43	232	46%	100%
D1+D2	33	1.38	606	449.40	264	40%	100%
D3+D5	34	1.42	578	416.64	264	42%	97%
D3+D4	38	1.58	702	538.17	292	31%	92%
D4+D5+D6	47	1.96	744	589.57	364	23%	94%
D1+D2+D3	52	2.17	645	465.02	336	32%	81%

Table 8.6: Summary Results for the HCS 24/7 Service under High Demand

From the results provided in Table 8.5, it can be seen that combining the requests for days D5 and D6, and for D1 and D2, giving 29 and 33 requests respectively, produces a schedule that can still be feasibly serviced by the 24/7 vehicle alone. For the case of combining the schedules, D3 and D5, and for D3 and D4, giving 34 and 38 requests respectively, results in no longer feasibly servicing all requests in a single route. The percentage of requests serviced decreases as the number of requests continues to increase in the following cases. However, the available time does not decrease any less than 23%, even though requests are not being feasibly serviced. It could still be possible to service the requests not feasibly serviced by the 24/7 vehicle using the current static schedules as discussed in Section 8.6.

To investigate the idle time of the vehicle once more, the case of combining the requests of days D1, D2 and D3, where it appears that 32% of the time the vehicle is idle, will be considered. Figure 8.9 shows the periods of inactivity of the vehicle during the schedule produced. We see that there is a significant reduction in the total number of periods that the vehicle is idle in Figure 8.9 compared to the 2 previous cases considered. The vehicle is now only waiting during the early hours of the morning, or late at night. During the periods of high demand, the vehicle is not idle (this is the period where requests are refused by the service).

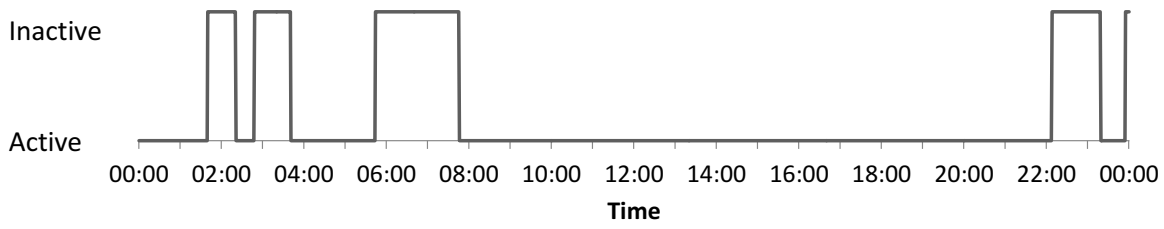


Figure 8.9: Waiting periods for the 24/7 vehicle under high demand

One way to improve the schedules produced for the 24/7 service would be to improve the allocation of time windows to requests. It is known that some of the requests received, during the periods investigated, were not of a high priority, hence these time windows could be widened. These could be scheduled during the current idle times of the vehicle; this could result in servicing a higher proportion of requests.

It can be seen by the results, that there is potential to both improve the scheduling of requests and also to increase the capacity of the service. It appears that the service can currently service approximately 33 requests feasibly using the 24/7 vehicle alone, although this would vary depending on the requests received.

8.8 Chapter Summary

This chapter has applied the research undertaken in this thesis to a real-life variant of both the static and dynamic PDPTW. A review of the current literature surrounding this real-life variant of the problem, highlighted a lack of research surrounding such problems, hinting that further research in this area would be beneficial. However, it has been made clear that the real-life problem brings with it added constraints and continued research is needed to better incorporate such information into our current algorithms.

The travel times and service times currently applied do not account for any variation experienced in a real-life situation, this is something that could be investigated further. There was also limited improvement to be made in improving the fixed schedules currently carried out by the service. However, analysis has proved useful in better understanding the opportunities to incorporate the dynamic requests and has highlighted which routes have the current capacity available for this.

For the requests arriving to the 24/7 service, for the days investigated, the service is able to cope with the demand it receives. It has been shown that if this was to increase on average by $\approx 50\%$ then this would no longer be the case. It has also be shown that the 24/7 vehicle is idle for large portions of the time, generally during the night. With better planning and scheduling of the dynamic requests, there appears to be potential for the service to increase its capacity and provide a similar service to other Health Boards in Wales.

It is clear that further work is needed in this area and would provide useful research. Ideas for future research are provided in Section 9.5. This thesis is concluded in the following chapter.

Chapter 9

Conclusions and Future Research

9.1 Introduction

This research has investigated both the PDPTW and the DPDPTW and has developed effective heuristic and metaheuristic approaches to solve these. It has also introduced a real-world application for both problem types with regards to a Health Courier Service (HCS). This chapter will present a summary of the main conclusions that can be drawn from this research, highlighting the contributions made to the field and will then identify areas for future work.

Section 9.2 will summarise the main conclusions that can be drawn from each chapter presented in this thesis. Section 9.3 provides a discussion on methods to further support the promising results achieved by our algorithm for the PDPTW by investigating larger instances of Li and Lim [2001]. Section 9.4 provides an insight into a future research area in applying the methods introduced in this thesis for the DPDPTW to large scale PDPTW. Section 9.5 further discusses the work that could be carried out for the HCS in producing a system that could be applied to assist with the scheduling of real-time requests. Finally Section 9.6 provides some closing remarks.

9.2 Conclusions

We now summarise the main conclusions that can be drawn from this work:

Chapter 1 highlighted the increasing need for algorithms able to perform in a real-time manner and also noted the minimal research into the DPDPTW. This provided the motive for this thesis.

Chapter 2 argued that heuristic and metaheuristic approaches are appropriate for this research as they produce competitive results quickly. This is crucial when looking to adapt the problem to the dynamic variant.

Chapter 3 provided a fast and effective method for the PDPTW that can be adapted to a real-time setting for use within a dynamic environment. Varying insertion heuristics adapted from methods in the literature were compared and two neighbourhood operators previously studied by Li and Lim [2001] were modified in accordance with the aim of minimising computational time. To further improve on the results, 4 reconstruction heuristics were introduced in Section 3.9 developed both from those in the literature and new approaches. In total, 26 of the best found solutions are achieved, and on average the results were $\approx$2% above the best known solutions. This has shown that the heuristic methods investigated are capable of finding high quality solutions in a reasonable amount of computational time.

Chapter 4 investigated both a tabu search heuristic and a branch and bound heuristic to improve the solutions for the PDPTW. For the tabu search heuristic a new tabu attribute and criteria for the search have been determined. The branch and bound heuristic was developed specifically to optimise sections or sub-sections of routes. The results achieved by combining these methods have been shown to be competitive with the state of the art results found in the literature: the algorithm obtained the best known solutions in 51 out of a possible 56 instances and performed consistently well over all types of instance.

One of the main advantages of our approach is the speed of individual constructions. In this case it has allowed us to produce large samples of solutions in times that are consistent with other approaches. This advantage is exploited when applying these methods to the dynamic variant of the problem.

Chapter 5 introduced the research into the DVRP and its variants. An overview of the instances available for the DVRP has been summarised, including for the first time, an evaluation of those instances available for the DPDPTW.

Chapter 6 investigated ways to adapt the algorithm introduced in Section 4.7 to the DPDPTW. New dynamic insertion heuristics that vary in the criteria they adopt for the insertion, are explored. These update the solution during the scheduling horizon for the requests that arrive dynamically and are analysed for both varying degrees of urgency and proportions of dynamic requests. It is shown that a simple slack insertion method when paired with the tabu search heuristic and the branch and bound heuristic provides the best results.

The initial investigations performed for the varying insertion methods and improvement heuristics provide insights into the characteristics of the problem which have not before been explored, these include which is the best method of insertion and what should be inserted at the varying degrees of urgency and proportions of dynamic requests.

It is seen that our algorithm produces improved results under circumstances of a high degree of urgency. The greatest improvements are achieved with the P2 set of instances (compared to Pankratz [2005b]). The instances in the set P2 contain varying proportions of dynamic requests that arrive with the highest degree of urgency, showing our algorithm performs well under these conditions.

Chapter 7 showed our algorithm continued to perform consistently well over a different set of instances with a larger number of requests. For the first time the improvements made to a solution during the scheduling horizon are considered and comparisons are made with the results of the static algorithm.

Chapter 8 applied the research undertaken in this thesis to a real-life variant of both the static and dynamic PDPTW. Investigations proved useful for better understanding the opportunities to incorporate the dynamic requests arriving to the service. With better planning and scheduling of the dynamic requests, there appears to be potential for the service to increase its capacity and provide a similar service to other Local Health Boards. It is clear that further work is needed in this area and would provide useful future research.

As we have seen, in most of this thesis we have chosen to base our studies on standard benchmark-instances. This, we believe, has been useful as it has provided a means by which we are able to compare our results against others in a meaningful way. However, it is worth bearing in mind that whilst the use of these sorts of instances may facilitate the analysis and comparison of algorithms, they do not necessarily allow insight into how these algorithms might fare with other kinds of problem instances. It is also worth remembering that while the requirements of this particular PDPTW and DPDPTW seem to show our methods provide promising results, in problems where the requirements are different to these; other algorithms might prove to be more suitable in some cases.

In conclusion, when designing algorithms for the PDPTW, it is always worth considering, as shown in Chapter 8, that in the real-world many different sorts of constraints, problem instances, user-requirements and political factors might be encountered. The idiosyncratic nature of real-world PDPTW seems to indicate an advantage to those algorithms that are robust with respect to changes in instance types which is what we have shown in Chapters 6 and 7.

The next section will consider ways in which the research in this thesis may be developed further - both with regards to extending the instances used to evaluate our algorithms, and for standard and real-world instances.

9.3 Further Work for the PDPTW

One way to further the research conducted in Chapters 3 and 4, for the PDPTW would be to examine how the algorithm performs with larger problem instances. The research into the PDPTW in this thesis has been carried out on the so called '100 customer' problems of Li and Lim [2001], which have between 100-106 locations (50-55 requests). There also exists instances of Li and Lim [2001] with 200, 400, 600, 800 and 1000 'customers'.

TOP is the Transportation Optimization Portal of SINTEF Applied Mathematics SINTEF [2004]. Here, the instance definitions and best known solutions, for the 100, 200, 400, 600, 800, and 1000 'customer' instances of Li and Lim [2001] PDPTW problems are found. The results reported are the best known solutions for a hierarchical objective: 1) Minimize number of vehicles 2) Minimize total travel distance. Hence, results are not directly comparable to our own. Best known solutions from the literature include those of Li et al. [2001], Bent and Van Hentenryck [2006], Ropke and Pisinger [2006a], Hasle and Kloster [2007] and Koning et al. [2011], also, those from a commercial heuristic developed by a Danish tool vendor, whose method is unknown, TetraSoft A/S are featured.

These have not been investigated in this research, as the best known solutions for these instances are achieved for an objective function which first minimises the number of vehicles required. Preliminary results show that for the larger instances, unlike the '100 customer' problems, the varying objectives provide very different solutions (in terms of the number of vehicles, etc.). Therefore it would be difficult to evaluate our algorithms successfully.

It may however be useful to extend our approach to produce a new set of 'best known' solutions where the objective is first to minimise the number of vehicles. Table 5 in Appendix C provides preliminary results for the 200 location instances of Li and Lim [2001]. It would also be beneficial to validate the computational times of our algorithm when applied to larger instances.

Using our objective of minimising the total travel distance, only 19 out of the 60 best known solutions, when the objective is first to minimise the number of vehicles, are

achieved for the case of 200 requests. New best found solutions, in terms of minimising the total travel distance, are achieved in 31 instances. However, these increase the number of vehicles required by 68. For 10 instances, the total travel distance is not improved and the number of vehicles required is increased by 12. This highlights further improvement could still be achieved for these instances. Overall our algorithm reduced the total travel distance by $\approx 5\%$ and increased the number of vehicles required by $\approx 13\%$, quantifying this difference is however subjective.

As this research has aimed to adapt our algorithm to the DPDPTW, this is where our research has concentrated. For the instances of Li and Lim [2001] only the '100 customer' problems have been adapted and analysed for the dynamic problem and hence these are the instances we have chosen to investigate. It would not be appropriate to choose the objective of first minimising the number of vehicles required in a dynamic environment as this would result in assigning the dynamic requests to the minimal number of vehicles early on in the scheduling horizon. This would likely result in having to introduce a higher number of vehicles later on in the scheduling horizon. The best known solutions for the instances provided in the literature are generated with an objective of minimising the total travel distance. Hence, to achieve a meaningful comparison, this objective is chosen for our research.

Again further research could look to adapt the instances of Li and Lim [2001] with a larger number of requests to a dynamic environment by adding a time stamp to each request as in Pankratz [2005b], this would then create a larger basis for comparison.

9.4 Further Work for the DPDPTW

Another idea for furthering the work in this thesis is to adapt the DPDPTW to solve larger instances of the PDPTW, such as the instances of Li and Lim [2001] with 1000 'customers'. It is thought that applying a rolling horizon framework, to solve larger instances of the PDPTW, may be useful in providing reasonable quality solutions in a short amount of time.

Here, a request could be assigned a 'dummy' time stamp according to its urgency and the requests could then be inserted at given intervals. It was shown in Chapters 6 and 7 that an insertion criterion based on inserting the most urgent requests first achieved the most promising results. Requests could then be inserted, whereby at each interval, a new subset of requests is available for insertion.

Whilst investigating the larger instances of Mitrovic-Minic et al. [2004] in Chapter 7

it was found that achieving a solution to the dynamic problem required a significantly reduced amount of computational time than that of its static counterpart. For the instances with 500 requests, on average the computational time required was reduced from 3 hours to only 2 minutes. The results achieved were on average 20% greater than that achieved by the static algorithm, with regards to the total travel distance. Therefore it would seem that a reasonable solution could be achieved for a large scale static problem by applying the methods adapted from those applied to solve the dynamic problem. This solution could be achieved in a significantly reduced amount of computational time, showing that with further work this could be an effective method to solve larger instances of the PDPTW.

Another area of research surrounding DPDPTW is the development of waiting strategies (see Section 5.4.4). The presence of time windows in this variant of the problem means that vehicles may have to wait at various locations along their routes. The solution quality may therefore be affected by the way the waiting time is distributed along vehicle routes. Mitrovic-Minic and Laporte [2004] investigate 4 waiting strategies for the DPDPTW as described in Section 5.4.4.

The first strategy considered by Mitrovic-Minic and Laporte [2004] is the drive-first (DF) strategy; this requires a vehicle to drive as soon as it finishes service at its current location. This is the most common strategy used in the literature and is the only appropriate strategy to use within the static problem, therefore it has been applied in this research for the PDPTW.

It can generally be seen in the literature that achieving a lower average waiting time with each route also results in a lower total travel distance. Therefore, there could be scope to investigate whether improving the distribution of waiting times during a route could further improve the total distance travelled of the solutions achieved.

There are also many other variants of the DPDPTW which could be considered, namely those which incorporate time-dependent travel times or stochastic demands seem to have generated interest in the last few years and provide exciting avenues for research.

9.5 Further Work for the HCS

Finally, considering the real-world variant of the problem studied in Chapter 8, it is clear that more work is needed in determining travel times for the model that take into account the changes in congestion during peak travel times. It is also clear that a method needs to be created to assign an appropriate service time for each request. These include what is being picked up or delivered, how many services there are at that location, whether there are multiple services of the same type, and finally, what type of location is being serviced. To produce accurate criteria for assigning a reliable service time, further investigations would be needed on a larger number of schedules. This could help the HCS to increase its capacity in the number of dynamic requests it can feasibly service.

A similar service is also provided by WAST alongside its HCS, this is the Patient Care Services. This service is responsible for transporting a wide range of patients including disablement service centre patients, out-patients, routine discharges and admissions and non-urgent inter-hospital transfers. As for the problem encountered by the HCS, this can also be adapted to a DPDPTW and again there exists limited literature available into this area.

9.6 Final Remarks

In this final chapter of the thesis we have highlighted the most significant findings and the most promising areas of our research. Research areas where further investigation are still needed have also been identified. Further publications of the findings of this thesis are also underway.

Appendices

A Methods to Randomise the Initial Insertion Heuristics

In Chapter 3 a new *random* insertion heuristic is proposed that adds randomisation to the *greedy* insertion heuristic of Nanry and Barnes [2000]. It is found that the *random* method produces promising results in comparison to other insertion heuristics from the literature including the *greedy* method of Nanry and Barnes [2000], the *max_dist* method of Li and Lim [2001], the *slack* method of Pankratz [2005a] and the *accept_first* method of Hosny and Mumford [2009b]. It is therefore investigated whether adding randomisation to any of the other methods, would also achieve such promising results. For further information on each of these insertion methods see Section 2.8.1.

To add randomisation to the initial solutions, first a greedy randomised adaptive search procedure (*grasp*) is introduced (see Pitsoulis and Resende [2002]). The *greedy* method of Nanry and Barnes [2000] constructs a solution by, at each iteration, inserting the request from all remaining requests, that evokes the lowest additional cost to the objective function.

For the case of the *grasp*, after all the feasible insertion positions (also known as candidate elements), of the request have been examined, they are ranked. Well-ranked candidate elements are placed in a restricted candidate list (RCL) and an element from the RCL is selected at random with a given probability and added to the solution. In the cardinality-based scheme which we use, an integer k is fixed and the k top ranked candidates are placed in the RCL (for an example of a grasp within a VRPTW, see Kontoravdis and Bard [1995]). In our case as a method of improving the initial solutions we add the *grasp* to both the *greedy* insertion heuristic and the *max_dist* insertion heuristic outlined in Section 3.4. The number of ranked elements, k, is set to 3 as this was shown to provide the most promising results in preliminary investigations.

In the case of the *greedy* heuristic, all feasible insertions will now be ranked and the 'best' 3 insertions will be stored. Each of these insertions will then be assigned a probability, based on the increase in cost of that insertion to the solution, and a random number will decide on which insertion to accept.

For the *max_dist* method, a route will still first be initialised with a request, using criterion based on maximum combined distance from depot. Then, rather than greedily choosing the request to insert next into the route, the process is carried out as above. This is performed for each route until no further requests can feasibly be inserted and the process of initialising a route starts again.

For the case of the *max_dist* insertion method, the routes of the initial solutions are generated sequentially and the criterion of maximum combined distance is used to initialise a route. To add randomisation to this method, another insertion heuristic to be investigated will be the *rand_initialise* method. This will generate routes sequentially as above but the first request to be inserted into a route will now be chosen randomly from all those remaining.

The final attempt to add randomisation to the initial insertion heuristics comes with the *accept_first* method, currently this first orders all requests in order of their slack time, i.e. the time available to feasibly service the request. To add randomisation to this method of insertion, the request is now chosen at random to be inserted and then again inserted into the first feasible position identified as before.

Table 1 provides results for each of these randomised insertion heuristics and the *random* insertion method introduced in Section 3.4. Results are best solution found after 100 runs and are provided for each set of instance.

	Random	Grasp	Max_dist_grasp	Rand_initialise	Rand_acc_first
LC1	12 564.81	8127.37	9703.51	7973.19	22 209.31
LR1	19 459.61	18 371.54	18 331.94	17 197.33	23 838.37
LRC1	15 392.08	14 314.49	13 441.57	13 161.94	18 329.46
LC2	5468.77	5981.22	5922.10	5186.02	21 335.69
LR2	15 003.45	15 049.54	15 801.91	15 120.63	27 936.97
LRC2	14 087.54	12 820.43	13 894.66	12 912.25	25 112.96
Total	81 976.26	74 664.60	77 095.69	71 551.36	138 762.76

Table 1: TD achieved by each of the Randomised Insertion Heuristics for each set of instances

It can be seen from Table 1 that the *rand_initialise* insertion method achieves the overall lowest total travel distance for the LC1, LR1, LRC1 and LRC2 instances. For the LR2 set of instances it is the *random* method that achieves the lowest total travel distance and for the LRC2 set, it is the *grasp* insertion method. Table 2 provides the computational times of applying each of these methods by each set of instances. The result is the average time to complete 100 runs on each instance in the set.

	Random	Grasp	Max_dist_grasp	Rand_initialise	Rand_acc_first
LC1	2.41	41.92	6.34	6.68	2.06
LR1	2.52	42.85	5.59	6.11	2.04
LRC1	1.50	25.14	3.27	3.60	1.29
LC2	7.70	165.59	52.52	58.65	8.77
LR2	14.14	274.97	133.49	138.01	9.47
LRC2	7.19	143.49	64.62	62.87	4.89

Table 2: Average CT required be each of the Randomised Insertion Heuristics for each set of instances (seconds)

It can be seen from Table 2 that there is an increase in computational time when applying the *grasp*, the *max_dist_grasp* and the *rand_initialise* insertion methods compared with the *random* insertion. The only method that achieves comparable times is the *rand_acc_first* method. It is felt that the lower costs achieved for two sets of instances by the *rand_initialise* method does not warrant the significant increase in computational time for the instances with a longer scheduling horizon that this method generates.

From the results in Section 3.8 it was found that, after applying the neighbourhood operators to the solutions achieved by the initial insertion methods, the best overall results were not achieved by the insertion method that had provided the initial lowest total distance travelled. Therefore, for further clarification, the *shift* and *exchange* neighbourhood operators (see Section 3.6) were added to the *rand_acc_first* method outlined above to see if this method could improve on the results of the *random* insertion method. This method was the only method of those examined which provided reasonable times for the insertion phase.

From Table 3 it can be seen that the *random* insertion method again achieves a lower total travel distance (TD) than the *rand_acc_first* method on each set of instances. The computational times (CT) provided are the average time taken to complete 100 runs on each instance and the total is the average time taken to complete 100 runs for all of the 56 instances. The increase in computational time for the *rand_acc_first* method further establishes that the *random* insertion method should be chosen for use within our research.

	Random		Rand_acc_first	
	TD	CT	TD	CT
LC1	7799.41	2.95	7821.51	3.44
LR1	15 609.71	2.91	15 682.62	2.92
LRC1	11 999.79	2.32	12 156.05	2.36
LC2	5171.77	22.07	6089.14	25.09
LR2	12 578.79	88.50	12 856.46	79.71
LRC2	10 903.35	27.71	11 244.27	28.36
Total	64 062.82	25.92	65 850.05	24.81

Table 3: TD achieved by the Randomised Insertion Heuristics and Neighbourhood Operators for each set of instances

B Results for Mitrovic-Minic et al. [2004] Instances with 1000 Requests

Instance	Our algorithm
Rnd8_10h_100_000	13909.7
Rnd8_10h_100_001	14545.7
Rnd8_10h_100_002	13997.1
Rnd8_10h_100_003	14606.1
Rnd8_10h_100_004	14257.4
Rnd8_10h_100_005	14312.6
Rnd8_10h_100_006	13754.1
Rnd8_10h_100_007	14202.8
Rnd8_10h_100_008	14003.1
Rnd8_10h_100_009	14408
Rnd8_10h_100_010	14222.8
Rnd8_10h_100_011	14496.5
Rnd8_10h_100_012	14324.2
Rnd8_10h_100_013	14079.1
Rnd8_10h_100_014	13923.8
Rnd8_10h_100_015	14463.9
Rnd8_10h_100_016	14398.4
Rnd8_10h_100_017	14626
Rnd8_10h_100_018	14288.4
Rnd8_10h_100_019	13366.2
Rnd8_10h_100_020	14111.5
Rnd8_10h_100_021	14140.1
Rnd8_10h_100_022	14010.2
Rnd8_10h_100_023	14077.9
Rnd8_10h_100_024	14203.8
Rnd8_10h_100_025	13709.2
Rnd8_10h_100_026	14080.9
Rnd8_10h_100_027	13907.8
Rnd8_10h_100_028	14655.2
Rnd8_10h_100_029	14562.1
Average	14188.15
Mitrovic-Minic	17610.45
% Decrease	19%

Table 4: TD achieved by that of Our Algorithm for 1000 requests for the RND8 Instances of Mitrovic-Minic et al. [2004]

C Results for Li and Lim [2001] Instances - 200 Requests

Instance	TD	CT	NV	Diff TD	Diff NV	TD	NV	Best Known
LC1_2_1	2704.57	44	20	0	0	2704.57	20	Li and Lim [2001]
LC1_2_2	2764.56	539	19	0	0	2764.56	19	Li and Lim [2001]
LC1_2_3	2772.18	1605	18	−356.43	1	3128.61	17	Ropke and Pisinger [2006a]
LC1_2_4	2661.40	3910	18	−32.01	1	2693.41	17	Bent and Van Hentenryck [2006]
LC1_2_5	2702.05	71	20	0	0	2702.05	20	Li and Lim [2001]
LC1_2_6	2701.04	104	20	0	0	2701.04	20	Li and Lim [2001]
LC1_2_7	2701.04	130	20	0	0	2701.04	20	Li and Lim [2001]
LC1_2_8	2689.83	460	20	0	0	2689.83	20	Li and Lim [2001]
LC1_2_9	2724.24	867	18	0	0	2724.24	18	Li and Lim [2001]
LC1_2_10	2741.56	2105	18	−201.93	1	2943.49	17	Ropke and Pisinger [2006a]
LC1_2	27 162.47	9834	191	−590.37	3	27 752.84	188	
LC2_2_1	1931.44	441	6	0	0	1931.44	6	Li and Lim [2001]
LC2_2_2	1881.40	3493	6	0	0	1881.40	6	Li and Lim [2001]
LC2_2_3	1886.01	8427	7	41.68	1	1844.33	6	Hasle and Kloster [2007]
LC2_2_4	1861.89	18 644	7	94.77	1	1767.12	6	Li and Lim [2001]
LC2_2_5	1891.21	962	6	0	0	1891.21	6	Li and Lim [2001]
LC2_2_6	1857.78	1563	6	0	0	1857.78	6	Hasle and Kloster [2007]
LC2_2_7	1850.13	2057	6	0	0	1850.13	6	Hasle and Kloster [2007]
LC2_2_8	1824.34	3153	6	0	0	1824.34	6	Li and Lim [2001]
LC2_2_9	1854.21	4499	6	0	0	1854.21	6	Hasle and Kloster [2007]
LC2_2_10	1817.45	4627	6	0	0	1817.45	6	Li and Lim [2001]
LC2_2	18 655.86	47 865	62	136.45	2	18 519.41	60	
LR1_2_1	4819.12	66	20	0		4819.12	20	Li and Lim [2001]
LR1_2_2	4093.05	689	19	−528.16	2	4621.21	17	Ropke and Pisinger [2006a]
LR1_2_3	3488.33	2082	18	−124.31	3	3612.64	15	TetraSoft
LR1_2_4	2858.95	6154	13	−178.43	3	3037.38	10	Ropke and Pisinger [2006a]
LR1_2_5	4221.62	208	18	−538.56	2	4760.18	16	Bent and Van Hentenryck [2006]
LR1_2_6	3804.71	1340	18	−370.45	4	4175.16	14	Bent and Van Hentenryck [2006]
LR1_2_7	3142.41	2566	16	−408.20	4	3550.61	12	Ropke and Pisinger [2006a]
LR1_2_8	2702.77	6354	12	−81.76	3	2784.53	9	Ropke and Pisinger [2006a]
LR1_2_9	3953.47	366	18	−401.19	4	4354.66	14	Ropke and Pisinger [2006a]
LR1_2_10	3386.34	837	16	−327.82	5	3714.16	11	Ropke and Pisinger [2006a]
LR1_2	36 470.77	20 660	168	−2958.88	30	39 429.65	138	
LR2_2_1	4085.82	903	7	12.72	2	4073.10	5	Hasle and Kloster [2007]
LR2_2_2	3867.31	4426	7	71.31	3	3796.00	4	
LR2_2_3	3265.77	13 060	6	167.41	2	3098.36	4	Ropke and Pisinger [2006a]
LR2_2_4	2090.25	36 816	4	−395.89	1	2486.14	3	Ropke and Pisinger [2006a]
LR2_2_5	3444.99	2132	4	6.60	0	3438.39	4	Hasle and Kloster [2007]
LR2_2_6	3201.54	7275	4	0	0	3201.54	4	Li and Lim [2001]
LR2_2_7	2736.98	21 014	4	−398.07	1	3135.05	3	Ropke and Pisinger [2006a]
LR2_2_8	1851.65	92 188	4	−703.75	2	2555.40	2	Ropke and Pisinger [2006a]
LR2_2_9	3198.45	3609	4	−732.04	1	3930.49	3	Ropke and Pisinger [2006a]
LR2_2_10	2820.56	7946	4	−502.81	1	3323.37	3	
LR2_2	30 563.32	189 370	48	−2474.52	13	33 037.84	35	
LRC1_2_1	3606.06	176	19	0	0	3606.06	19	Hasle and Kloster [2007]
LRC1_2_2	3292.43	720	19	−380.76	4	3673.19	15	Bent and Van Hentenryck [2006]
LRC1_2_3	3159.2	2513	15	−2.55	2	3161.75	13	Bent and Van Hentenryck [2006]
LRC1_2_4	2615.2	7457	12	−16.62	2	2631.82	10	Ropke and Pisinger [2006a]
LRC1_2_5	3742.2	462	17	26.39	1	3715.81	16	Bent and Van Hentenryck [2006]
LRC1_2_6	3360.86	342	18	−7.80	1	3368.66	17	Hasle and Kloster [2007]
LRC1_2_7	3367.3	715	17	−301.09	3	3668.39	14	Ropke and Pisinger [2006a]
LRC1_2_8	3157.33	1284	15	−17.22	2	3174.55	13	Ropke and Pisinger [2006a]
LRC1_2_9	3107.03	1111	16	−119.69	3	3226.72	13	Ropke and Pisinger [2006a]
LRC1_2_10	2867.98	1863	14	−83.31	2	2951.29	12	Ropke and Pisinger [2006a]
LRC1_2	32 275.59	16 643	162	−902.65	20	33 178.24	142	
LRC2_2_1	2997.06	914	7	−608.34	1	3605.40	6	Ropke and Pisinger [2006a]
LRC2_2_2	2713.62	3869	7	−613.56	2	3327.18	5	Ropke and Pisinger [2006a]
LRC2_2_3	2677.35	10 188	7	−260.93	3	2938.28	4	Ropke and Pisinger [2006a]
LRC2_2_4	2237.06	27 185	5	−650.91	2	2887.97	3	Ropke and Pisinger [2006a]
LRC2_2_5	2863.12	2961	6	86.19	1	2776.93	5	Bent and Van Hentenryck [2006]
LRC2_2_6	2707.96	2703	5	0	0	2707.96	5	
LRC2_2_7	2541.38	4916	5	−508.65	1	3050.03	4	Bent and Van Hentenryck [2006]
LRC2_2_8	2525.67	7082	5	125.72	1	2399.95	4	Ropke and Pisinger [2006a]
LRC2_2_9	2243.78	10 396	4	35.29	0	2208.49	4	Ropke and Pisinger [2006a]
LRC2_2_10	2059.10	16 175	4	−491.46	1	2550.56	3	Ropke and Pisinger [2006a]
LRC2_2	25 566.10	86 388	55	−2886.65	12	28 452.75	43	
Total	170 694.10	370 761	686	−9676.62	80	180 370.70	606	

Table 5: TD, CT and NV achieved by MetaheuristicAlgorithm compare to the Best Known solutions in the literature for TD

D All Fixed Schedules of the HCS

ID	Purpose	Shift start	Shift end	Days	Total hours	Shift duration
A1	Stationery and Stores	07:30	15:30	Mon-Fri	37.5	7.5
A2	Podiatry	07:30	15:30	Mon-Fri	37.5	7.5
A3	Histopathology	07:30	17:30	Mon-Fri	45	9
A4	HSDU Specimen to outlying hospitals	Various			14.5	
A5	Unit 13 to hospitals	07:30	15:30	Mon-Fri	37.5	7.5
A6	Procurement Southern Delivery	07:30	15:30	Mon-Fri	37.5	7.5
A7	Procurement North Delivery	07:30	15:30	Mon-Fri	37.5	7.5
A8	Royal Gwent to St Woolos Sterile Instruments	07:30	15:30	Mon-Sat	45	7.5
A9	Royal Gwent to Caerphilly	08:30	16:30	Mon-Fri	37.5	7.5
A10	Patient Notes - Royal Gwent	08:00	16:00	Mon-Fri	37.5	7.5
A11	Pathology - Western Valley	08:30	16:30	Mon-Fri	37.5	7.5
A12	Ad-hoc/Haematology/taxi/misc	08:00	16:00	See schedule		7.5
A13	NNH Notes	07:00	10:00	Mon-Fri	20	4
A14	Rhymney Valley	09:30	18:00	Mon-Fri	37.5	7.5
A15	Call Centre (Ad-hoc)	08:00	16:00	Mon-Fri	37.5	7.5
A16	Saturday & Sunday Service	08:00	16:00	Sat & Sun	7.5	7.5
B1	Courier Service Route A	10:30	16:00	Mon-Fri	27.5	5.5
B2	Courier Service Route B	07:30	16:00	Mon-Fri + 2.5 OT	40	8
B3	Courier Service Route C	09:00	17:00	Mon-Fri	37.5	7.5
B4	Courier Service Route D	08:30	16:30	Mon-Fri + 2.5 OT	40	8
B5	Courier Service Route E	09:00	17:30	Mon-Fri + 1.5 OT	39	7.8
B6	Courier Service Route F	09:00	17:30	Mon-Fri + 2.5 OT	40	8
B7	Courier Service Route G	09:00	17:00	Mon-Fri	37.5	7.5
B8	Courier Service Route H	08:30	16:30	Mon-Fri + 1.5 OT	39	7.8
B9	Courier Service South Mail Shared Vehicle	10:30	18:30	Mon-Fri	37.5	7.5
B10	Courier Service North Mail Shared Vehicle	10:00	18:00	Mon-Fri	37.5	7.5
B11	Courier Service South Valley Hospital CSSD/Pharmacy	08:00	16:00	Mon-Fri	37.5	7.5
B12	Courier Service Additional to above	09:00	17:00	Saturday	7.5	7.5
B13	Courier Service Bank Holiday	10:30	15:30	Bank Holiday	5	5

Table 6: Summary of current Fixed Schedules of the HCS

E HCS Fixed Schedules Investigated

WAST HCS Cardiff & Vale Schedule

Schedule: 52

Shift Hours: 08:30 -17:00 Monday - Friday (37.5 hours a week)

Base: Lansdowne HCS office

Collect and deliver specimens, mail and all other items as required, routinely

from all locations as shown below.

08:30 Carry out daily vehicle checks. Collect vehicle and internal mail from Lansdowne HCS offices for delivery to:-

08:40 Lansdowne Community Buildings. Exchange mail.

09:00 UHW. Post Room, Specimen Reception, Histopathology (Collect specimens for Llandough Cytology).

09:15 Temple of peace and Welsh office exchange mail.

09:20 CRI. Exchange mail and specimens for Llandough. Also check for any specimens for UHW.

09:25 West Wing collect mail and specimens for Llandough

09:40 Llandough Hospital. Switchboard, Post Room, Specimen Reception and Theatre Reception (Medical Instruments

10:00 Lansdowne HCS office. Unload/load vehicle.

10:10 Rookwood Hospital. Main Reception exchange mail and collect specimens.

10:25 Velindre Hospital. Exchange mail and collect specimens from Blood Room.

10:35 Whitchurch Hospital. Exchange mail and collect specimens.

Also collect Nurse Bank & Finance pouches for delivery to Lansdowne Post Room.

UHW franking mail to be collected every lap and delivered directly to UHW Post Room.

10:40 Whitchurch UHB HQ. Exchange internal mail items.

11:00 UHW. Post Room, Specimen Reception, Histopathology (Collect specimens for Llandough Cytology).

11:20 CRI. Exchange mail and specimens for Llandough. Also check for any specimens for UHW.

11:25 West Wing collect mail and specimens for Llandough.

11:40 Llandough Hospital. Switchboard, Post Room, Specimen Reception and Theatre Reception (Medical Instruments).

Collect Bank Nurses Blue Metal box for delivery to Lansdowne Community

Deliver all Nurse Bank items to Lansdowne Community.

And finance items to Lansdowne finance

12:00 Lansdowne HCS office. Unload vehicle. **Lunch Break**

13:00 Lansdowne HCS office. Load vehicle and commence delivery.

13:10 Ely Bridge Surgery. Exchange mail and collect specimens to deliver to UHW.

13:15 Rookwood Hospital. Main Reception exchange mail and collect specimens.

Trenewydd LHB. Exchange mail.

13:25 Velindre Hospital. Exchange mail and collect specimens from Blood Room.

13:35 Whitchurch Hospital. Exchange mail and collect specimens.

13:40 Whitchurch Hospital UHB HQ. Exchange internal mail items.

13:55 Pentwyn Dialysis Unit (Cardiff North Renal Unit). Exchange mail and collect specimens.

14:00 UHW. Post Room, Specimen Reception, Histopathology (Collect specimens for Llandough Cytology).

14:20 CRI. Exchange mail and specimens for Llandough. Also check for any specimens for UHW.

14:25 West Wing collect mail and specimens for Llandough.

14:40 Llandough Hospital. Switchboard, Post Room, Specimen Reception and Theatre Reception (Medical Instruments). Cytology as required.

15:00 Lansdowne HCS office. Unload/load vehicle.

15:10 Rookwood Hospital. Reception exchange mail and collect specimens.

15:25 Velindre Hospital. Exchange mail and collect specimens from Blood Room.

15:35 Whitchurch Hospital. Exchange mail and collect specimens.

15:40 Whitchurch Hospital UHB HQ. Exchange internal mail items.

15:50 UHW. Post Room, Specimen Reception, Histopathology (Collect specimens for Llandough Cytology).

16:20 CRI Exchange mail and specimens for Llandough.

16:25 West Wing collect mail and specimens for Llandough.

16:40 Llandough - on this journey ensure that ALL items for onward transmission, including internal mail, are collected **- do not collect specimens**

17:00 Lansdowne HCS Office unload and return vehicle.

END OF DUTY

These schedule times and locations must be strictly adhered to unless authorized by HCS Control and/or the HCS Supervisor.

WAST HCS Cardiff & Vale Schedule

Schedule: S5

Shift Hours: 08:30 — 13:30 Saturday

Base: Lansdowne HCS office

Collect and deliver specimens, mail and all other items as required, routinely

from all locations as shown below.

08:30 Lansdowne Hospital. Collect vehicle and commence shift.

08:35 St David's Hospital. Collect family planning specimens from the security room. For delivery to Llandough Specimen Reception.

08:45 Llandough Hospital. Deliver specimens.

09:00 Barry Hospital. Collect specimens and drug boxes for delivery to Llandough Hospital.

09:30 Llandough Hospital. Deliver Barry Hospital drug boxes to Main Pharmacy and specimens to Specimen Reception. Check Post Room for any mail items needing delivery to Lansdowne HCS office. Also collect any specimens from Theatre Reception for delivery to UHW.

10:45 Velindre Hospital. Collect specimens and deliver to UHW Specimen Reception.

11:00 UHW. Specimen Reception, deliver all specimens from Velindre Hospital and Llandough Hospital.

12:00 Llandough Hospital. Pharmacy, collect drugs for Barry Hospital.

12:30 Barry Hospital. Deliver pharmacy.

13:30 Lansdowne HCS office. Unload and return vehicle.

END OF DUTY

During substantial breaks between collection/deliveries staff must be available for Ad-hoc duties, mail sorting etc.

These schedule times and locations must be strictly adhered to unless authorized

By HCS Control and/or the HCS Supervisor.

WAST HCS Cardiff & Vale Schedule

Schedule: S6

Shift Hours: 08:30 - 17:00 Monday - Friday (37.5 hours a week)

Base: Lansdowne HCS office

Collect and deliver all specimens, mail and all other items as required, routinely from all locations as listed below.

08:30 Carry out daily vehicle checks. Collect vehicle and internal mail from Lansdowne HCS offices for delivery to:-

08:40 St David's Hospital. Main Reception exchange mail and collect specimens.

08:50 Llandough Hospital. Switchboard, Post Room, Specimen Reception and Theatre Reception (Medical Instruments).

09:10 Rookwood Hospital. Main Reception exchange mail and collect specimens.

09:25 Velindre Hospital. Exchange mail at Post Room, collect specimens at Blood Room. Also check for any blood boxes to be returned to UHW Blood Bank.

09:35 Whitchurch Hospital. Exchange mail and collect specimens.

Also collect Nurse Bank & Finance pouches for delivery to Lansdowne Post Room. UHW franking mail to be collected every lap and delivered directly to UHW Post Room.

10:00 UHW. Post Room, Specimen Reception, Histopathology (Collect specimens for Llandough Cytology).

10:20 St David's Hospital. Exchange mail and collect specimens at main reception.

Dental Unit first floor. Collect Dental boxes to return to UHW Dental unit at 12:00 noon.

10:40 Llandough Hospital**.** Post Room, Specimen Reception, Cytology deliver Histopathology specimens from UHW.

Switchboard collect Nurse Bank & Finance boxes for delivery to Lansdowne.

11:00 Lansdowne HCS office. Unload/load vehicle.

11:10 Rookwood Hospital. Main Reception exchange mail and collect specimens.

11:25 Velindre Hospital. Exchange mail and collect specimens from Blood Room.

11:35 Whitchurch Hospital Pharmacy Department. Collect UWIC stores.

11:40 Whitchurch Hospital. Exchange mail and collect specimens.

12:00 UHW. Post Room, Specimen Reception, and Histology if required.

Deliver dental boxes to UHW Dental unit reception 2.

12:10 UWIC Podiatry Department Western Ave. Deliver Stores items to Main Reception and exchange internal mail.

12:20 St. David's Hospital. Exchange mail and collect specimens at main reception.

12:40 Llandough Hospital. Switchboard, Post Room, Specimen Reception, Theatre Reception and Cytology as required.

13:00 Lansdowne HCS office. Unload/load vehicle. **Lunch Break.**

14:00 Depart Lansdowne.

14:10 Rookwood Hospital. Main Reception exchange mail and collect specimens. Also collect mail for franking for delivery to UHW.

14:25 Velindre Hospital. Exchange mail and collect specimens from Blood Room.

14:35 Whitchurch Hospital. Exchange mail and collect specimens.

15:00 UHW. Post Room, Specimen Reception, and Histology as required.

15:15 St David's Court Surgery. 68a Cowbridge Rd East, exchange mail and collect specimens for delivery to UHW.

15:20 St. David's Hospital. Exchange mail and collect specimens at main reception.

15:40 Llandough Hospital. Switchboard, Post Room, Specimen Reception, Theatre Reception and Cytology as required.

16:00 Lansdowne HCS office. Unload/load vehicle.

16:10 Ely Bridge Surgery. Collect specimens for UHW.

16:15 Rookwood Hospital. Exchange mail and collect specimens only for UHW.

16:25 Velindre Hospital. Exchange mail and collect specimens at Blood Room.

Collect Nantgarw mail sack for delivery to Lansdowne.

16:30 Whitchurch Hospital. Exchange mail and collect specimens at main reception.

Also collect specimens from The Park Lodge (DORS) Team, previously Tegfan Day Hospital.

16:40 UHW. Post Room, Specimen Reception, and Histolo gy as required.

(Deliver only at this time.)

17:00 Lansdowne HCS Office unload and return vehicle.

END OF DUTY

These schedule times and locations must be strictly adhered to unless authorized by HCS Control and/or the HCS Supervisor.

WAST HCS Cardiff & Vale Schedule

Schedule: S7

Shift Hours: 11:00 — 15:30 Monday — Friday (22.5 hours per week)

Base: Lansdowne HCS office

Collect and deliver all specimens, mail and all other items as required, routinely from all locations as listed below.

11:00 Collect vehicle and internal mail from Lansdowne HCS offices for delivery to: -

11:40 187 City Rd Surgery.

11:50 Roath Clinic.

12:00 63 Wellwood Surgery, Llanederyn.

12:10 46 St Isan Rd Surgery.

12:20 104 Caerphilly Rd Surgery.

12:25 84 Caerphilly Rd Surgery.

12:30 182 North Rd Surgery.

12:35 210 Whitchurch Rd Surgery.

12:40 Highfields Centre, Allensbank Rd.

12:45 UHW Specimen Reception Deliver specimens.

13:00 151 Newport Rd Surgery.

13:05 116 Newport Rd Surgery.

13:15 98 Wentloog Rd, Daintree Surgery.

13:20 842 Newport Rd, Rumney Medical Practice.

13:25 C.E.L.T Llanrumney Ave Health Centre.

13:30 Clan Yr Afon School (Mon, Wed, Fri only)

13:40 Llanrumney Medical Centre, Ball Rd.

13:50 Willowbrook Surgery, 5 Strathy Rd.

13:55 Rainbow House Flying Start office, 1 Newent Rd.

14:00 Brynderwyn Surgery, Crickhowell Rd.

14:05 Trowbridge Health Centre, Abergele Rd.

14:15 Minster Rd Surgery.

14:20 100 Penylan Rd, Roath House Surgery.

14:25 74 Penylan Rd, Penylan Surgery.

14:30 219/221 City Rd Surgery Roath.

14:40 92 Salisbury Rd Surgery.

14:45 137 Cathays Terrace.

14:50 Crwys Medical Centre, Wedal Rd.

15:00 UHW Specimen Reception. Deliver all specimens.

15:30 Lansdowne HCS offices. Unload and return vehicle.

END OF DUTY

These schedule times and locations must be strictly adhered to unless authorized by HCS Control and/or the HCS Supervisor.

Figure 1: Details for the HCS Fixed Schedules Investigated

References

A. Attanasio, J-F. Cordeau, G. Ghiani, and G. Laporte. Parallel tabu search heuristics for the dynamic multi-vehicle dial-a-ride problem. *Parallel Computing*, 30(3):377–387, 2004.

N. Azi, M. Gendreau, and J.-Y. Potvin. A dynamic vehicle routing problem with multiple delivery routes. *Annals of Operations Research*, pages 1–10, 2012.

R. Baldacci, E. Bartolini, and A. Mingozzi. An exact algorithm for the pickup and delivery problem with time windows. *Operations Research*, 59(2):414–426, 2011.

R. Baldacci, A. Mingozzi, and R. Roberti. Recent exact algorithms for solving the vehicle routing problem under capacity and time window constraints. *European Journal of Operational Research*, 218(1):1–6, 2012.

J. E. Beasley. OR-library: Distributing test problems by electronic mail. *Journal of the Operational Research Society*, 41(11), 1990.

A. Beaudry, G. Laporte, T. Melo, and S. Nickel. Dynamic transportation of patients in hospitals. *OR Spectrum*, 32:77–107, 2010.

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.

G. Berbeglia, J-F. Cordeau, I. Gribkovskaia, and G. Laporte. Static pickup and delivery problems: a classification scheme and survey. *TOP*, 15:1–31, 2007.

G. Berbeglia, J-F. Cordeau, and G. Laporte. Dynamic pickup and delivery problems. *European Journal of Operational Research*, 202(1):8–15, 2010.

D.J. Bertsimas and D. Simchi-Levi. A new generation of vehicle routing research: Robust algorithms, addressing uncertainty. *Operations Research*, 44(2):286–304, 1996.

D.J. Bertsimas and G. Van Ryzin. A stochastic and dynamic vehicle routing problem in the euclidean plane. *Operations Research*, 39(4):601–615, 1991.

D.J. Bertsimas and G. Van Ryzin. Stochastic and dynamic vehicle routing in the euclidean plane with multiple capacitated vehicles. *Operations Research*, 41(1):60–76, 1993.

C. Blum and A. Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Comput. Surv.*, 35(3):268–308, 2003.

L. Bodin and B. Golden. Classification in vehicle routing and scheduling. *Networks*, 11:97–108, 1981.

L. Bodin, B. Golden, A. Assad, and M. Ball. Routing and scheduling of vehicles and crews: The state of the art. *Computers & Operations*, 10(2):63–211, 1983.

R. Borndörfer, M. Grötschel, Klostermeier F., and C. Küttner. Telebus berlin: Vehicle scheduling in a dial-a-ride system. Technical Report SC 97-23, Konrad-Zuse-Zentrum für Informationstechnik Berlin, 1997.

P. Bouros, D. Sacharidis, T. Dalamagas, and T. Sellis. Dynamic pickup and delivery with transfers. *Advances in Spatial and Temporal Databases*, pages 112–129, 2011.

J. Branke, M. Middendorf, G. Noeth, and M. Dessouky. Waiting strategies for dynamic vehicle routing. *Transportation Science*, 39(3):298–312, 2005.

O. Bräysy and M. Gendreau. Tabu search heuristics for the vehicle routing problem with time windows. *TOP*, 10:211–237, 2002.

O. Bräysy and M. Gendreau. Vehicle routing problem with time windows, part I: Route construction and local search algorithms. *Transportation Science*, 39:104–118, 2005a.

O. Bräysy and M. Gendreau. Vehicle routing problem with time windows, part II: Metaheuristics. *Transportation Science*, 39:119–139, 2005b.

E.G. Carabetti, S.R. de Souza, and M.C.P. Fraga. An application of the ant colony system metaheuristic to the vehicle routing problem with pickup and delivery and time windows. In *Eleventh Brazilian Symposium on Neural Networks (SBRN)*, pages 176–181, 2010.

M. Caramia, G.F. Italiano, G. Oriolo, A. Pacifici, and A. Perugia. Routing a fleet of vehicles for dynamic combined pickup and delivery services. In *Proceedings of the Symposium on Operation Research*, pages 3–8. Springer-Verlag, Berlin/Heidelberg, 2001.

H-K. Chen, C-F. Hsueh, and M-S. Chang. The real-time time-dependent vehicle routing problem. *Transportation Research Part E: Logistics and Transportation Review*, 42 (5):383–408, 2006.

W-C. Chiang and R.A. Russell. A reactive tabu search metaheuristic for the vehicle routing problem with time windows. *INFORMS Journal on Computing*, 9:417–430, 1997.

N. Christofides. Vehicle routing. In E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, and D.B. Shmoys, editors, *The Traveling Salesman Problem. A Guided Tour of Combinatorial Optimization*, pages 431–448. Wiley, Chichester, 1985.

N. Christofides and S. Eilon. An algorithm for the vehicle-dispatching problem. *Operations Research*, 20:309–318, 1969.

N. Christofides, R. Mingozzi, and P. Toth. The vehicle routing problem. In N. Christofides, R. Mingozzi, P. Toth, and C. Sandi, editors, *Combinatorial Optimization*, pages 315–338. John Wiley & Sons, Chichester, 1979.

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.

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, 2003.

J-F. Cordeau and G. Laporte. The dial-a-ride problem: models and algorithms. *Annals of Operations Research*, 153(1):29–46, 2007.

C.E. Cortés, M. Matamala, and C. Contardo. The pickup and delivery problem with transfers: Formulation and a branch-and-cut solution method. *European Journal of Operational Research*, 200(3):711–724, 2010.

L. Coslovich, R. Pesenti, and W. Ukovich. A two-phase insertion technique of unexpected customers for a dynamic dial-a-ride problem. *European Journal of Operational Research*, 175(3):1605–1615, 2006.

G. B. Dantzig and J. H. Ramser. The truck dispatching problem. *Management Science*, 6(1):80–91, 1959.

U. Dergis and T. Dohmer. Indirect search for the vehicle routing problem with pickup and delivery and time windows. *OR Spectrum*, 30:149–165, 2008.

G. Desaulniers, J. Desrosiers, A. Erdmann, M.M. Solomon, and F. Soumis. Vrp with pickup and delivery. In P. Toth and D. Vigo, editors, *The Vehicle routing problem*, volume 9 of *SIAM Monographs on Discrete Mathematics and Applications*, pages 225–242. SIAM, Philadelphia, 2002.

M. Desrochers, J.K. Lenstra, and M.W.P. Savelsbergh. A classification scheme for vehicle routing and scheduling problems. *European Journal of Operational Research*, 46:322–332, 1990.

M. Desrochers, J. Desrosiers, and M. Solomon. A new optimization algorithm for the vehicle routing problem with time windows. *Operations Research*, 40:342–354, 1992.

J. Desrosiers, F. Soumis, and M. Desrochers. Routing with time windows by column generation. *Networks*, 14:545–565, 1984.

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*, 6:301–326, 1986.

J. Desrosiers, Y. Dumas, M.M. Solomon, and F. Soumis. Time constrained routing and scheduling. In M.O. Ball, T.L. Magnanti, C.L. Monma, and G.L. Nemhauser, editors, *Handbooks in Operations Research and Management Science*, volume 8 of *Network Routing*, pages 35–139. North-Holland, Amsterdam Elsevier, 1995.

G. Ding, L. Li, and Y. Ju. Multi-strategy grouping genetic algorithm for the pickup and delivery problem with time windows. In *GEC '09: Proceedings of the first ACM/SIGEVO Summit on Genetic and Evolutionary Computation*, pages 97–104, New York, NY, USA, 2009. ACM.

Y. Dumas, J. Desrosiers, and F. Soumis. The pickup and delivery problem with time windows. *European Journal of Operational Research*, 54:7–22, 2001.

B. Eksioglu, A. Volkan Vural, and A. Reisman. The vehicle routing problem: A taxonomic review. *Computers and Industrial Engineering*, 57(4):1472–1483, 2009.

A. Fabri and P. Recht. On dynamic pickup and delivery vehicle routing with several time windows and waiting times. *Transportation Research Part B: Methodological*, 40(4):335–350, 2006.

C. Fiegl and C. Pontow. Online scheduling of pick-up and delivery tasks in hospitals. *Journal of Biomedical Informatics*, 42:624–632, 2009.

M. Fischetti, P. Toth, and D. Vigo. A branch-and-bound algorithm for the capacitated vehicle routing problem on directed graphs. *Operation Research*, 42:846–859, 1994.

M. L. Fisher and R. Jaikumar. A generalized assignment heuristic for vehicle routing. *Networks*, 11(2):109–124, 1981.

M.L. Fisher. Vehicle routing. In M.O. Ball, T.L. Magnanti, C.L. Monma, and G.L. Nemhauser, editors, *Handbooks in Operations Research and Management Science*, volume 8 of *Network Routing*, pages 1–33. North-Holland, Amsterdam Elsevier, 1995.

Y. Gajpal and P.L. Abad. Multi-ant colony system (MACS) for a vehicle routing problem with backhauls. *European Journal of Operational Research*, 196:102–117, 2009.

L.M. Gambardella, Ẽ.D. Taillard, and G. Agazzi. Macs-vrptw: A multiple ant colony system for vehicle routing problems with time windows. In D. Corne, M. Dorigo, and F. Glover, editors, *New Ideas in Opimization*, pages 63–76. London, McGraw Hill, 1999.

M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-completeness*. WH Freeman and Company, New York, 1979.

T.J. Gaskell. Bases for vehicle fleet scheduling. *Operations Research*, 18:281–295, 1967.

M. Gendreau and J-Y. Potvin. *Dynamic vehicle routing and dispatching*. Kluwer, Boston, 1998.

M. Gendreau and C.D. Tarantilis. Solving large-scale vehicle routing problems with time windows: The state-of-the-art. Technical Report 2010-04, CIRRELT, University of Montreal, 2010.

M. Gendreau, A. Hertz, and G. Laporte. A tabu search heuristic for the vehicle routing problem. *Management Science*, 18:1276–1290, 1994.

M. Gendreau, F. Guertin, Y-J. Potvin, and Ẽ.D. Taillard. Tabu search for real-time vehicle routing and dispatching. Technical Report CRT-96-47, Centre de recherche sur les transports, Universit de Montral, Montral, Canada, 1996.

M. Gendreau, G. Laporte, and J-Y. Potvin. Vehicle routing: modern heuristics. In E. Aarts and J.K. Lenstra, editors, *Local Search in Combinatorial Optimization*, pages 311–336. Wiley, Chichester, 1997.

M. Gendreau, F. Guertin, Y-J. Potvin, and R. Seguin. Neighborhood search heuristics for a dynamic vehicle dispatching problem with pick-ups and deliveries. Technical Report CRT-98-10, Centre de recherche sur les transports, Universit de Montral, Montral, Canada, 1998.

M. Gendreau, F. Guertin, Y-J. Potvin, and Ẽ.D. Taillard. Parallel tabu search for real-time vehicle routing and dispatching. *Transportation Science*, 33:381 – 390, 1999.

M. Gendreau, F. Guertin, Y-J. Potvin, and R. Seguin. Neighborhood search heuristics for a dynamic vehicle dispatching problem with pick-ups and deliveries. *Transportation Research Part C: Emerging Technologies*, 14(3):157–174, 2006.

G. Ghiani, F. Guerriero, G. Laporte, and R. Musmanno. Real-time vehicle routing: Solution concepts, algorithms and parallel computing strategies. *European Journal of Operational Research*, 151(1):1–11, 2003.

G. Ghiani, E. Manni, A. Quaranta, and C. Triki. Anticipatory algorithms for same-day courier dispatching. *Transportation Research Part E: Logistics and Transportation Review*, 45(1):96–106, 2009.

B.E. Gillett and L.R. Miller. A heuristic algorithm for the vehicle-dispatch problem. *Operations Research*, 22:340–349, 1974.

F. Glover. Future paths for integer programming and links to artificial intelligence. *Computers and Operations Research*, 13:533–549, 1986.

F. Glover. Tabu search part i. *ORSA Journal on Computing*, 1(3):190, 1989.

C. Groër, B. Golden, and E. Wasil. A library of local search heuristics for the vehicle routing problem. *Mathematical Programming Computation*, 2:79–101, 2010.

T. Hanne, T. Melo, and S. Nickel. Bringing robustness to patient flow management through optimized patient transports in hospitals. *Interfaces*, 39(3):241–255, May-June 2009.

G. Hasle and O. Kloster. Industrial vehicle routing. In G. Hasle, K-.A. Lie, and E. Quak, editors, *Geometric Modelling, Numerical Simulation, and Optimization*, pages 397–435. Springer Berlin Heidelberg, 2007.

P. L. Holborn, J.M. Thompson, and R. Lewis. Combining heuristic and exact methods to solve the vehicle routing problem with pickups, deliveries and time windows. In J.-K. Hao and M. Middendorf, editors, *Evolutionary Computation in Combinatorial Optimisation*, volume 7245 of *Lecture Notes in Computer Science*, pages 63–74. Berlin: Springer-Verlag, 2012.

J. H. Holland. *Adaptation in Natural and Artificial Systems*. The University of Michigan Press, Ann Arbor, MI., 1975.

L. Hong. An improved lns algorithm for real-time vehicle routing problem with time windows. *Computers & Operations Research*, 39(2):151 – 163, 2012.

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 2009a.

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 2009b.

E. Hyytiä, L. Häme, A. Penttinen, and R. Sulonen. Simulation of a large scale dynamic pickup and delivery problem. In *Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques*, SIMUTools '10, pages 1–77, 2010.

S. Ichoua, M. Gendreau, and J-Y. Potvin. Diversion issues in real-time vehicle dispatching. *Transportation Science*, 34(4):426–438, 2000.

S. Ichoua, M. Gendreau, and J-Y. Potvin. Exploiting knowledge about future demands for real-time vehicle dispatching. *Transportation Science*, 40:211–225, 2006.

J.J. Jaw, A.R. Odoni, H.N. Psaraftis, and N.H.M. Wilson. A heuristic algorithm for the multi-vehicle advance request dial-a-ride problem with time windows. *Transportation Research Part B: Methodological*, 20:243–257, 1986.

W-R. Jih and J. Yung-Jen Hsu. Dynamic vehicle routing using hybrid genetic algorithms. *Proceedings of the IEEE International Conference on Robotics and Automation*, 1:453–458, 1999.

D.E. Joslin and D.P. Clements. 'squeaky wheel' optimization. *Journal of Artificial Intelligence Research*, 10:353–373, 1999.

C. Kalapesi, S. Willersdorf, and P. Zwillenberg. The connected kingdom: How the internet is transforming the u.k. economy. *The Boston Consulting Group*, 2010.

Y. Kergosien, C. Lenté, D. Piton, and J-C. Billaut. A tabu search heuristic for a dynamic transportation problem of patients between care units. Technical report, CCSd/HAL : e-articles server (based on gBUS) [http://hal.ccsd.cnrs.fr/oai/oai.php] (France), 2010.

M.R. Khouadjia, B. Sarasola, E. Alba, L. Jourdan, and E-G. Talbi. A comparative study between dynamic adapted pso and vns for the vehicle routing problem with dynamic requests. *Applied Soft Computing*, 12(4):1426 – 1439, 2012.

P. Kilby, P. Prosser, and P. Shaw. Dynamic vrp's: a study of scenarios. Technical Report APES-06-1998, Univeristy of Strathclyde, 1998.

G.A.P. Kindervater and M.W.P. Savelsbergh. Vehicle routing: handling edge exchanges. In E. Aarts and J.K. Lenstra, editors, *Local Search in Combinatorial Optimization*. Wiley, Chichester, 1997.

K.W. Knight and J.P. Hofer. Vehicle scheduling with timed and connected calls: A case study. *Operational Research Quarterly*, 19:299–310, 1968.

V.A. Knight, P.R. Harper, and L. Smith. Ambulance allocation for maximal survival with heterogeneous outcome measures. *Omega*, 40(6):918–926, 2012.

D. Koning, JM. van den Akker, and JA. Hoogeveen. Using column generation for the pickup and delivery problem with disturbances. Master's thesis, Department of Computer Science, Utrecht University, 2011.

G. Kontoravdis and J.F. Bard. A grasp for the vehicle routing problem with time windows. *ORSA Journal on Computing*, 7(1):10, 1995.

A. Lackner. Dynamische tourenplanung mit ausgewhlten metaheuristiken. In J. Biethahn and M. Schumann, editors, *Gttinger Wirtschaftsinformatik*, Band 47. Cuvillier Verlag, 2004.

S. Landry and R. Philippe. How logistics can service healthcare. *Supply Chain Forum: an International Journal*, 5(2):24–30, 2004.

G. Laporte. The vehicle routing problem: An overview of exact and approximate methods. *European Journal of Operational Research*, 59:345–358, 1992.

G. Laporte. Fifty years of vehicle routing. *Transportation Science*, 43:408–416, 2009.

G. Laporte and Y. Nobert. Exact algorithms for the vehicle routing problem. In S. Martello, G. Laporte, M. Minoux, and C. Ribeiro, editors, *Surveys in Combinatorial Optimization*, pages 147–184. North-Holland, Amsterdam, 1987.

G. Laporte and I.H. Osman. Routing problems: A bibliography. *Annals of Operations Research*, 61:227–262, 1995.

A. Larsen. *The dynamic vehicle routing problem*. PhD thesis, Department of Informatics and Mathematical Modeling, Technical University of Denmark, 2001.

H.C. 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, 2001.

J.K. Lenstra and A.H.G. Rinnooy Kan. Complexity of vehicle routing and scheduling problems. *Networks*, 11:221–227, 1981.

H. Li and A. Lim. A metaheuristic for the pickup and delivery problem with time windows. *IEEE International Conference on Tools with Artificial Intelligence*, pages 160–7, 2001.

H. Li, A. Lim, and J. Huang. Local search with annealing-like restarts to solve the vrptw. Technical report, Research paper, Department of Computer Science, National University of Singapore, 2001.

J.-Q. Li, P.B. Mirchandani, and D. Borenstein. Real-time vehicle rerouting problems with time windows. *European Journal of Operational Research*, 194(3):711–727, 2009.

H. Lim, A. Lim, and B. Rodrigues. Solving the pickup and delivery problem with time windows using 'squeaky wheel' optimization with local search. In *Proceedings of Americal Conferecr on Information Systems, AMCIS 2002*, pages 2335–2344, Dallas, USA, 2002.

S. Lin. Computer solutions of the travelling salesman problem. *Bell Systems Technical Journal*, 44(10):2245–2269, 1965.

S. Lin and B. Kernigham. An effective heuristic algorithm for the traveling-salesman problem. *Operations Research*, 21:498–516, 1973.

Q. Lu and M.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, 2006.

K. Lund, O.B.G. Madsen, and Rygaard. Vehicle routing problems with varying degrees of dynamism. Technical report, Institute of Mathematical Modelling, Technical University of Denmark, 1996.

O. Madsen, H. Ravn, and J. Rygaard. A heuristic algorithm for a dial-a-ride problem with time windows, multiple capacities, and multiple objectives. *Annals of Operations Research*, 60:193–208, 1995.

O.B.G. Madsen. Optimal scheduling of trucks - a routing problem with tight due times for delivery. In R. Strobel, R. Genser, and M. Etschmaier, editors, *Optimization Applied to Transportation Systems*, pages 126–136. International Institute for Applied System Analysis, CP-77-7, Laxenburgh, Austria, 1976.

T.L. Magnanti. Combinatorial optimization and vehicle fleet planning: Perspectives and prospects. *Networks*, 11:179–214, 1981.

E. Melachrinoudis and H. Min. A tabu search heuristic for solving the multi-depot, multi-vehicle, double request dial-a-ride problem faced by a healthcare organisation. *International Journal of Operational Research*, 10(2):214–239, 2011.

E. Melachrinoudis, A.B. Ilhan, and H. Min. A dial-a-ride problem for client transportation in a health-care organization. *Computers & Operations Research*, 34:742–759, 2007.

H. Min. The multiple vehicle routing problem with simultaneous delivery and pick-up points. *Transportation Research Part A: General*, 23(5):377–386, 1989.

S. Mitrovic-Minic and G. Laporte. Waiting strategies for the dynamic pickup and delivery problem with time windows. *Transportation Research Part B: Methodological*, 38(7):635–655, 2004.

S. Mitrovic-Minic and G. Laporte. The pickup and delivery problem with time windows and transshipment. *INFOR*, 44(3):217–227, 2006.

S. Mitrovic-Minic, R. Krishnamurti, and G. Laporte. Double-horizon based heuristics for the dynamic pickup and delivery problem with time windows. *Transportation Research Part B: Methodological*, 38(8):669–685, 2004.

F. A. T. Montané and R. D. Galvão. A tabu search algorithm for the vehicle routing problem with simultaneous pick-up and delivery service. *Computers and Operations Research*, 33(3):595–619, 2006.

R. Montemanni, L.M. Gambardella, A.E. Rizzoli, and A.V. Donati. A new algorithm for a dynamic vehicle routing problem based on ant colony system. In *ODYSSEUS 2003: Proceedings of the Second International Workshop on Freight Transportation and Logistics*, Palermo, Italy, 2003.

R. Montemanni, L. M. Gambardella, A. E. Rizzoli, and A. V. Donati. Ant colony system for a dynamic vehicle routing problem. *Journal of Combinatorial Optimization*, 10(4):327–343, 2005.

Q. Mu, Z. Fu, J. Lysgaard, and R. Eglese. Disruption management of the vehicle routing problem with vehicle breakdown. *Journal of the Operational Research Society*, 62 (4):742–749, 2010.

Y. Nagata and S. Kobayashi. Guided ejection search for the pickup and delivery problem with time windows. In P. Cowling and P. Merz, editors, *Evolutionary Computation in Combinatorial Optimization*, volume 6022 of *Lecture Notes in Computer Science*, pages 202–213. Springer Berlin / Heidelberg, 2010a.

Y. Nagata and S. Kobayashi. A memetic algorithm for the pickup and delivery problem with time windows using selective route exchange crossover. In R. Schaefer, C.s Cotta, J. Kolodziej, and G. Rudolph, editors, *Parallel Problem Solving from Nature, PPSN XI*, volume 6238 of *Lecture Notes in Computer Science*, pages 536–545. Springer Berlin / Heidelberg, 2010b.

W.P. Nanry and J.W. Barnes. Solving the pickup and delivery problem with time windows using reactive tabu search. *Transportation Research Part B: Methodological*, 34(2):107–121, 2000.

I. Or. *Traveling salesman-type combinatorial problems and their relation the logistics of regional blood banking.* PhD thesis, Northwestern University, Evanston, IL., 1976.

I.H. Osman. Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem. *Annals of Operations Research*, 41:421–451, 1993.

G. Pankratz. A grouping genetic algorithm for the pickup and delivery problem with time windows. *OR Spectrum*, 27:21–41, 2005a.

G. Pankratz. Dynamic vehicle routing by means of a genetic algorithm. *International Journal of Physical Distribution and Logistics Management (IJPDLM)*, 35(5):362–383, 2005b.

G. Pankratz and V. Krypczyk. Benchmark data sets for dynamic vehicle routing problems. http://www.fernuni-hagen.de/WINF/inhalte/benchmark_data.htm, 2009.

S.N. Parragh. *Ambulance Routing Problems with Rich Constraints and Multiple Objectives.* PhD thesis, University of Vienna, Faculty of Business, Economics and Statistics, 2009.

S.N. Parragh. Introducing heterogeneous users and vehicles into models and algorithms for the dial-a-ride problem. *Transportation Research Part C: Emerging Technologies*, 19(5):912–930, 2011.

S.N. Parragh, K.F. Doerner, and R.F. Hartl. A survey on pickup and delivery problems: Part I: Transportation between customers and depot. *Journal Fr Betriebswirtschaft*, 58:21–51, 2008a.

S.N. Parragh, K.F. Doerner, and R.F. Hartl. A survey on pickup and delivery problems: Part II: Transportation between pickup and deliery locations. *Journal Fr Betriebswirtschaft*, 58:81–117, 2008b.

M. Pavone, N. Bisnik, E. Frazzoli, and V. Isler. A stochastic and dynamic vehicle routing problem with time windows and customer impatience. *Mobile Networks and Applications*, 14:350–364, 2009.

V. Pillac, M. Gendreau, C. Guéret, and A.L. Medaglia. A review of dynamic vehicle routing problems. *European Journal of Operational Research*, 225:1–11, 2013.

L.S. Pitsoulis and M.G.C. Resende. Greedy randomized adaptive search procedures. In P.M. Pardalos and M.G.C. Resende, editors, *Handbook of Applied Optimization*, pages 168–183. Oxford University Press, 2002.

A. Potini and G. Viola. *CabDispatcher: sistema software per linstradamento di una flotta di veicoli nellambito di un servizio di trasporto collettivo*. PhD thesis, Università di Roma "Tor Vergata", Rome, 2002.

J-Y. Potvin and J-M. Rousseau. An exchange heuristic for routeing problems with time windows. *Journal of the Operational Research Society*, 46:1433–1446, 1995.

H. Psaraftis. Dynamic vehicle routing problems. In B.L. Golden and A.A. Assard, editors, *Studies in Management Science and Systems*, volume 16, pages 223–248. North Holland, Amsterdam, 1988.

H. Psaraftis. Dynamic vehicle routing: status and prospects. *Annals of Operational Research*, 61:143–164, 1995.

H.N. Psaraftis. A dynamic programming solution to the single vehicle many-to-many immediate request dial-a-ride problem. *Transportation Science*, 14(2):130–154, 1980.

H.N. Psaraftis. An exact algorithm for the single vehicle many-to-many dial-a-ride problem with time windows. *Transportation Science*, 17(3):351–357, 1983a.

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, 1983b.

H. Pullen and M. Webb. A computer applicaion to a transport scheduling problem. *Computer Journal*, 10:10–13, 1967.

V. Pureza and G. Laporte. Waiting and buffering strategies for the dynamic pickup and delivery problem with time windows. *INFOR*, 46(3):165–176, 2008.

B. Rekiek, A. Delchambre, and H.A. Saleh. Handicapped person transportation: An application of the grouping genetic algorithm. *Engineering Applications of Artificial Intelligence*, 19(5):511–520, 2006.

R. Ropke and D. Pisinger. An adaptive large neighborhood search heuristic for the pickup and delivery problem with time windows. *Transportation Science*, 40(4): 455–472, 2006a.

S. Ropke and D. Pisinger. A unified heuristic for a large class of vehicle routing problems with backhauls. *European Journal of Operational Research*, 171:750–775, 2006b.

K.S. Rutland and E.Y. Rodin. The pickup and delivery problem: Faces and branch-and-cut algorithm. *Computers & Mathematics with Applications*, 33(12):1–13, 1997.

D. Sáez, C. E. Cortés, and A. Núñez. Hybrid adaptive predictive control for the multi-vehicle dynamic pick-up and delivery problem based on genetic algorithms and fuzzy clustering. *Computers & Operations Research*, 35(11):3412–3438, 2008.

M. Savelsbergh and M. Sol. Drive: Dynamic routing of independent vehicles. *Operations Research*, 46(4):474–490, 1998.

M. W. P. Savelsbergh and M. Sol. The general pickup and delivery problem. *Transportation Science*, 29(1):17–29, 1995.

M.W.P. Savelsbergh. Local search in routing problems with time windows. *Annals of Operations Research*, 4:285–305, 1985.

T. Sexton and L. Bodin. Optimizing single vehicle many-to-many operations with desired delivery times: I. Scheduling. *Transportation Siience*, 19:378–410, 1985a.

T. Sexton and L. Bodin. Optimizing single vehicle many-to-many operations with desired delivery times: II. Routing. *Transportation Siience*, 19:411–435, 1985b.

J.S. Shang and C.K. Cuff. Multicriteria pickup and delivery problem with transfer opportunity. *Computers & Industrial Engineering*, 30(4):631–645, 1996.

P. Shaw. Using constraint programming and local search methods to solve vehicle routing problems. In M. Maher and J-F. Puget, editors, *Principles and Practice of Constraint Programming CP98*, volume 1520 of *Lecture Notes in Computer Science*, pages 417–431. Berlin / Heidelberg, Springer, 1998.

SINTEF. Transportation Optimization Portal - TOP Li & Lim Benchmark. `http://www.sintef.no/Projectweb/TOP/PDPTW/Li--Lim-benchmark/`, 2004. Updated: December 12, 2011.

L. Smith, P.R. Harper, V.A. Knight, and I.T. Vieira. Google Maps Travel Matrix Generator - User Guide. `http://www.cardiff.ac.uk/maths/research/researchgroups/opresearch/healthcare/index.html`, November 2011.

M. M. Solomon. Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations Research*, 35(2):254–265, 1987.

M.M. Solomon, E. Baker, and J. Schaffer. Vehicle routing and scheduling problems with time window constraints: Efficient implementations of solution improvemnt procedures. In B.L. Golden and A.A. Assad, editors, *Vehicle Routing: Methods and Studies*, pages 85–105. North-Holland, Amsterdam Elsevier, 1988.

A. Subramanian, L.M.A. Drummond, C. Bentes, L.S. Ochi, and R. Farias. A parallel heuristic for the vehicle routing problem with simultaneous pickup and delivery. *Computers & Operations Research*, 37(11):1899–1911, 2010.

J.M. Sussman. Intelligent transportation systems in a real-time, customer-oriented society. *The Bridge*, 38(2), 2008.

M.R. Swihart and J.D. Papastavrou. A stochastic and dynamic model for the single-vehicle pick-up and delivery problem. *European Journal of Operational Research*, 114(3):447–464, 1999.

Ẽ.D. Taillard, P. Badeau, M. Gendreau, F. Guertin, and J-Y. Potvin. A tabu search heuristic for the vehicle routing problem with soft time windows. *Transportation Science*, 31(2):170–186, 1997.

K.C. Tan, L.H. Lee, Q.L. Zhu, and K. Ou. Heuristic methods for vehicle routing problem with time windows. *Artificial Intelligence in Engineering*, 15:281–295, 2001.

R. Tavakkoli-Moghaddama, A.R. Saremia, and M.S. Ziaee. A memetic algorithm for a vehicle routing problem with backhauls. *Applied Mathematics and Computation*, 181:1049–1060, 2006.

D. Teodorovic and G. Radivojevic. A fuzzy logic approach to dynamic dial-a-ride problem. *Fuzzy Sets and Systems*, 116(1):23–33, 2000.

P.M. Thompson and H.N. Psaraftis. Cyclic transfer algotihms for multivehicle routing and scheduling problems. *Operations Research*, 41:935–946, 1993.

P. Toth and D. Vigo. Fast local search algorithms for the handicapped persons transportation problem. In I.C.H. Osman and J.P. Kelly, editors, *Meta-heuristics: theory and applications*, pages 677–690. Kluwer, Norwell, 1996.

P. Toth and D. Vigo. Heuristic algorithms for the handicapped persons transportation problem. *Transportation Science*, 31(1):60–71, 1997.

P. Toth and D. Vigo. An overview of vehicle routing problems. In P. Toth and D. Vigo, editors, *The Vehicle routing problem*, volume 9 of *SIAM Monographs on Discrete Mathematics and Applications*, pages 1–26. SIAM, Philadelphia, 2002a.

P. Toth and D. Vigo. Branch-and-bound algorithms for the capacitated vrp. In P. Toth and D. Vigo, editors, *The Vehicle routing problem*, volume 9 of *SIAM Monographs on Discrete Mathematics and Applications*, pages 29–51. SIAM, Philadelphia, 2002b.

P. Toth and D. Vigo. Vrp with backhauls. In P. Toth and D. Vigo, editors, *The Vehicle routing problem*, volume 9 of *SIAM Monographs on Discrete Mathematics and Applications*, pages 195–224. SIAM, Philadelphia, 2002c.

B. Turan, V. Schmid, and K.F. Doerner. Models for intra-hospital patient routing. In *3rd IEEE International Symposium on Logistics and Industrial Informatics (LINDI)*, pages 51–60, aug. 2011.

A. Van Breedam. Comparing descent heuristics and metaheuristics for the vehicle routing problem. *Computers and Operations Research*, 28(4):289–315, 2001.

L.J.J. Van Der Bruggen, J.K. Lenstra, and P.C. Schuur. Variable-depth search for the single-vehicle pickup and delivery problem with time windows. *Transportation Science*, 27:298–311, 1993.

M. Wen, J-. F. Cordeau, G. Laporte, and J. Larsen. The dynamic multi-period vehicle routing problem. *Computers & Operations Research*, 37(9):1615–1623, 2010.

H. Wilson and N. Colvin. Computer control of the Rochester dial-a-ride system. Technical Report R-77-31, Department of Civil Engineering, MIT, Cambridge, MA, 1977.

H. Wilson and H. Weissberg. Advanced dial-a-ride algorithms research project: Final report. Technical Report R76-20, Department of Civil Engineering, MIT, Cambridge, MA, 1976.

H. Wilson, J. Sussman, H. Wang, and B. Higonnet. Scheduling algorithms for dial-a-ride systems. Technical Report USL TR-70-13, Urban Systems Laboratory, MIT, Cambridge, MA, 1971.

Z. Xiang, C. Chu, and H. Chen. The study of a dynamic dial-a-ride problem under time-dependent and stochastic environments. *European Journal of Operational Research*, 185(2):534–551, 2008.

J. Yang, P. Jaillet, and H. Mahmassani. Real-time multivehicle truckload pickup and delivery problems. *Transportation Science*, 38:135–148, 2004.

C.W. Yuen, K.I. Wong, and A.F. Han. Waiting strategies for the dynamic dial-a-ride problem. *International Journal of Environment and Sustainable Development*, 8: 314–329, 2009.