THE DYNAMIC PICKUP AND DELIVERY PROBLEM WITH TIME WINDOWS

by

Snežana Mitrović-Minić

B.Sc. University of Belgrade 1987
M.Sc. University of Belgrade 1994

A THESIS SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY
in the School
of
Computing Science

SIMON FRASER UNIVERSITY
November 2001

APPROVAL

Name: Snežana Mitrović-Minić

Degree: Doctor of Philosophy

Title of Thesis: The Dynamic Pickup and Delivery Problem with Time Windows

Examining Committee: Dr. Thomas C. Shermer, Chair

Dr. Ramesh Krishnamurti, Co-Senior Supervisor

Dr. Gilbert Laporte, Co-Senior Supervisor

Dr. Arvind Gupta, Supervisor

Dr. Binay Bhattacharya, SFU Examiner

Dr. Fayez Fouad Boctor, External Examiner

Date Approved:

Abstract

This thesis investigates the large-scale dynamic pickup and delivery problem with time windows, and proposes heuristic solution approaches. The pickup and delivery problem with time windows deals with finding a set of optimal routes for a fleet of vehicles in order to serve transportation requests. Each transportation request requires pickup of a load at the pickup location and delivery of the load at the delivery location. The load is too small for capacity constraints to be relevant, and each stop location has to be served within a given time window. The dynamic pickup and delivery problem with time windows considers the process of solving the problem in an environment where not all requests are known in advance. While vehicles are serving requests, new requests are coming in, and their assignment is to be done on-line. Our focus is on the pure dynamic problem for which the future requests are not stochastically modelled or predicted. This models the problem faced by dispatchers in courier companies that serve same-day pickup and delivery requests for transporting letters and small parcels.

The major results of our research include: a two-goal dynamic model for a dynamic routing problem, a two-strategy heuristic (a framework in which any heuristic or metaheuristic can be embedded), a dynamic utility function for the evaluation of one location insertion, recognizing the importance of developing a schedule in a dynamic environment (as opposed to a static environment), analysis of simple waiting strategies, design of complex waiting strategies for solving the route scheduling problem, and the use of precedence graphs for testing feasibility of insertion of a request into a route in the pickup and delivery problem with time windows. (The

precedence graphs can also be used for finding bounds on the number of vehicles for the multiple traveling salesman problem with time windows.)

Mojim dragim roditeljima
mami Spomenki i tati Dušanu.

To my dear parents
Spomenka and Dušan Minić.

Acknowledgments

It has been a real pleasure and privilege working with both of my senior supervisors, Prof. Gilbert Laporte and Prof. Ramesh Krishnamurti.

I would like to thank Prof. Gilbert Laporte for initiating the research, allowing me freedom to follow my own interests, and providing valuable comments and suggestions that helped shape this research. Many discussions by phone and throughout Canada, in Vancouver, Montréal and Edmonton, were very productive and truly inspiring. I am grateful for the encouragement he provided during the whole process of research. His trust and belief in my work have counted for a lot. His effectiveness during meetings, and his promptness and precision in replying to each e-mail message gave me additional impetus on my way.

I would like to thank my senior supervisor, Prof. Ramesh Krishnamurti for his valuable help, excellent research environment, and encouragement. His availability for discussions and his kindness were amazing. He was always able to meet me whenever I needed to talk, and the discussions were always rich, fruitful and inspiring. He has been incredibly generous with his time and insights while reading this manuscript in progress. His boundless patience has indeed helped shape this thesis and improved its readability.

My gratitude also goes to External Examiner Prof. Fayez F. Boctor and to SFU Examiner Binay Bhattacharya for providing valuable feedback that has served to improve this thesis.

I would like to thank Prof. Arvind Gupta for organizing and directing the first of my research assistant appointments at SFU. I would also like to thank Dr. Daya Gaur for interesting and

fruitful discussions and suggestions at the beginning of my research.

I want to thank my friends for valuable and important advice along the way, and for contributing their time in reading the thesis: Desanka Polajnar, Prof. Jernej Polajnar, Prof. Ljiljana Trajković, and Christine Stojaković.

Special thanks go to administrative and technical staff at the School of Computing Science.

I would also like to acknowledge significant importance of my early postgraduate research years done under supervision of Prof. Dragoš Cvetković, Prof. Slobodan Guberinić and Dr. Gordana Šenborn.

I am very thankful to numerous friends (my sister included) and colleagues that helped in their special ways: by giving a helping hand, interesting advice, or just being there to listen and talk.

However all of this would have been much harder to achieve, if not impossible, without my parents, their enormous help, dedication and support during all my educational years, and especially during my PhD studies. Their winter-spring stays in Vancouver, dedicated care of and enormous patience with my kids were precious to me.

Lastly, but most importantly, I would like to thank my family for their support. I would like to thank my husband, Boban, for being there when I needed him the most, through all the ups and downs of our lives, and of this research. For being with our kids many evenings and many weekends, for helping me a lot, and for allowing me to pursue my goals. Also, thanks go to my kids Marko, Jelena and Dušan for being helpful and quiet when needed, for sleeping through nights when it counted a lot, and for giving me joy and laughs through all these years, even when I did not feel like it.

Finally, I highly appreciate the financial support, in the form of research assistantships, of my senior supervisors Prof. Ramesh Krishnamurti and Prof. Gilbert Laporte, as well as of my supervisor Prof. Arvind Gupta. I am very grateful for Prof. Ramesh Krishnamurti's grants that made possible my trip to Edmonton, and my trip and one-month stay in Montréal. I would also like to acknowledge the financial support, in the form of fellowships, from SFU and FAS, SFU.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This thesis investigates the dynamic pickup and delivery problem with time windows, and proposes heuristic solution approaches. Our problem is faced by dispatchers in courier companies that serve same-day pickup and delivery requests for transporting letters and small parcels. The problem is of large-scale.

The pickup and delivery problem with time windows deals with finding a set of optimal routes for a fleet of vehicles in order to serve transportation requests. Each transportation request is defined by two stop locations, a pickup location and a delivery location, and requires pickup of a load at the pickup location and delivery of the load at the delivery location. The load is of no consequence compared to the vehicle capacity. Each stop location has to be served within a given time window.

The dynamic pickup and delivery problem with time windows considers the process of solving the problem in an environment where not all requests are known in advance. While vehicles are serving requests, new requests are coming in, and their assignment is to be done on-line. Dynamic problems are often called real-time problems. Our focus is on the pure dynamic problem whose future requests are not stochastically modelled or predicted.

1.1 Motivation

1.1.1 Dynamic problems

Recently, dynamic problems have become an interesting research area. Processing of real-time data is feasible today due to advances in information technology and the telecommunication industry. In particular, Global Positioning Systems (GPS), Geographical Information Systems (GIS), Intelligent Vehicle Highway Systems (IVHS) and similar systems, are increasing the importance of solution methods and strategies for dynamic routing problems. This is recognized by many researchers including Dror, Powell (1993):

> "The design of solution methodologies and computational procedures for this [dynamic] kind of problem represents a significant challenge and opportunity for the operations research community." [pp.11]

Psaraftis (1995) has also analyzed dynamic vehicle routing problems and described their importance:

> "Within systems of such size [as transportation systems are], even a modest percentage improvement in the overall cost of distribution can be very important for the overall competitiveness of the industries on the market, both internal and external, and might very well be critical for the survival of many businesses that depend on such distribution." ... "All those sectors will benefit from an improved distribution scheme in dynamic environment." [pp.148]

Many routing problems in practice are in fact dynamic and, because research on dynamic routing problems is in its early stages, much yet has to be done. Compared to static routing problems one can find many similarities, but some differences, too. In contrast to static routing problems with well-defined objective functions, models and benchmarks, the area of dynamic routing problems is characterized by the lack of (Dror, Powell, 1993):

- standard problem formulations,
- well-defined objective functions,

- dynamic models,
- test data sets,
- criteria for comparing the solutions.

Also, the real-world offers a huge variety of problems, making comparisons between problems and between corresponding solution strategies even harder.

1.1.2 Dynamic pickup and delivery problem with time windows

The courier industry has grown steadily in the last several decades. Every big city in the world has several dozen courier companies whose main services are same-day pickup and delivery. The size of such companies varies from a few vehicles to a few hundred vehicles. A typical company can serve more than one thousand pickup and delivery requests every day.

A courier company's main task is to serve same-day pickup and delivery requests as they come-in during the day. The request assignment, the vehicle routing, and the route scheduling have to be solved on-line, while vehicles are driven along specific routes. A dispatcher solves these problems by taking into consideration (Shen, Potvin, Rousseau, Roy (1995)): the current time, the current vehicle positions, the routes that have already been scheduled, the stop locations and the time windows of the new requests, the distances and the travel times between locations, and the characteristics of the underlying network. The dispatcher's job is hard and stressful, and few dispatchers work for more than ten years. The courier company operation largely depends on a good dispatcher, and a good dispatcher is hard to find and difficult to train. The need for research on methods and policies for solving the real-time dispatching problem is therefore evident.

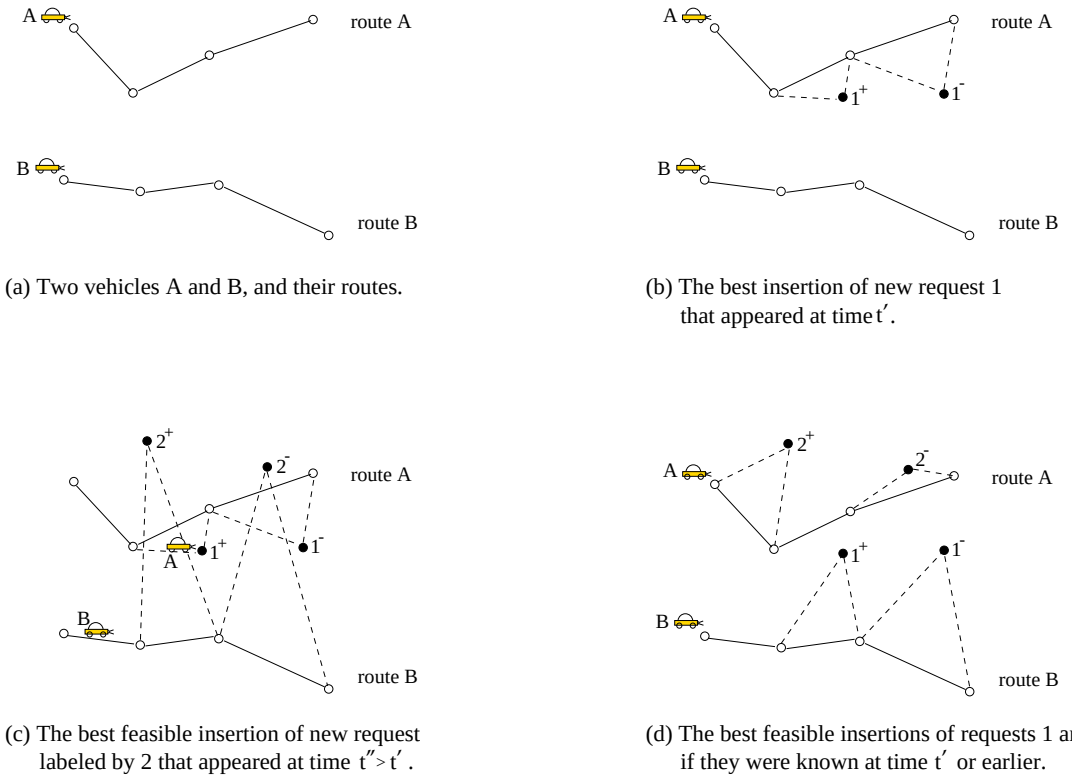
Furthermore, many other industries and their operations rely on the success, reliability and punctuality of courier companies. Making courier companies even more fast and reliable can make industry as a whole more effective and successful.

1.2 Solving large-scale dynamic problems

The most promising approach to a pure dynamic pickup and delivery problem is an on-line procedure that solves the problem repeatedly in time. With this procedure a solution that is currently available is being implemented (at each instant of the problem service period). In addition to finding such a solution, solving the problem involves designing a strategy (policy) that will be applied when new data becomes available. It is also worth mentioning that many difficulties that do not exist in the static environment may arise in the dynamic case.

We illustrate below one such a difficulty on a simple example. The goal is to minimize the total route lengths. Assume that we have two vehicles, A and B, whose current (future) routes are shown by solid lines in Figure 1.1 (a). Vehicle positions are also indicated. Each vehicle has to visit four locations. Assume a new request, labeled by 1, appears with pickup location at 1^+ and with delivery location at 1^- (see Figure 1.1 (b)). Since we want to minimize the distance travelled, we would insert request 1 into route A (assume the insertion is feasible). Unfortunately, after some time, another request, labeled by 2, appears with pickup location at 2^+ and with delivery location at 2^- (Figure 1.1 (c)). Insertion of request 2 in route A is not feasible, but would have been feasible if request 1 had not been inserted in route A. Location reassignment may not be possible, due to other time constraints (for example, as shown in Figure 1.1 (c), vehicle A may already be on the way to location 1^+). Thus, it is not possible to minimize the total distance travelled, which would have been possible if we had known all requests at the beginning (Figure 1.1 (d)), *i.e.*, if we were solving the corresponding static problem.

Moreover, the majority of practical routing problems are of large-scale. Powell (1998) has mentioned that "as the problem becomes messier, the solution strategies become simpler" [pp.148]. Some recent studies on the dynamic traveling repairmen problem suggests that simple heuristics like nearest-neighbor achieved very good results in large-scale instances (also called high-demand instances). Preliminary results of our empirical study has also shown that simple strategies achieve good results when used for solving scheduling aspect of the problem. Con-



(a) Two vehicles A and B, and their routes.

(b) The best insertion of new request 1 that appeared at time t'.

(c) The best feasible insertion of new request labeled by 2 that appeared at time $t'' > t'$.

(d) The best feasible insertions of requests 1 and 2 if they were known at time t' or earlier.

Figure 1.1: The dynamic pickup and delivery problem with time windows.

sequently, we recommend that the development of an on-line method (for solving a dynamic problem) include algorithm engineering tailored to cover an empirical study of simple and naive heuristics.

Modern heuristics have given very promising results in solving a variety of static vehicle routing problems. However, in the case of dynamic problems when the time available for decision making is short, one has to be careful. As stated by (Gendreau, Hertz, Laporte, 1992) modern heuristic such as simulated annealing and tabu search are open-ended improvement procedures whose performance is directly related to running time. These methods might not be the most appropriate techniques for quickly generating good tours. In solving large-scale dynamic pickup

and delivery problems with time windows, quick responses may be required, especially when changing the portion of a route along which a vehicle will drive in the near future. Nevertheless, modern heuristics can be used as improvement procedures, especially over those portions of routes along which vehicles will drive later in time. (It is worth mentioning that many modern heuristics also have a characteristic, described by Gendreau, Laporte, Vigo (1999) in the case of tabu search, of being "a generic technique that must be carefully tailored to the problem at hand in order to produce good results". [pp.707])

1.3 Thesis overview

Our approach concentrates on solving large-scale pure dynamic pickup and delivery problem with time windows. This research has involved theoretical methods together with thorough experimental investigation which enabled a better understanding of our problem and its behavior in the dynamic environment. These insights help one to obtain theoretical results, and build a heuristic approach which addresses most of the issues arising in solving dynamic pickup and delivery problems with time windows. The major results of our research include:

- the development of the two-goal dynamic model for a dynamic routing problem;
- the development of the two-strategy heuristic for solving the dynamic pickup and delivery problem with time windows - a framework in which any heuristic or metaheuristic can be embedded;
- a dynamic utility function for the evaluation of one location insertion;
- recognition of increased importance of the schedule development for a fixed route in dynamic vs. static environment;
- analysis of simple waiting strategies, and design of complex waiting strategies for solving the route scheduling problem;

- precedence graphs that can be used for testing feasibility of insertion of a request into a route of the pickup and delivery problem with time windows (and for finding bounds on the number of vehicles for the multiple traveling salesman problem with time windows).

Other important results are:

- the development of an on-line system for simulating and solving large-scale dynamic pickup and delivery problem with time windows; the system also allows monitoring and analysis of the process of solving the problem;
- dynamic route clustering in space and time;
- a formal description of the dynamic combinatorial optimization problem, the optimal dynamic solution, and the optimal static solution;
- empirical study of different heuristic versions.

The thesis starts with an overview of static and dynamic time-constrained vehicle routing problems. Chapter 3 gives an analysis of the real-world pickup and delivery problem based on research done in courier companies. A static model of the real-world problem is given afterwards. The second part of the chapter comprises an overview of the solution methodology. The general layout and elements of the on-line system for solving the dynamic pickup and delivery problem with time windows are specified. Chapter 4 gives details about solving the problem by a greedy insertion procedure. It also treats the testing of feasibility of a request insertion in a route by the usage of precedence graphs, and the complexity of finding the best request insertion. Chapter 5 outlines scheduling for a fixed route, which is more important in a dynamic environment. Chapter 6 covers a formal description of a dynamic optimization problem, a dynamic model of dynamic routing problem, and the two-strategy heuristic as an example of a dynamic on-line method. Finally, the empirical study herein together with the experimental results are discussed and analyzed in Chapter 7.

Chapter 2

An overview of time-constrained routing problems

2.1 Introduction

Routing problems involve the assignment of locations to vehicles such that the assignment cost and the corresponding routing cost are minimized (Desrosiers, Dumas, Solomon, Soumis, 1995). While the spatial aspect of the problem has been a major research subject for decades, much of the current research focuses on the time-constrained version of these problems. Usually, the time dimension has the form of time window constraints that restrict the start of service for a customer to begin after a prespecified release time and to end before a prespecified deadline. When the time windows are hard constraints, a vehicle has to serve a customer before the corresponding deadline. However, if a vehicle arrives too early it is permitted to wait. When the time windows are soft constraints, they can be violated at a cost.

Time-constrained routing problems can be found in industry and in the service sector. Practical problems have often been a source of research topics and as Osman (1993) stated: "The vehicle routing problem is an extremely active research area that has seen an exciting interplay

between theory and practice. It is probably one of the greatest success stories of operations research." [pp.423]

In all practical problems, the time at which the problem is fully known, with all its characteristics, data and information, plays an important role. Based on this time, researchers classify problems as static and dynamic. A *static problem* is a problem for which all the characteristics and data are known in advance. On the contrary, if some information becomes known during the process of problem solving and during the process of implementing the solution, the problem is called *dynamic*. Decisions have to be made and acted on before all the information about the problem is collected. The dynamic data of a routing problem might include real-time customer demands, traffic conditions, and driver statuses. In practice, the dynamic problem is often called a *real-time problem*.

This chapter presents an analysis of methods for solving time-constrained routing problems, covering both static and dynamic routing. Static routing has been extensively researched for the last several decades, while work on dynamic routing started just recently and is still in its infancy. Consequently the majority of methods developed solve static routing problems and any research on dynamic routing usually starts with studying static routing problems and solution methods.

This survey reports on methods for solving the traveling salesman problem with time windows, the vehicle routing problem with time windows, the pickup and delivery problem with time windows and capacity constraints, and the pickup and delivery problem with time windows. The abbreviations used in this chapter and throughout the thesis are:

TSP	- the traveling salesman problem,
TSPTW	- the traveling salesman problem with time windows,
m-TSPTW	- the multiple traveling salesman problem with time windows,
VRP	- the vehicle routing problem,
VRPTW	- the vehicle routing problem with time windows,
PDPTW	- the pickup and delivery problem with time windows,
PDPTWC	- the pickup and delivery problem with time windows and capacity constraints,
DARP	- the dial-a-ride problem,
HTP	- the handicapped transportation problem.

Our survey is not intended to be a report on all published methods, but presents methods that have contributed to developments in the PDPTW area.

2.2 The traveling salesman problem with time windows

The traveling salesman problem with time windows (TSPTW) is the problem of finding an optimal route passing through a set of locations (customers) and satisfying time window constraints. Each location has to be visited exactly once. The time window of a location is the time interval during which the location has to be visited.

TSPTW is a relaxation of the single-vehicle pickup and delivery problem with time windows. The additional constraints that exist in the pickup and delivery problem are pairing and precedence constraints (see page 20).

TSPTW is NP-hard in a strong sense because it is a generalization of TSP which is an NP-hard problem. Savelsbergh (1985) has proved that even finding a feasible solution of TSPTW is NP-hard. Problem analysis and surveys on solution methods for TSP and TSPTW can be found in several studies, *e.g.*, Lawler, Lenstra, Rinnooy Kan, Shmoys (1985); Desrosiers, Dumas, Solomon, Soumis (1995).

Most ***optimization algorithms*** developed for TSPTW are branch and bound methods, with a variety of bounding procedures. Dynamic programming type algorithms have also been developed.

A *branch and bound method* developed by Baker (1983) for TSPTW has been reported to solve 50-node problems when a small percentage of time windows overlap. It was one of the first such method for TSPTW. The author used a mathematical programming formulation of TSPTW whose dual was then bounded from above by the usage of the longest path algorithm. A *state-space relaxation* method for finding a lower bound was used by Christofides, Mingozzi, Toth (1981) for solving 50-node problems with moderately narrow time windows. These authors developed a forward dynamic programming algorithm with a recursion formula whose relaxation is the shortest q-path relaxation. The solution of the relaxed recursion formula is a lower bound to the initial TSPTW. Branching on flow variables was used to solve the problem optimally. The state-space relaxation method is analogous to Lagrangian relaxation in integer programming. Constraints in integer programming formulations appear as state variables in dynamic programming recursions, and hence constraint relaxation corresponds to state-space relaxation. *A two-commodity network flow formulation* for TSP has been extended to TSPTW by Langevin, Desrochers, Desrosiers, Gélinas, Soumis (1993). Each commodity corresponds to a resource that is either distributed or picked up along the tour of all nodes. The time window constraints of the problem are modelled by setting both the commodities to be time. Problems with up to 60 nodes were solved, with computational time strongly influenced by the time window widths. The time window widths, of the problem instances solved, were between ten and twenty minutes. This set of test instances was later used in empirical studies by many other researchers. An exact *constrained logic programming algorithm* was developed by Pesant, Gendreau, Potvin, Rousseau (1998). The constrained logic programming model of TSPTW yields an exact branch-and-bound optimization algorithm without any restrictive assumption on the time windows.

A *dynamic programming method*, based on time windows constraints, was developed by Dumas, Desrosiers, Gélinas, Solomon (1995). TSPTW instances with up to 200 nodes were solved,

with fairly wide overlapping time windows, in four minutes. As expected, the computational time increases exponentially with the width of time windows. However, for small time window widths the algorithm was capable of solving problems with 800 nodes in ten minutes. For these problems the CPU time increases linearly with the problem size.

Regarding the ***heuristic methods*** based on k-interchange concepts of Lin, Kernighan (1973); Or (1976), it has been proved by Savelsbergh (1985) that the 2-interchange or the restricted 3-interchange procedures perform a route feasibility check in constant time. A more complex heuristic introduced by Gendreau, Hertz, Laporte, Stan (1998) was based on the Gendreau, Hertz, Laporte (1992) GENIUS heuristic developed for TSP. The algorithm is a two-phase procedure that combines a constructive procedure (GENI) and a post-optimization procedure (US). The GENI is an insertion procedure with the incorporation of local optimization steps. The number of insertion attempts is lower, but each one is examined more carefully with the simultaneous testing of a limited number of local transformations of the tour. The US procedure consists of an unstringing and a stringing part. The stringing part is equivalent to GENI. The results reached were the best in the literature, in terms of solution quality and computational time. The solutions are either optimal or close to optimal for instances with up to 100 nodes and different time window widths. The running time does not appear to be affected by the time window widths.

Yet another type of heuristic starts by solving a relaxation problem of the original TSPTW, followed by a transformation into a solution of the original problem. One example is the heuristic proposed by Wolfler Calvo (2000) that starts by solving an assignment problem with a specific objective function. The solution of the assignment problem is a set of subtours, and it is usually a large number of subtours that is far from a feasible solution of TSPTW. The author has proposed an objective function that incorporates the total travel times and the time window constraints. This objective function increases the feasibility of TSPTW and in that way decreases the number of subtours in the solution of the assignment problem. The algorithm continues by inserting the subtours into a main route by a greedy insertion procedure. If feasibility is not reached, a k-opt

exchange procedure, together with a procedure for avoiding local optima, is used to improve the solution. The reported solutions are either optimal or near-optimal. On instances of size up to 100-nodes, the algorithm gets the best results among the heuristics compared. The results for instances with 150 and 200 nodes were also reported. The CPU time is less than five minutes for 100-node instances and between 20 and 30 minutes on 200-node instances.

One of the extensions of TSPTW is the *multiple traveling salesman problem with time windows* (m-TSPTW). The m-TSPTW is the problem of finding a set of routes for a fleet of vehicles in order to visit all customer locations. The m-TSPTW can be also viewed as a special case of the vehicle routing problem with time windows with relaxed capacity constraints. The m-TSPTW arises in school bus routing and in aircraft crew scheduling. In school bus routing the problem is one of scheduling a fleet of buses to serve a set of trips. Each trip has a start location and an end location. Each trip has to start within a certain time window, and has to be completed before the bus goes for another trip. Each trip is considered as one customer. The aircraft crew scheduling can be modelled similarly.

The most successful solution method for m-TSPTW uses the Dantzig-Wolfe decomposition/column generation method by Desrosiers, Soumis, Desrochers (1984). An instance with 151 trips is solved optimally. Unfortunately the method is highly sensitive to the time window widths. One of the first studies on heuristic approaches to solving m-TSPTW has been reported in Orloff (1976). The research since then has been scarce (references can be found in Desrosiers, Dumas, Solomon, Soumis (1995)).

2.3 The vehicle routing problem with time windows

The vehicle routing problem with time windows (VRPTW) is the problem of finding a set of optimal routes, starting and ending at a depot, for a fleet of vehicles that serves a set of customers with certain demands. Each customer has to be assigned to only one vehicle such that the vehicle capacity is not exceeded and the customer demand is satisfied. Each customer

imposes a time window which constrains the time when the customer can be served.

There are several versions of this problem. Some versions allow split loads, some deal with several depots, some are characterized by precedence relations among customers, some allow soft time windows, etc. VRPTW with relaxed capacity constraints is m-TSPTW.

Since both VRPTW and PDPTW are generalizations of m-TSPTW, and since research in the area of vehicle routing problems is extensive, the review of VRPTW methods can provide insights into PDPTW and other time-constrained routing problems. The vehicle routing problem can be considered as PDPTW where all VRPTW stop locations are pickup locations and the depot is a delivery location, or vice versa (the stop locations are delivery locations and the depot is a pickup location). The main similarities of the two problems lie in the time window constraints and the characteristic of handling a vehicle fleet.

VRPTW is an NP-hard problem. Even finding a feasible solution is NP-hard (Desrosiers, Dumas, Solomon, Soumis, 1995), when the number of vehicles is fixed. (This is a corollary of the results proved for TSPTW by Savelsbergh (1985).) As a consequence, many heuristics that have been developed allow extension to fleet size. (For the complexity of different versions of VRP refer to Lenstra, Rinnooy Kan (1981).)

The majority of methods for VRPTW are based on methods developed for VRP. Modelling issues and surveys on exact and approximate methods (classic and modern) for solving VRP can be found in Bodin, Golden (1981); Lenstra, Rinnooy Kan (1981); Magnanti (1981); Bodin, Golden, Assad, Ball (1983); Laporte, Nobert (1987); Assad (1988); Dror, Laporte, Trudeau (1989); Laporte (1992); Ball, Magnanti, Monma, Nemhauser (1995); Osman, Laporte (1996); Gendreau, Laporte, Potvin (1997); Laporte, Gendreau, Potvin, Semet (2000). Emphasis on modelling complex real-world VRPs is given in Schrage (1981). Surveys focussed on the time-constrained vehicle routing problem include Desrochers, Lenstra, Savelsbergh, Soumis (1988); Solomon, Desrosiers (1988); Desrosiers, Dumas, Solomon, Soumis (1995). Desaulniers, Desrosiers, Ioachim, Solomon, Soumis, Villeneuve (1998) have given a unified framework for time-constrained routing problems by giving a general model and by presenting a branch-and-

bound approach to solve nonlinear multi-commodity network flow models. The approach is an extension of the Dantzig-Wolfe decomposition/column generation method. Since many surveys exist, this overview focuses on methods that solve large-scale problem instances.

Many studies have dealt with ***optimally solving*** VRP, but the difficulty of the problem has constrained researchers to solving instances of relatively modest size (10-30 customer instances). Only if some specific characteristics are attached to VRP, larger instances could be optimally solved: an asymmetrical 260-customer instance solved by a branch and bound approach with the assignment problem as subproblem (Laporte, Mercure, Nobert, 1986), a special case of a 50-customer instance solved by dynamic programming method (Christofides, 1985b), etc. For further details, see the surveys of Laporte, Nobert (1987) and Laporte (1992). The first optimization method for VRPTW is a branch-and-bound procedure developed by Kolen, Rinnooy Kan, Trienekens (1987). The other optimization algorithms for VRPTW are also based on mathematical programming formulations of the problem and on approaches using branch-and-bound. The methods include usage of state-space relaxation, integer programming relaxation, Lagrangian relaxation, and column generation. The Dantzig-Wolfe decomposition/column generation method shows the best results in solving the largest instances of VRPTW. Even large-scale problems have been solved in the case of narrow time windows. The method was first proposed by Appelgren (1969) for solving a ship scheduling problem with time window constraints (cited in Desrosiers, Dumas, Solomon, Soumis (1995)). The method was later improved and used for solving m-TSPTW and many versions of VRPTW Desrosiers, Dumas, Soumis (1988); Desrosiers, Dumas, Soumis, Taillefer, Villeneuve (1991); Ioachim, Desrosiers, Dumas, Solomon, Villeneuve (1995). The 100-customer instances were solved by Desrochers, Desrosiers, Solomon (1992) within 30 minutes. As expected, the computation time increases as the time windows widen.

Despite the success of optimization methods, heuristics are still very popular, because they can solve much larger VRPTW in much less computation time. ***Heuristic methods*** developed for solving VRP can be characterized as: decomposition methods, construction methods, improvement methods, composite heuristics, and incomplete optimization methods (mathematical

programming based methods). A *construction method* is called *sequential construction* if it builds feasible routes one at a time. If several routes are built simultaneously, the procedure is called *parallel construction*. A large number of the construction methods have explored the two-phase methods known as the *'cluster first - route second'* or the *'route first - cluster second'* methods. The criterion usually used for clustering has been the vehicle capacity constraint. The insertion criteria have been: maximum savings, nearest neighbor and minimum additional distance and time. For early work on these concepts and methods applied on VRPTW see Solomon (1987).

An *improvement heuristic* strives to improve a feasible solution to the problem. The solution improvements can be done within each separate route, in an *intra-route improvement procedure*, or by applying some transformations among the routes, in an *inter-route improvement procedure*. The majority of improvement methods have been some kind of exchange procedures. Branch exchange methods Baker, Schaffer (1986); Solomon, Baker, Schaffer (1988) have been adaptations, modifications and improvements of the well-known k-opt interchange developed by Lin, Kernighan (1973) and the Or-opt interchange developed by Or (1976) for TSP. An extensive comparison of different 2-opt, 3-opt and Or-opt heuristics for VRPTW is given in Potvin, Rousseau (1995). Many variations of k-opt interchange procedures exist, including different restricted explorations of the search space. Among the interchange procedures known to be widely used in modern metaheuristics, such as tabu search and simulated annealing, is the λ-interchange proposed by Osman (1993). Simply described, the method explores exchanges of two subsets of nodes belonging to two different routes. All described improvement procedures belong to the class of *local search methods*. Roughly speaking, a local search algorithm starts from an initial solution and continually tries to find better solutions by searching neighborhoods Aarts, Lenstra (1997). This is why lately, the local search methods are called *the neighborhood search methods*.

A new concept in neighborhood search is the *cyclic transfer algorithm*, proposed by Thompson, Psaraftis (1993). The method is a complex inter-route improvement procedure with a generalized definition of a neighborhood. The method has been tested on Solomon's 100-customer

VRPTW instances. A *greedy randomized adaptive search procedure* (GRASP) by Kontoravdis, Bard (1995) was also tested on Solomon's 100-customer problems. GRASP combines greedy heuristics, randomization and local search. At its advantage, GRASP generates many good alternatives, that could be important for real-life situations where unpredictable conditions can happen, such as difficult weather conditions and traffic congestions. On the other hand, a disadvantage of GRASP is that the quality of solution depends strongly on the quality of the starting feasible solution. The GRASP method has solved a large 417-node problem. Another GRASP procedure proposed by Atkinson (1998) incorporates a learning strategy. The "learning" consists of an attempt to learn from previous mistakes by an appropriate updating of the greedy function from one run to the next. Both GRASP procedures perform very well on problems with tight time windows or tight capacity constraints. GRASP belongs to the group of methods called metaheuristics.

A metaheuristic is formally defined in Osman, Laporte (1996) as an iterative generation process which guides a subordinate heuristic, by combining intelligently different concepts for exploring and exploiting the search space, in order to find effectively near-optimal solutions. Application of many metaheuristics on VRP have shown that the method called *tabu search* usually reached the best quality solutions. The tabu search is a local search heuristic proposed by Glover (1989, 1990). The tabu search explores a neighborhood of the current solution and moves to the best neighbor even if the solution is worse than the current one. This allows the method to escape from a local optima, and enables the algorithm to explore more areas of the search space. Cycling is prevented by maintaining a list of forbidden "tabu" moves (solutions). The list contains visited solutions for a certain period of time. The analysis and comparisons of the best tabu search methods for VRP are given in Gendreau, Laporte, Potvin (1997); Laporte, Gendreau, Potvin, Semet (2000). It is reported that the best solutions were obtained by algorithms proposed by Taillard (1993); Rochat, Taillard (1995), followed closely solutions achieved by TABUROUTE (Gendreau, Hertz, Laporte, 1994). (TABOUROUTE incorporates the GENIUS approach described in Section 2.2 into a tabu search procedure.) Problems with

200 customers were solved in less than 1.5 hours. Many special cases of VRP are also solved by tabu search, such as periodic and multi-depot VRP (Cordeau, Gendreau, Laporte, 1997). These authors have reported solving large problems, with 250 and 360 customers, in less than 30 and 50 minutes respectively.

For VRPTW, Potvin, Kervahut, Garcia, Rousseau (1996) have developed a *tabu search* procedure with restricted 2-opt and Or-opt neighborhoods, that achieved very good results on Solomon's 100-customer problems. Several problems have even been solved optimally. The running time varies from five to ten minutes. A parallel implementation of this tabu search heuristic has been explained in Garcia, Potvin, Rousseau (1994). The vehicle routing problem with soft time windows has been solved by a method of Taillard, Badeau, Gendreau, Guertin, Potvin (1997) that uses an adaptive memory for generating good starting solutions, and both inter-route and intra-route exchanges for generating the neighborhoods. Adaptive memories, proposed by Rochat, Taillard (1995), stores routes associated with the best solutions visited. These routes can be used later for different purposes, but usually are used to provide a good new starting solution for the search. The Taillard, Badeau, Gendreau, Guertin, Potvin (1997) method generates very good solutions on 100-node Solomon problems: 37 best solutions out of 56 test problems. Problem instances with 417 requests have also been solved. A parallel implementation (Badeau, Guertin, Gendreau, Potvin, Taillard, 1997) of the same method, using a decomposition/reconstruction concept (Taillard, 1993), has shortened the computing time without degrading the quality of solutions, when implemented on a 16-processor computer. Periodic VRPTW and multi-depot VRPTW (Cordeau, Laporte, Mercier, 2000) were also solved by the tabu search method.

A tabu search heuristic (Gendreau, Guertin, Potvin, Taillard, 1999) for solving the dynamic VRPTW, an adaptation of the tabu search method initially designed for the static version of VRPTW (Badeau, Guertin, Gendreau, Potvin, Taillard, 1997), was tested on 100-request Solomon's instances, with the first 50 requests known in advance. Service period was close to an hour, and rejection of requests was allowed. A similar parallel tabu search algorithm (Ichoua,

Gendreau, Potvin, 2000) for solving the dynamic VRPTW has allowed diversion of vehicles from their current destination locations in order to reduce costs. The algorithm was tested on modified Solomon's 100-customer Euclidean instances. The Solomon's instances were changed by addition of dynamic aspect in the form of the specified appearance times of some requests. Two scenarios were tested: scenario 1 with half of the requests known in advance, and scenario 2 with 25% of the requests known in advance. The results were encouraging with slightly better improvements achieved in scenario 2.

Evolutionary metaheuristics (Homberger, Gehring, 1999), genetic algorithms (Thangiah, Nygard, Juell, 1992; Thangiah, Gubbi, 1992; Thangiah, 1993) (GIDEON), and simulated annealing (Chiang, Russell, 1996) have also been successfully applied to VRPTW. Interesting combinations of metaheuristics (GenSAT) has been proposed by Thangiah, Osman, Sun (1994).

It is worth mentioning that some improvement procedures allow violations of the feasibility of the current solution. The solution feasibility will be restored later. This enhancement facilitates an escape from local optima, but it is rarely recommended in dealing with time-constrained routing problems, because feasibility recovery could be a difficult task.

Note that the Solomon test problems that represent a benchmark for testing VRPTW methods are described in Solomon (1987).

2.4 The pickup and delivery problem with time windows

The pickup and delivery problem (PDP) is a problem of finding a set of optimal routes for a fleet of vehicles in order to serve transportation requests. Each transportation request is defined by a pickup location, a delivery location, and a load. The locations are also called the *stop locations*. When each stop location has defined a time interval within which it should be visited, the problem is known as *the pickup and delivery problem with time windows* (PDPTW). Vehicle capacities may or may not be bounded, causing the presence or absence of capacity constraints. PDPTW with vehicle capacities is a VRPTW with precedence and pairing constraints. PDPTW

with infinite vehicle capacities is a m-TSPTW with precedence and pairing constraints.

Precedence constraints deal with the restriction that each pickup location has to be visited prior to visiting the corresponding delivery location. *Pairing constraints* restrict the set of admissible routes such that one vehicle has to visit both the pickup location and the delivery location of one transportation request.

If the pickup and delivery locations are distinct, the problem is said to be *many-to-many* PDPTW. If the requests are served by one vehicle, the problem is called *the single-vehicle pickup and delivery problem with time windows* (1-PDPTW). If a fleet of vehicles is available, the problem is *the multi-vehicle pickup and delivery problem with time windows* (m-PDPTW or PDPTW). The problem can be characterized with one or more depots, or it can be without a depot. (The no-depot problem can be considered as a many-depot problem, where number of depots is equal to the number of vehicles.)

The version of the pickup and delivery problem with time windows, where not all requests are known before the service period starts, is called *the dynamic pickup and delivery problem with time windows* (or *real-time dispatching of pickup and delivery vehicles*). In such a problem the inflow of requests occurs in real-time, and many known requests have to be served before all requests are received.

2.4.1 Problem complexity

The pickup and delivery problem with time windows is a generalization of m-TSPTW, and therefore is NP-hard. The m-TSPTW is an NP-hard problem in the strong sense. Savelsbergh (1985) has proved, by a transformation from 3-partition, that even finding a feasible solution to TSPTW is NP-hard. As a corollary, the problem of finding a feasible solution to PDPTW when the number of vehicles is fixed, is also NP-hard.

Furthermore, Solomon (1986) proves for a variety of VRPTW heuristics that the worst-case ratio behavior on n-customer problems is $\Omega(n)$. Thus, the worst-case performance ratio for

all examined heuristics cannot be bounded by a constant. The examined objectives were the minimization of the number of vehicles, the minimization of the total distance travelled, and the minimization of the total schedule time. All the worst-case instances are characterized by the infinite vehicle capacities. (Each VRPTW instance is a m-TSPTW instance.)

These theoretical results suggest that the time-constrained routing problems are essentially more difficult than their non-time-constrained counterparts (with an exception of a problem with very narrow time windows). Thus, PDPTW is essentially more difficult than PDP. To render the problem of finding a feasible solution to PDPTW tractable, at least one of the following has to be allowed:

- extension to the fleet size,
- rejection of requests, or
- soft time windows.

2.4.2 The pickup and delivery problem with time windows and with capacity constraints

The pickup and delivery problem with time windows with capacity constraints (PDPTWC) is a generalization of the VRPTW.

Research on the pickup and delivery problem started almost three decades ago, and the majority of studies have dealt with the capacity-constrained version of pickup and delivery. The most studied problems have been:

- the dial-a-ride problem (DARP), that deals with transporting people;
- the handicapped transportation problem (HTP), that deals with transporting people with special needs;
- the pickup and delivery problem of transporting goods, with tight capacity constraints.

The problems of transporting goods are primarily concerned with the cost of the routes (distance, time, or a combination), while the problems that deal with transporting people mostly use a weighted combination of the following objectives:

- minimization of total time to service customers,
- minimization of total distance travelled,
- minimization of customers inconvenience or 'dissatisfaction'.

Customer inconvenience is usually a linear function of a customer waiting time and his/her excess riding time. Excess riding time is the difference between the actual riding time and the direct riding time from the customer pickup location to the customer delivery location.

The following paragraphs give an overview of the PDPTWC solution methods. Extended surveys on the pickup and delivery problems can be found in Bodin, Golden, Assad, Ball (1983); Savelsbergh, Sol (1995); Fisher (1995); Desrosiers, Dumas, Solomon, Soumis (1995); Powell, Jaillet, Odoni (1995); Assad (1988).

The PDPTWC was first examined by Wilson, Sussman, Wong, Higonnet (1970), when the authors developed real-time algorithms for dial-a-ride systems operating in Haddonfield, NJ and Rochester, NY. Their paper presents some fundamental concepts for solving pickup and delivery problems: building tours through sequential insertion of customers, and the general form of the objective function.

Early work on mathematical formulation of the problem was done in the late 1970s. The formulation has since changed and improved. See Savelsbergh, Sol (1995) for the latest work on the formulation of the general pickup and delivery problems.

Optimization algorithms for the pickup and delivery problem include dynamic programming methods and the Dantzig-Wolfe decomposition/column generation method.

The first optimization methods for solving pickup and delivery problems were the *dynamic programming algorithms*. Psaraftis (1980) worked on the single-vehicle immediate-request dial-

Optimization algorithms			
1-DARP	**Dynamic programming**. Forward dynamic programming, $O(n^2 3^n)$.	Psaraftis (1983a)	10 requests.
1-DARP	**Dynamic programming**. Forward dynamic programming. State elimination criteria very effective when the time windows are tight and the vehicle capacities are low.	Desrosiers, Dumas, Soumis (1986)	40 requests.

Table 2.1: Optimization methods for solving the single-vehicle pickup and delivery problem with time windows and with capacity constraints.

a-ride-problem. Static and dynamic versions of the problem were solved by a backward recursion scheme. Indefinite deferment of a request is prevented by introducing the maximum-position-shift constraint. The problem with time windows was also solved (Psaraftis, 1983a). A forward recursion was used in solving problem instances with ten requests. Desrosiers, Dumas, Soumis (1986) developed a new dynamic programming algorithm which was able to solve a 1-DARP with 40 requests.

Dumas, Desrosiers, Soumis (1991) developed a *mathematical programming based method* that optimally solves the pickup and delivery problem with time windows with tight capacity constraints. The authors embedded a complex decomposition procedure, the Dantzig-Wolfe decomposition/column generation, into a branch and bound tree, in order to reach exact solutions. Aimed to solve a specific class of pickup and delivery problems with multiple vehicles, the method is robust and easily adaptable to handle different objective functions, multiple depots, and a non-homogeneous fleet of vehicles. The size of the largest problem solved was 55 requests with 22 vehicles. The load demand distribution was approximately 40% of the full load, 40% of the half load and 20% of one third of the vehicle capacity.

Other papers in the area of PDPTWC deal with ***heuristic algorithms*** that can be divided into decomposition methods, construction methods, improvement methods, and mathematical

Optimization algorithms		
Problem	*Solution method*	*Authors/Instance*
PDPTWC	**Mathematical programming method**. Dantzig-Wolfe decomposition/column generation embedded in branch and bound tree. Master problem is set partitioning, subproblem is constrained shortest path problem.	Desrosiers, Dumas, Soumis, Taillefer, Villeneuve (1991) ..... 55 requests, 22 vehicles, tight capacity constraints.

Table 2.2: Optimization methods for solving the pickup and delivery problem with time windows and with capacity constraints.

programming methods. Many *decomposition methods* have implemented the 'cluster-first, route-second' idea, where clustering was done on the request load and on the request locations. Jaw, Odoni, Psaraftis, Wilson (1986) have developed a three-phase method by introducing a preliminary phase of clustering in time, for solving large-scale DARP. The algorithm has been improved in the (Jaw, Odoni, Psaraftis, Wilson, 1986) ADARPTW package and in the (Madsen, Ravn, Rygaard, 1995) package named REBUS. Among the *construction heuristic* the most known are the sequential construction heuristic (which builds one route at a time), and the parallel construction heuristic which builds several routes simultaneously. Requests are often ordered by some criteria. Many measures and many other criteria have been developed and tested. Most of the *improvement methods* have been based on local search. Note that, the design of an *improvement* local search procedure for PDPTWC has to deal with the precedence constraints, the pairing constraints, the capacity constraints, and the time windows constraints. Improvement procedures can be divided into intra-route and inter-route improvement procedures, but there are many methods that combine the two. The majority of *intra-route improvement* procedures are based on edge exchange concept proposed by Lin, Kernighan (1973) and Or (1976) for

solving TSP. Those procedures are known as k-opt interchange and Or-opt interchange. *Inter-route improvement* procedures deal with exchange of nodes among routes. Thompson, Psaraftis (1993) have investigated an application of the cyclic transfer algorithms to PDPTWC. Good experimental results were reported on problems with up to 52 locations and up to ten vehicles. Another successful idea is the ejection chain concept, proposed by Glover (1996). (For more details see explanation on page 29).

Heuristics			
Problem	*Solution method*	*Authors*	*Instance*
1-DARP	**Mathematical programming based heuristic**. Benders decomposition applied to a mixed 0-1 non-linear programming formulation. Decomposition separates the routing and scheduling component.	Sexton, Bodin (1985a,b)	20 requests.

Table 2.3: Heuristics for solving the single-vehicle pickup and delivery problem with time windows and with capacity constraints.

Heuristics based on mathematical programming methods include the application of Benders decomposition Sexton, Bodin (1985a,b), and the method called the Dantzig-Wolfe decomposition/column generation Desrosiers, Dumas, Soumis (1988); Dumas, Desrosiers, Soumis (1989); Desrosiers, Dumas, Soumis, Taillefer, Villeneuve (1991); Ioachim, Desrosiers, Dumas, Solomon, Villeneuve (1995). Benders decomposition, applied to a mixed binary nonlinear formulation of 1-DARP, separates the routing and scheduling components of the problem. An initial feasible route is built by a heuristic. Given the route, an optimal schedule is formed by solving a scheduling problem. The route is then altered, by a heuristic, if the optimal schedule based on the new route improves the objective function. This process continues until no new improved route and schedule can be found. Another mathematical programming based heuristic, the Dantzig-Wolfe

decomposition/column generation consists of mini-clustering of requests, followed by optimal routing. Mini-clustering has been modelled by an m-PDPTW that has been solved by a heuristic in Dumas, Desrosiers, Soumis (1989); Desrosiers, Dumas, Soumis, Taillefer, Villeneuve (1991) or by column generation in Ioachim, Desrosiers, Dumas, Solomon, Villeneuve (1995). (When solved by column generation, m-PDPTW becomes a set partitioning problem over the set of requests, and with subproblems that are the constrained shortest path problems with time windows.) The resulting routes are divided into mini-clusters, where each period of time when a vehicle is non-empty represents one mini-cluster. Over mini-clusters that are constructed, an optimal routing is done. The routing problem is m-TSPTW and it is solved by column generation as a set partitioning problem over the set of mini-clusters, with the shortest path problem with time windows as a subproblem. This method has succeeded in solving large-scale DARP with thousands of requests. For a detailed description of the method refer to Mitrović-Minić (1998). Ioachim, Desrosiers, Dumas, Solomon, Villeneuve (1995) have used the method for solving HTP and have argued that their application can be easily adapted to multi-dimensional capacity problems. Another interesting column generation technique was developed earlier - an interactive procedure Cullen, Jarvis, Ratliff (1981) that solves a set partitioning problem over the set of columns generated by a human. The method was used for solving DARP.

Modern heuristics were recently applied to pickup and delivery problems. Toth, Vigo (1995, 1997) have worked on the tabu threshold procedures for solving a real-life HTP. Potvin, Rousseau (1992) have developed a constraint-directed search algorithm for DARP. When tested on instances proposed by Jaw, Odoni, Psaraftis, Wilson (1986), the method reached solutions that are 5-15% better with running times which are two to five times longer. A reactive tabu search approach (Nanry, Barnes, 2000) has been used for solving PDPTWC. The method was tested on a set of instances built from Solomon's VRPTW instances. The set of PDPTWC instances were constructed in such a way that the results in solving them could be compared with the results reached in solving VRPTW.

The history of solving PDPTWC problems are summarized in four tables by presenting some

Heuristics		
Problem	*Solution method*	*Authors/Instance*
DARP	**Construction heuristic**. Insertion procedure.	Wilson, Sussman, Wong, Higonnet (1970) ..... 4586 requests, 136 vehicles.
DARP	**Decomposition heuristic**. Three phase approach: 1. Clustering in time. 2. Clustering in space. 3. Routing phase. Terms: active period, slack time.	Jaw, Odoni, Psaraftis, Wilson (1986) ..... 250 requests, 14 vehicles; 2600 requests, 20 vehicles.
DARP, HTP	**Mathematical programming based heuristic**. Dantzig-Wolfe decomp./column generation method: Mini-clustering followed by optimal routing. Mini-clustering is m-PDPTW solved in the following manner: construct routes by a heuristic, divide routes into mini-clusters. Routing problem is m-TSPTW. It is solved optimally by delayed column generation.	Desrosiers, Dumas, Soumis (1988); Dumas, Desrosiers, Soumis (1989); Desrosiers, Dumas, Soumis, Taillefer, Villeneuve (1991) ..... 190 requests, 31 vehicle, 85 mini-clusters; 880 requests, 53 vehicles, 282 mini-clusters in several time-slices; 2411 requests.
DARP	**Modern heuristics**. Constraint-directed search.	Potvin, Rousseau (1992) ..... 90 requests.
PDPTWC	**Improvement procedure**. Cyclic transfers.	Thompson, Psaraftis (1993) ..... 52 requests, 9 vehicles.
HTP	**Mathematical programming based heuristic**. Dantzig-Wolfe decomp./column generation method: Mini-clustering followed by optimal routing. Column generation is used for solving both m-PDPTW and m-TSPTW.	Ioachim, Desrosiers, Dumas, Solomon, Villeneuve (1995) ..... 2545 requests.
HTP	**Metaheuristic**. Tabu threshold algorithm used as an improvement procedure.	Toth, Vigo (1995, 1997) ..... 312 requests, multi-dimensional capacity constraints.
PDPTWC	**Metaheuristic**. Reactive tabu search.	Nanry, Barnes (2000) ..... 100-requests (modified Solomon's VRPTW instances).

Table 2.4: Heuristics for solving the pickup and delivery problem with time windows and with capacity constraints.

milestones. Table 2.1 and Table 2.3 deal with the single-vehicle pickup and delivery problem with time windows and capacity constraints (1-PDPTWC), while Table 2.2 and Table 2.4 illustrate the work on the multi-vehicle pickup and delivery problem with time windows and capacity constraints (m-PDPTWC).

2.4.3 The pickup and delivery problem with time windows and without capacity constraints

The pickup and delivery problem with time windows in which vehicle capacities are unlimited is called the pickup and delivery problem with time windows (PDPTW).

Despite the enormous body of research in the area of vehicle routing, little work has been done on PDPTW. One of the rare methods specifically developed for solving the static PDPTW is a variable-depth arc-exchange procedure proposed by Van Der Bruggen, Lenstra, Schuur (1993). The method solves 1-PDPTW, by variable depth search over basic types of arc-exchange procedures. The procedure has reached near-optimal solutions for problems with up to 38 requests, in reasonable time. The time windows were small comparing to the average travel times. A simulated annealing technique was used as an additional improvement procedure.

There are a few more methods developed for solving the dynamic PDPTW (see Table 2.5). Shen, Potvin, Rousseau, Roy (1995) developed an expert system whose aim was to help the dispatcher in a courier company. The learning module is a neural network, trained by an expert dispatcher. An interactive interface allows the dispatcher to take the final decision. The system is easily adaptable to different dispatching environments. It was tested on real-life problem instances with 140 requests, 12 vehicles, and with service period that lasted six hours. The time windows were defined by: the 30 minutes time allowed for pickup and the 90 minutes time allowed for delivery, from the time a request came in.

Another decision support system with learning capabilities (Benyahia, Potvin (1998)) uses a utility function that is aimed at approximating the decision process of a professional dispatcher.

Modern heuristics			
Problem	*Solution method*	*Authors*	*Instance*
1-PDPTW	**Improvement procedure and metaheuristic**. Variable-depth local search based on Lin-Kernighan arc-exchange method for TSP. Improvement procedure: simulated annealing.	Van Der Bruggen, Lenstra, Schuur (1993)	38 requests, narrow time windows, large vehicle capacities.
Dynamic PDPTW	Expert system based on neural networks.	Shen, Potvin, Rousseau, Roy (1995)	22 requests per hour.
Dynamic PDPTW	Genetic programming.	Benyahia, Potvin (1998)	22 requests per hour.
Dynamic PDPTW	Tabu search with ejection chains and adaptive memory.	Gendreau, Guertin, Potvin, Séguin (1998)	24-33 req. per hour.

Table 2.5: Heuristics for the dynamic time-constrained routing problems.

The utility function was pre-constructed by a genetic programming technique, and it was than applied as a part of the dispatcher decision support system. The real-life instances, described in Shen, Potvin, Rousseau, Roy (1995), were used to test the genetic programming approach. The genetic algorithm was run in an 'off-line' mode, so the computational expense of the method did not pose a problem.

Gendreau, Guertin, Potvin, Séguin (1998) solve the dynamic pickup and delivery problem with soft time windows by tabu search with ejection chains and adaptive memory. The ejection chain concept guides a building of a neighborhood solution in the following way: one request is taken from a route and moved to another route, forcing a request from that route to move to yet another route, and so on. The chain may be of any length and may be cyclic or not. An approximation of the objective function has been used in the request insertion evaluation. The stop locations of one request were inserted sequentially: the best insertion place for the pickup location was first found, and then the best insertion place for the delivery location was

chosen. The propagation was performed only when the insertion place for a stop location was definitively chosen. The problem of finding the best ejection chain over the current set of routes is modelled as a constrained shortest path problem, solved by a heuristic which is an adaptation of the all-pairs Floyd-Warshall shortest-path algorithm. The adaptive memory concept (Rochat, Taillard, 1995) and the decomposition procedure (Taillard, 1993) are added in order to reach better quality solutions. The method achieves good results on instances with 24 requests per hour (over six hours service period) and on instances with 33 requests per hour (over four hours service period). The authors also report on the parallel implementation (16 processors) of the method.

(Research on the dynamic routing problems, in general, is scarce, but there are several recently published reports Psaraftis (1988); Powell (1991); Gendreau, Potvin (1998) that are focused on issues specific to the problem modelling and to the problem solving within a dynamic environment.)

2.4.4 Role of the capacity constraints

Since large-scale PDPTWC (with more than a thousand requests) has successfully been solved by different heuristic methods, we discuss the application of these methods in solving large-scale PDPTW. The studied PDPTWC problems fall in two categories:

- the problems that deal with transporting goods, and
- the problems that deal with transporting people.

Problem of transporting goods are often characterized by tight capacity constraints. The fairly large load (ranging in some solved instances from 30% to 100% of vehicle capacity) makes very effective usage of clustering (with respect to load) which reduces the problem to several single-vehicle problems that are more easily solvable. Also, the successful column generation methods are suitable, because the number of corresponding columns in the problem is directly proportional to the tightness of the problem constraints.

The problems of *transporting people* falls in one of the categories: dial-a-ride problems (DARP), transportation of handicapped people (HTP), and prisoners transportation. They are all characterized by:

- narrow time windows,
- tight capacity constraints,
- the existence of the maximal riding time for each customer,
- the constraint that the vehicle is not allowed to wait while carrying a customer.

Compared to our PDPTW problem, *the time windows constraints* are much tighter: the most common time windows in transporting people are 10-30 minutes, while the time windows in our PDPTW range from 30 minutes to almost six hours. The customer ride time is bounded by a value that is approximately twice the direct ride time, whereas in our problem, maximal ride times do not exist. The capacity constraints are also tight, especially in problems that deal with transporting persons with special needs, like HTP. Accordingly, we argue that, as in the case of transporting goods, the tight constraints allow more effective usage of the methods based on clustering, and of the methods based on column generation. Furthermore, tight constraints increase the possibility of a vehicle being empty for some period of time, which is utilized in the Dantzig-Wolfe decomposition/column generation method. The periods of time when a vehicle is non-empty are used to divide vehicle route in 'mini-clusters' or 'pieces of work', on which the decomposition is based. In our PDPTW, it is unlikely to expect that a vehicle will be empty for any period of time because there is no maximal ride time and the time windows are wide. It is expected that if either of the PDPTWC constraints are not tight the Dantzig-Wolfe decomposition/column generation method would behave quite differently. This is supported by the authors Dumas, Desrosiers, Gélinas, Solomon (1995) who say that when applied to TSPTW the method "experiences extreme degeneracy difficulties" [pp.367].

2.5 Summary

The methods for solving time-constrained vehicle routing problems have gradually evolved over the years, and nowadays many of the methods can optimally solve fairly large problem instances: 800-node TSPTW by the dynamic programming algorithm (Dumas, Desrosiers, Gélinas, Solomon, 1995), 40-request 1-DARP by the dynamic programming algorithm (Desrosiers, Dumas, Soumis, 1986), 151-trip m-TSPTW by the Dantzig-Wolfe decomposition/column generation method (Desrosiers, Soumis, Desrochers, 1984). The dynamic programming methods have been developed for single-vehicle time-constrained routing problems, while the Dantzig-Wolfe decomposition/column generation method can solve multi-vehicle problems as well. Unfortunately both approaches have been sensitive to the tightness of the time window and capacity constraints. (To make the discussion on solved problems complete, we note that regarding VRP only instances of modest dimensions (around 30 customers) can be solved if no additional constraints or characteristics are attached. For details see surveys Laporte, Nobert (1987); Laporte (1992).)

Despite the success in the development of exact optimization methods, a great number of practical problems would have been left unresolved, due to their size, if it were not for developments in the area of heuristics. There are a large number and a great variety of heuristic methods developed for solving the time-constrained vehicle routing problems: incomplete optimization heuristics, decomposition methods, construction procedures, improvement procedures, and modern heuristics. Incomplete optimization methods include the Dantzig-Wolfe decomposition/column generation method that has been proven to be the best for solving large-scale routing problems with tight capacity constraints and tight time window constraints. The method has succeeded in solving 2500-request HTP instances (Ioachim, Desrosiers, Dumas, Solomon, Villeneuve, 1995). Note also that insertion heuristics, two-phase/three-phase construction heuristics, and local search improvement heuristics, despite their relative simplicity, produce good solutions in complex practical environments.

More recently, metaheuristics have been developed which embed other simple subordinate heuristic and use complex neighborhood search procedures. Tabu search, simulated annealing, constrained logic programming, neural networks, genetic algorithms and evolutionary methods all have been applied to time-constrained routing problems, with success. A great variety of ideas and approaches are incorporated in these methods. A characteristic of these methods is that the quality of the solution depends on the running time of the algorithms. This suggests that if there is enough time, algorithms can reach near optimal or even optimal solutions, as borne out by empirical studies.

We conclude that despite the immense research in the area of vehicle routing very little has been done in the area of PDPTW, especially on the large-scale PDP with relatively wide time windows. Furthermore, the successful methods in solving large-scale PDPTWC utilize the problem characteristics that are not met in PDPTW. Another motivating factor can be the fact that the research on dynamic routing problems is in its early stages although it has a very promising future.

Chapter 3

Problem description, mathematical model, and the solution methodology

3.1 Real-life problem

3.1.1 Couriers

One of the most well-known messengers is Phaedippas, the ancient Greek runner, after whom the marathon event was established. Phaedippas ran from Marathon to Athens to announce the Greek victory against the Persians. As per Knox (1995):

> Marathon [490 BC]
>
> The Athenians had won at Marathon, but they certainly had not destroyed the Persian army, and they knew it. Well before the battle, they had made provision for whatever might happen at Marathon.
>
> Should the Athenians lose, then word must get back quickly to the city and the citizens would abandon Athens, retreating to the Peloponnesus. Should the Greeks win, then word must likewise get back quickly, for the Persian navy was sure to sail

> around Attica and attempt to take the city while it was undefended. In the case of victory, the citizens were to man the walls and make it appear that Athens was strongly defended.
>
> So Miltiades sent his best runner, Phaedippas, to take word back to Athens. He ran the entire distance [approximately 42.2 km]. When he arrived, he gasped out a single word, "victory!" ("nike" [nee-kay] in Greek) and died.
>
> The Persians did indeed sail around Attica, hoping to find the city helpless. When they met with resistance, they hesitated. Not long after, the Greek army arrived. The Persians decided they had had enough of these Greeks, and they sailed home.

The world has changed, but the need for messengers has not. The growth of world population, industrialization, and the growth of telecommunication, computer and information industries have dramatically changed the ways of sending messages. But there is still a need for sending messages in the old manner.

A few hundred years ago, the only way of operating a postal service was to have messengers (postmen) report in the morning to a central office, pickup letters and leave, for the day or the month, to deliver the letters. Several decades ago the situation was much improved - a messenger (postal) company had the luxury of telephones: messengers could call from almost every pickup or delivery location. Thus the central office had the opportunity to send the messenger, from the location where he/she was calling, to a nearby location to pickup some new letter (not known about earlier). Nowadays the situation goes even further: a messenger company can enjoy the advantages of the developments in telecommunication, information and computer technology. Paging systems, two-way paging, cellular phones, Global Positioning Systems (GPSs), Geographical Information Systems (GISs) bring new possibilities to courier companies. Constant communication between a messenger and a central office is in common use, bringing new services to people, businesses and industries. There is a great need for messengers who are available on call to pickup a letter from one location and to deliver it quickly to another location. Such messengers are nowadays called couriers, and the corresponding companies are the courier companies.

3.1.2 Courier company operations

The data displayed in this section represent one-month sample collected in two medium-to-large courier companies situated in Vancouver, BC. These courier companies serve customers with transportation requests that entail picking up a specific load at one location after a specified time, and delivering it to another location before a specified time. A customer places a transportation request by a phone call to a courier company, or through the Internet. About ten telephone operators receive requests. The assignment of the requests to available vehicles, and the vehicle routing are done by three to four dispatchers. The scheduling of a route is done either by a dispatcher or by the driver. The drivers serve the requests, and report to a dispatcher.

Definition 1 *A route is a sequence of stop locations assigned to one vehicle.*

Definition 2 *A schedule of a route is a list of the following times for each location in the route: the scheduled arrival time, the scheduled start of service time, and the scheduled departure time.*

Definition 3 *The process of routing is the process of finding a set of routes.*

Definition 4 *The process of* scheduling *is the process of finding schedules for fixed routes.*

Each courier company serves on the order of thousand requests daily. Very few of the requests (5%) are known in advance, *i.e.*, they are entered the previous day. Almost all requests (about 90%) are without specified pickup time meaning that the load is ready for pickup at the time of the call. Each company offers several types of services with different maximal time that is allowed for serving a whole request (assuming that the request is picked-up at the start of the pickup time window). Requests may be roughly classified into four categories by service type: the six-hour requests, the four-hour requests, the two-hour requests, and one-hour requests. Also available is overnight (next-day-delivery) service, but the percentage of these requests is

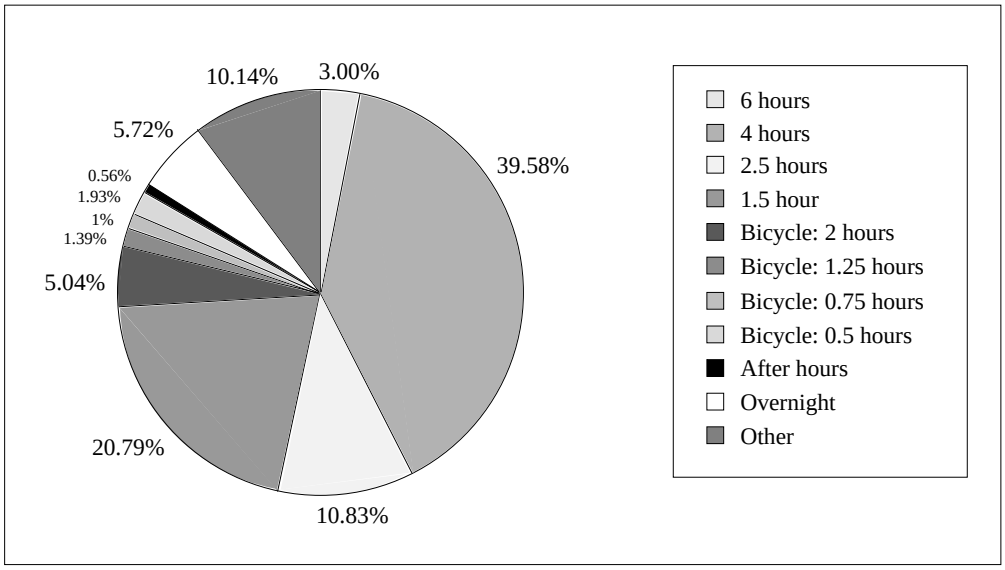


Figure 3.1: The service types of a courier company regarding the time maximal available for serving the transportation request (assuming that the request is picked-up at the start of the pickup time window).

very low - around 5%. Similarly only a small percentage of requests deal with renting a vehicle or with dedicating a vehicle to one customer. For details see Figure 3.1 and Figure 3.2.

The same-day pickup and delivery service poses specific conditions and constraints, of which the most important one is non-transshipment. Transshipment involves moving a load from one vehicle to another while the load is being transported from its pickup location to its delivery location. Courier companies require that the entire request be served by one vehicle (driver), mostly due to commission-based payments that are given to drivers.

The majority of requests correspond to transporting letters and small parcels: more than 80% of the parcels have weights of less than one kilogram, and an additional 4% have weights of less than two kilograms. More than 90% of the requests are for one-piece load, with an additional 7% of the requests consisting of two to five piece loads.

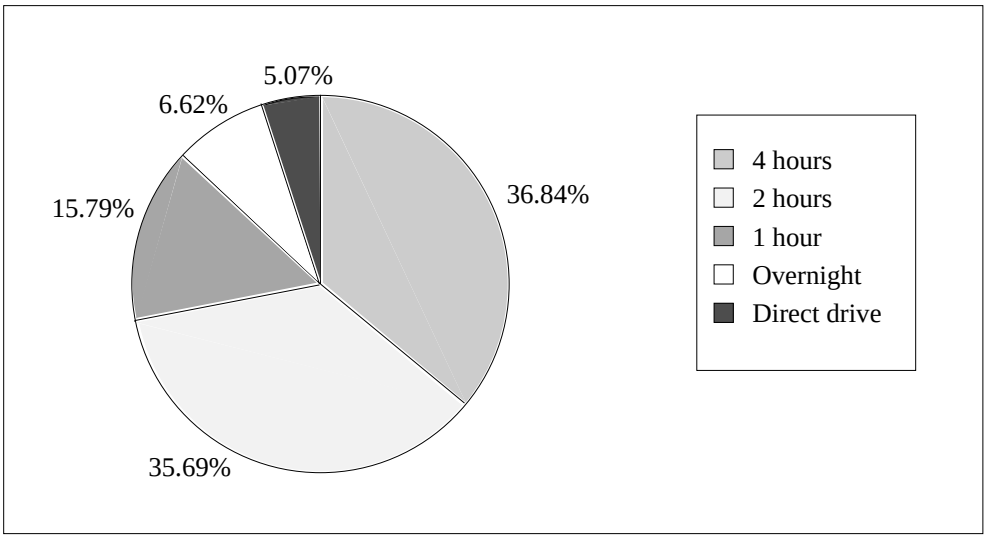


Figure 3.2: The service types of another courier company regarding the time maximal available for serving the transportation request (assuming that the request is picked-up at the start of the pickup time window).

The service period of a courier company is determined by its hours of operation. Normally, the service period lasts from 7:00 to 17:00, with just a few requests being served in early morning hours or late evening hours. Figure 3.3 shows a distribution of the start of the pickup time window of requests during a day, while Figure 3.4 shows a distribution of the actual delivery time of requests during a day at one courier company. Peaks on graphs in the figures are due to entering data manually where customers, drivers and telephone operators tend to report and enter rounded values for time, *i.e.*, instead of 2:02 pm the time 2:00 pm would more often be entered.

The service area of a courier company is a city. About 20% of the requests have both stop locations within the downtown core of the city. Since the downtown area is crowded during peak hours, the courier company employs couriers on bicycles. Bicycles can be driven through

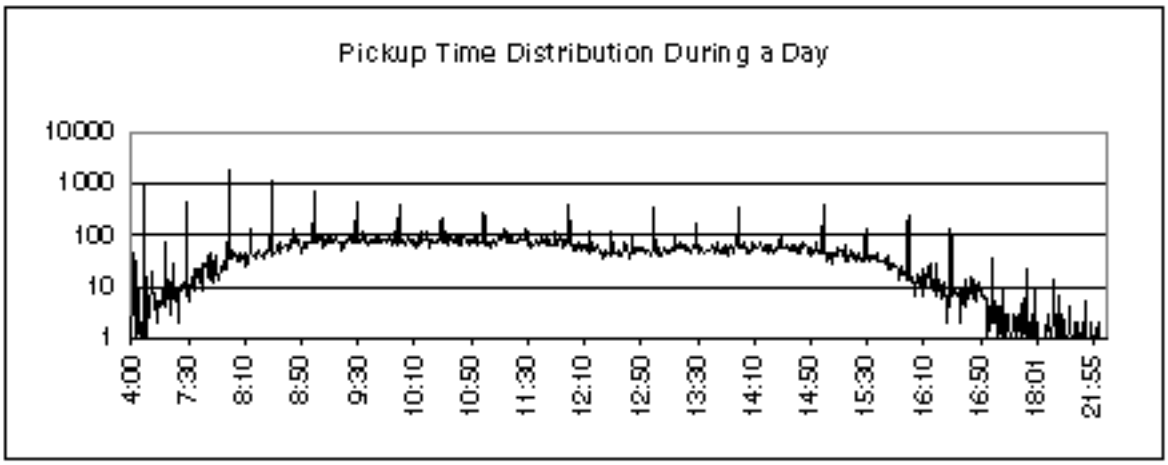


Figure 3.3: The distribution of the requests pickup time, during a day.

city streets as if there are no rush hours or traffic jams, and thus the travel times are not dramatically different during rush-hour and non-rush-hour periods. In addition, the capacity difference between a vehicle and a bicycle is not much of an issue, because a courier company mostly deals with letters and small parcels.

Within the whole city, travel times are subject to change according to the time of the day, the current weather conditions, the time of the year, the current traffic conditions (accidents, holidays), etc. The travel times from one location to another do not have to be the same in both directions. It is also worth noting that the calculation of travel times within a city is different from the calculation of travel times for intercity transportation. The intercity transportation has been the subject of numerous research projects, many of which deal with the travels spanning over large areas and covering several countries or even a continent. Rivers, mountains, lakes do not change travel times much in the context of travelling thousands of kilometers. However, within a city travel times are very sensitive to the topology and geography of the area. The most common approach is to divide a city in to several hundred zones, and precalculate distances between each pair of zones. A detailed description of city zoning and approximate calculation

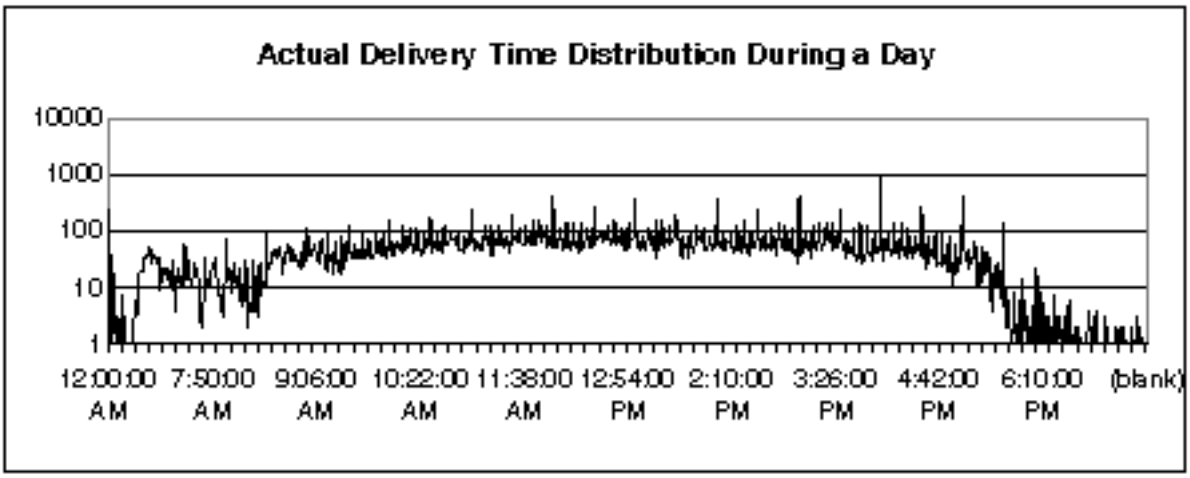


Figure 3.4: The distribution of the actual delivery times, during a day.

of locations distances within a city network is given in Shen, Potvin, Rousseau, Roy (1995). Another possible approach to the calculation of travel times would be to model city areas based on congestion and topology, and to repeatedly solve problem of weighted shortest path, where weights are the travel times that change over time. Yet another approach would be to realize that lakes and rivers represent real obstacles, and to solve the problem known in the computational geometry literature as the shortest paths with obstacles (Lee, Preparata, 1984). On the other hand major roads and highways can be considered as narrow areas that allow faster traffic flows. Together with congested and hilly areas, all those regions of different traffic flows can be modelled either as a weighted region problem (Mitchell, Papadimitriou, 1987), or as a discrete geodesic problem (Mitchell, Mount, Papadimitriou, 1987) addressed in research of algorithmic aspects of robot motion and terrain navigation. Other studies of motion planning in robotics (Schwartz, Sharir, 1988; Chakravarthy, Ghose, 1998) might also be relevant in calculating travel times within a city. Consequently, as the travel time calculation between two locations within a city is a problem by itself, in our research we assume that travel times are known.

In the courier industry, it is common practice that a fleet comprises vehicles owned by drivers under contract. A driver is paid by the courier company according to the type and the number of requests he/she has served. Usually, the amount paid is a percentage (50-65%) of the service price. The courier company does not factor in the vehicle costs. There is no central depot: the drivers (vehicles) can start their day at any location in the city, and there is no obligation to end at the same location or any particular one. A driver's workday lasts as long as there are unserved requests, unless the driver decides to finish for the day. Most drivers do not mind working overtime especially because the after-hours service price is higher (and so is the driver's commission). The courier company is not obliged to take account of lunch breaks. While working hours and lunch breaks are not constraints, it would be better to schedule requests in such a manner that drivers have fewer but longer breaks. The size of the vehicle fleet is a variable. The courier company maintains a list of drivers willing to work, and the company calls drivers as per demand. A medium-to-large courier company handles around 100 vehicles. It is worth mentioning that the variable fleet size can result in situations where some drivers do not get a sufficient number of requests to serve during a day. In practice, this problem is solved by contracting part-time (on-call) drivers that are called if required. Also, the courier company is so reluctant to reject a request that even renting a cab is a possibility, if needed.

The objective of a courier company is to maximize its profits. The company is committed to serve all requests in order to minimize customer dissatisfaction. The number and the nature of requests determine the company's profit, and the company can indirectly influence the increase in the number of requests by serving the current customers effectively and on time. Furthermore, the company can influence the profit by optimizing vehicle routes, *i.e.*, by serving requests using the shortest routes possible which would bring more money in a shorter period of time. Note that the drivers' goal of maximizing earnings is strongly connected to the company's objective because the drivers' salaries are commission-based.

The time windows imposed on locations can be considered as being hard because the service price of a request depends on the service type of the request, *i.e.*, on the time allowed for serving

whole request. If the request is served later than requested, the service price goes down, and so does the profit. Thus, one of the main goals of the courier company is to serve customers on time.

3.2 Model of the real-life problem

3.2.1 Dynamic pickup and delivery problem with time windows

The dispatcher of a courier company solves large-scale dynamic pickup and delivery problem with time windows. The details and assumptions about the problem are given below.

Requests. The requests are the same-day pickup and delivery requests with time windows. The service area is within the city, and the service period is one day. The time windows are closed (each having a start and an end). The time windows are wide, ranging from 30 minutes to almost six hours. (In the literature, time windows of duration 10-30 minutes are considered to be narrow.) The time windows are hard. The loads are by far smaller than vehicle capacities, thus posing no capacity constraints. There are no customer priorities; the transshipment of load is not allowed; and all requests must be served. (An experimental study was carried out on instances with no requests known in advance.)

Information on requests. There is a real-time inflow of requests: new requests are coming-in during the entire service period. A request is completely certain when it is known: the pickup and delivery locations are strictly specified, and the time windows are defined. There are no repeat demands; there are no fixed frequencies of visits; and there are no dependencies among requests.

Vehicle fleet. The size of the fleet is not fixed. There is an initial number of active vehicles at the start of the service period, but additional vehicles are available if some requests cannot

be served by already active vehicles. The vehicles are homogeneous. There is no central depot: vehicles can report to work from any position in town (the vehicle start position), and there is no specified end position for any vehicle.

Crew requirements. The number of drivers is equal to the number of vehicles. There are no minimum or maximum on-duty times, nor, are there over-time payments. Lunch breaks are not factored in.

Other routing and scheduling requirements. Waiting is allowed at any stop location. Each two stop locations are distinct, and the travel time between each pair of two stop locations is strictly positive. The travel times are considered to be known. (In our experimental study the travel times are a linear function of the Euclidean distance between locations.)

The objective function is the minimization of the total route lengths.

A feasible solution consists of a set of open routes and a set of schedules.

To be more precise, the pure dynamic PDPTW is employed to model the dispatching problem of a courier company. (Here the pure dynamic problem is a problem whose future data are assumed not to be predicted, forecasted, or stochastically modelled.) Future requests are not predicted or forecasted herein, because of the uncertainty in the positions of the stop locations, and the uncertainty in the widths and starting times of their corresponding time windows. Similar pure dynamic problem has been used to model dynamic time-constrained routing problems in the majority of the vehicle routing studies: many proposed solution methods optimize over known requests, without attempting to anticipate future customer requests and to use them to improve decision making. An exception is the problem of real-time ambulance relocation solved by an anticipatory heuristic developed by Gendreau, Laporte, Semet (2000).

3.2.2 Mathematical programming formulation of the static problem version

The mathematical programming formulation of our PDPTW problem, derived from the formulation of the general pickup and delivery problem Savelsbergh, Sol (1995), is given in Figure 3.5. Suppose that there are n customers. Let P^+ denote the set of all pickup locations, P^- the set of all delivery locations, and $P = P^+ \cup P^-$. A stop location is labeled i if it is not important to know whether the location is a pickup location or a delivery location. Otherwise, a pickup location is labeled i^+ and its corresponding delivery location is labeled i^-. Set W is a set of vehicles. The starting position of vehicle v is $s(v)$, and set of all starting positions is D^+. The ending position of vehicle v is a delivery location. Set D^- of all ending positions is subset of P^-. Set N includes P and D^+. The time window for stop location i is $[a_i, b_i]$. The earliest time for vehicle v to leave its starting position is $a_{s(v)}$. Labels $d_{i,j}$ and $t_{i,j}$ represent respectively the distance and the travel time between two distinct stop locations i and j.

Two types of variables are used in the mathematical formulation: binary flow variables $X^v_{i,j}$, and time variables T^v_i. The binary flow variable $X^v_{i,j}$ has value 1 if vehicle v travels from node i to node j. Time variable T^v_i is the time when vehicle v starts serving location i.

Without the loss of generality it is assumed that the service time s_i at each stop location i is equal 0. (Service time of a location is the time that vehicle spends serving the location.) Each PDPTW with positive service times, can be transformed to the problem with 0 service times by the following transformation: $t^\theta_{i,j} = t_{i,j} + s_i$, where $t^\theta_{i,j}$ is the travel time between locations i and j in the 0-service-time problem. In other words, the travel time between a pair of locations is increased by the service time at the starting location. Unfortunately this transformation could cause a symmetric problem to be transformed into an asymmetric one.

The mathematical programming formulation of a pickup and delivery problem with time windows is a non-linear integer program. The problem objective is the minimization of the total route lengths, *i.e.*, the minimization of the sum of the total distance travelled.

$$\text{min} \quad \sum_{v\in W}\sum_{i,j\in N} d_{i,j}X^v_{i,j} \tag{1}$$

$$\text{subject to} \quad \sum_{i^+\in P^+} X^v_{s(v),i^+} = 1, \qquad v\in W \tag{2}$$

$$\sum_{j\in N} X^v_{j,s(v)} = 0, \qquad v\in W \tag{3}$$

$$\sum_{v\in W}\sum_{j\in P} X^v_{i^+,j} = 1, \qquad i^+\in P^+ \tag{4}$$

$$\sum_{v\in W}\sum_{j\in P} X^v_{j,i^-} = 1, \qquad i^-\in P^- \tag{5}$$

$$\sum_{j\in P} X^v_{i^+,j} - \sum_{j\in P} X^v_{j,i^-} = 0, \qquad i^+\in P^+,\ v\in W \tag{6}$$

$$\sum_{i^-\in P^-}\sum_{j\in N} X^v_{i^-,j} = (\sum_{i\in N}\sum_{j\in N} X^v_{i,j})/2 - 1, \qquad v\in W \tag{7}$$

$$\sum_{j\in P} X^v_{i,j} - \sum_{j\in P} X^v_{j,i} = 0, \qquad i\in P,\ i\notin D^-,\ v\in W \tag{8}$$

$$X^v_{s(v),i^+} = 1 \ \Rightarrow\ T^v_{s(v)} + t^v_{s(v),i^+} \le T^v_{i^+}, \qquad i^+\in P^+,\ v\in W \tag{9}$$

$$X^v_{i,j} = 1 \ \Rightarrow\ T^v_i + t^v_{i,j} \le T^v_j, \qquad i,j\in P,\ v\in W \tag{10}$$

$$T^v_{i^+} + t^v_{i^+,i^-} \le T^v_{i^-}, \qquad i^+\in P^+,\ v\in W \tag{11}$$

$$a_{s(v)} \le T^v_{s(v)}, \qquad v\in W \tag{12}$$

$$a_i \le T^v_i \le b_i, \qquad i\in P,\ v\in W \tag{13}$$

$$X^v_{i,j}\in\{0,1\}, \qquad i,j\in N,\ v\in W \tag{14}$$

Figure 3.5: Mathematical formulation of the pickup and delivery problem with hard time windows and without a depot.

The constraints of this model can be explained as follows:

(2) There is a unique pickup location immediately after the start vehicle position.

(3) There are no stop locations preceding the start vehicle position.

(4) Each pickup location is followed by one stop location.

(5) Each delivery location is preceded by one stop location.

(6) Pairing constraints.

(7) Each vehicle route finishes with one delivery location.

(8) The number of vehicles that leaves stop location i is equal to the number of vehicles that reaches location i, except for the last route location.

(9)-(10) Compatibility between routes and schedules.

(11) Precedence constraints.

(12)-(13) Time window constraints.

(14) Binary flow variables constraints.

Constraint (9) $X^v_{i,j} = 1 \Rightarrow T^v_i + t^v_{i,j} \leq T^v_j$ can be written as quadratic constraint $X^v_{i,j}(T^v_i + t^v_{i,j} - T^v_j) \leq 0$, and it can be linearized by $T^v_i + t^v_{i,j} - T^v_j \leq (1 - X^v_{i,j})M^v_{i,j}$, where $M^v_{i,j}$ is a large constant greater than $\max(b_i + t^v_{i,j} - a_j, 0)$.

3.3 Overview of the solution methodology

For solving a pure dynamic optimization problem, whose future unknown data are not predicted or stochastically modelled, the most appropriate method would be one that solves the problem at times when data (or a set of data) becomes known. Unfortunately, decisions have to often be taken and acted upon before all data are known. And this happens repeatedly during the problem service period. Therefore, the most suitable approach for solving the pure dynamic pickup and delivery problem is an on-line method, a method that repeatedly solves the problem during the problem service period.

3.3.1 Subproblems in on-line problem solving

In an on-line problem solving of the dynamic pickup and delivery problem with time windows, one can observe three subproblems:

(1) *the problem of pre-optimization:* the problem of making an initial routing and scheduling, if some requests are known in advance;

(2) *the on-line problem:* the problem of assigning new requests when vehicles are already driving according to planned routes, routing, and scheduling;

(3) *the problem of periodic solution improvement (usually called re-optimization):* the problem of periodically improving the routes by applying an improvement procedure.

We mainly concentrate on solving the on-line problem (2) dealing with the appearance of new requests and their insertion in routes. Nevertheless, we have touched the problem of solution improvement (3) by incorporating a tabu search procedure in the on-line system for solving the dynamic PDPTW. This tabu search is an intra-route improvement procedure.

In the thesis we do not address the problem of pre-optimization (1), the main reason being that this subproblem is a static problem and as such has been addressed in literature. Another reason lies in the fact that the percentage of requests known in advance is very low as per studies made in courier companies. However, the solution of the pre-optimization problem can have an impact on solving other subproblems of the problem. The pre-optimization problem can be solved either by (a) trying to serve known requests by few vehicles, making good and tight routes, or (b) distributing requests among many vehicles with the aim to better spread the routes over the service zone. In the case where a very low percentage of requests is known in advance, both approaches might work. Approach (a) could produce much shorter initial routes. In addition, applying approach (a) will leave the rest of the vehicles to deal with (almost) all future requests as being a separate on-line problem. On the other hand, if the percentage of requests known in advance is high, one has to be careful. Building good and tight routes by a pre-optimization procedure can result in a high number of vehicles that are almost fully dedicated to serving requests known in advance. Therefore, just a few vehicles would be available to cover a whole service zone and all future requests.

3.3.2 Our on-line system

In our research we have used algorithm engineering that is concerned with the design, theoretical and experimental analysis, and engineering and tuning of algorithms. This emerging discipline in the algorithmic community addresses issues of realistic algorithm performance by carefully combining traditional theoretical methods together with thorough experimental investigations. Hence, we have designed an on-line system for simulating and solving the dynamic PDPTW.

3.3.2.1 System layout

Our on-line system for the dynamic PDPTW comprises two parts, a part that simulates events of the dynamic PDPTW, and a part that solves the dynamic PDPTW. The on-line system is introduced through diagrams of the unified modelling language (UML) used in software engineering. A general layout of a courier company operation is given in Figure 3.6 (the use case view diagram of UML). A customer places a request that is transferred to a dispatcher who decides about assigning the request to a vehicle, about the vehicle routing, and about the route scheduling. The routes and the schedules are sent to drivers. (Sometimes, in practice, routes and schedules are not given to drivers, but only requests assigned. Nevertheless, finding shorter routes is in the interests of the company and the driver (see discussion on page 41).) The drivers report their current positions and status of requests which are assigned to them (whether the requests are unserved, picked-up, or served).

The operation of a courier company is translated into the on-line system depicted in Figure 3.7. When the problem service period starts, the simulator of request appearance begins imitating customers by generating requests, or by reading requests from a file that contains a pre-generated problem instance. The requests are transferred to the component called router that decides about assigning the requests to vehicles, and about routing. Because of the possibility that the vehicles has to serve some known assigned requests before other new requests become known, the routing has to be followed by the route scheduling, and it is done by the

scheduler. The routes and the schedules are then sent to the simulators representing vehicles. The routes might also be shown on the graphical user interface. The vehicle simulators move vehicles according to the routes and the schedules, and take care about vehicle statuses. The vehicle simulators also report on request statuses. The timer simulates the passing of real time. This part of the system is self-sufficient and it can solve the simulated dynamic PDPTW problem. In addition to that, if there is available time the system will try to improve routes by the improvement component (meaning that this on-line system can solve the dynamic problem with and without the usage of the improvement component).

Figure 3.7 also shows that the system components can be divided on two fronts. There are components that simulate real-life events, and components that solve the problem. On the other side, some components work truly on-line, and some work off-line (in the background solving a static subproblem of the problem at hand). More details about the components are given in Figure 3.8.

The sequence diagram of the time-dependent interaction between the components is given in Figure 3.9. Vertical lines represent the components in time. Time passes top-down in the figure. Horizontal lines represent events, or messages about finished procedures. The arrow on a horizontal line points toward a component that is affected by the event. The events and messages are ordered in time (top-down).

The most sophisticated components of the system are the router, the scheduler, and the improvement component. A version of router, a greedy on-line insertion heuristic is given in Chapter 4. The scheduling and some of possible implementations are given in Chapter 5. All other parts of the on-line system are discussed in the rest of this chapter, to a certain extent.

3.3.2.2 The vehicle and its states

A vehicle in the on-line system can be either active or inactive. A given number of vehicles are active at the beginning of the problem service period. Other vehicles in the fleet become active

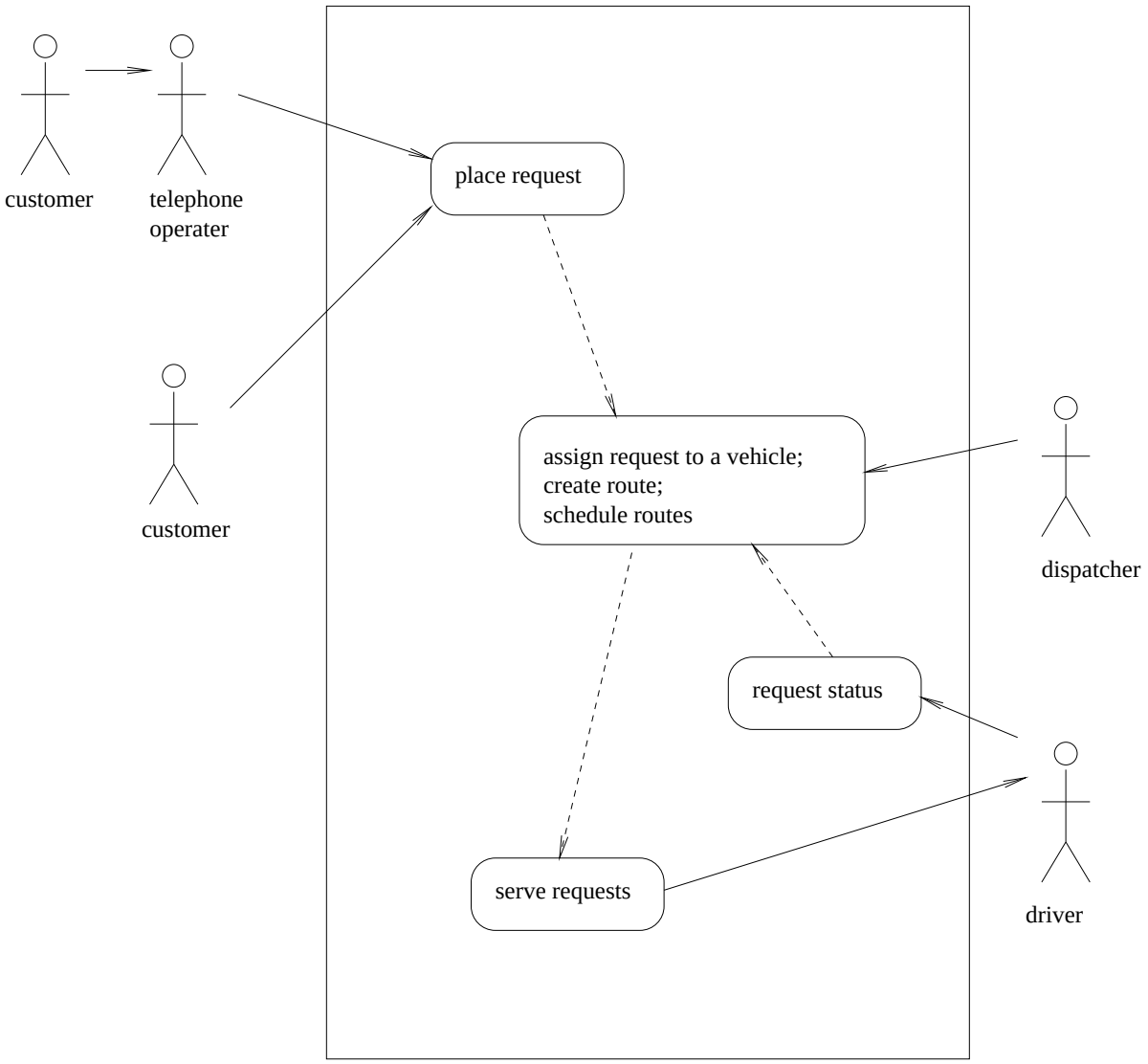


Figure 3.6: A courier company.

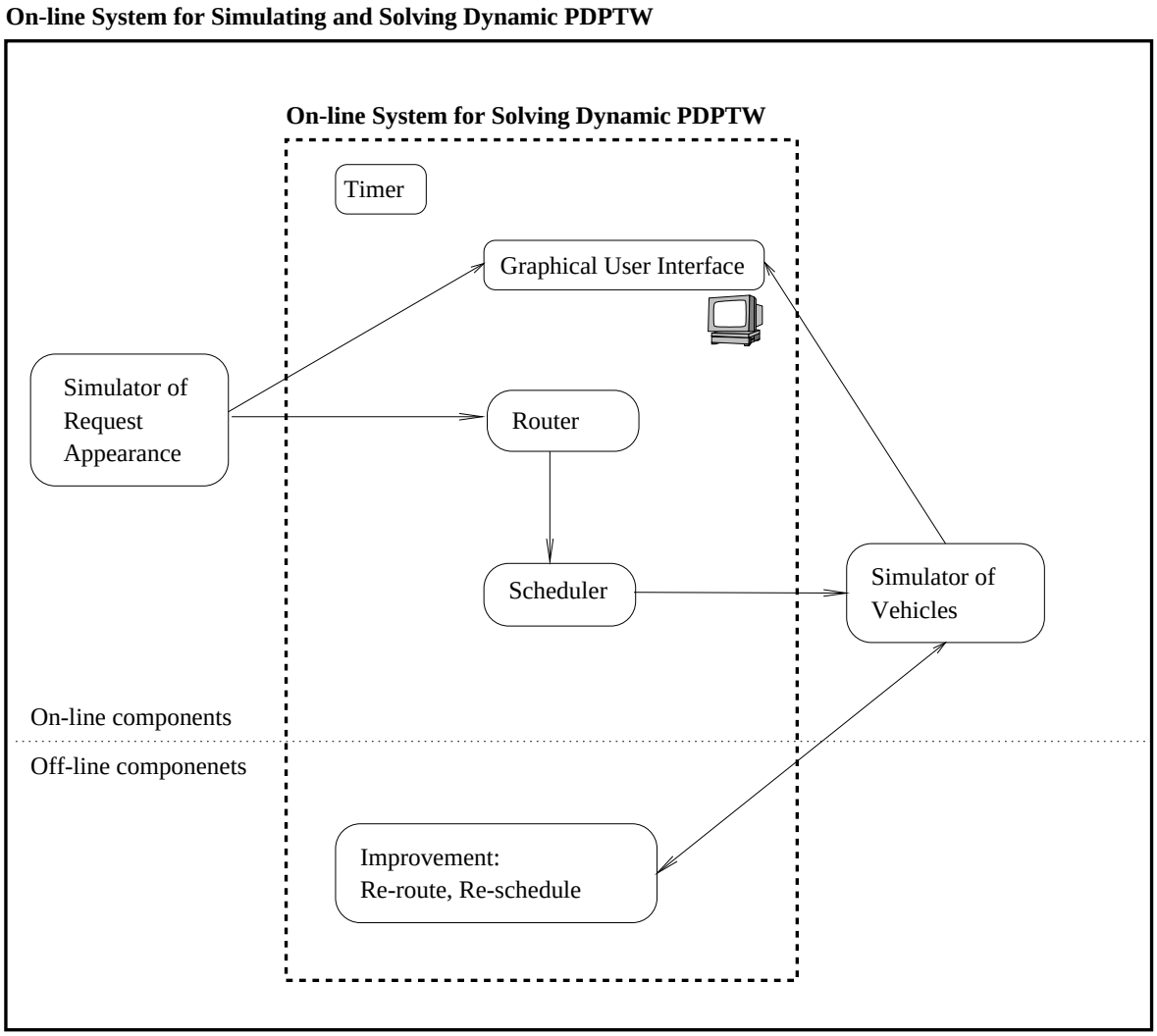


Figure 3.7: On-line system for simulating and solving the dynamic PDPTW.

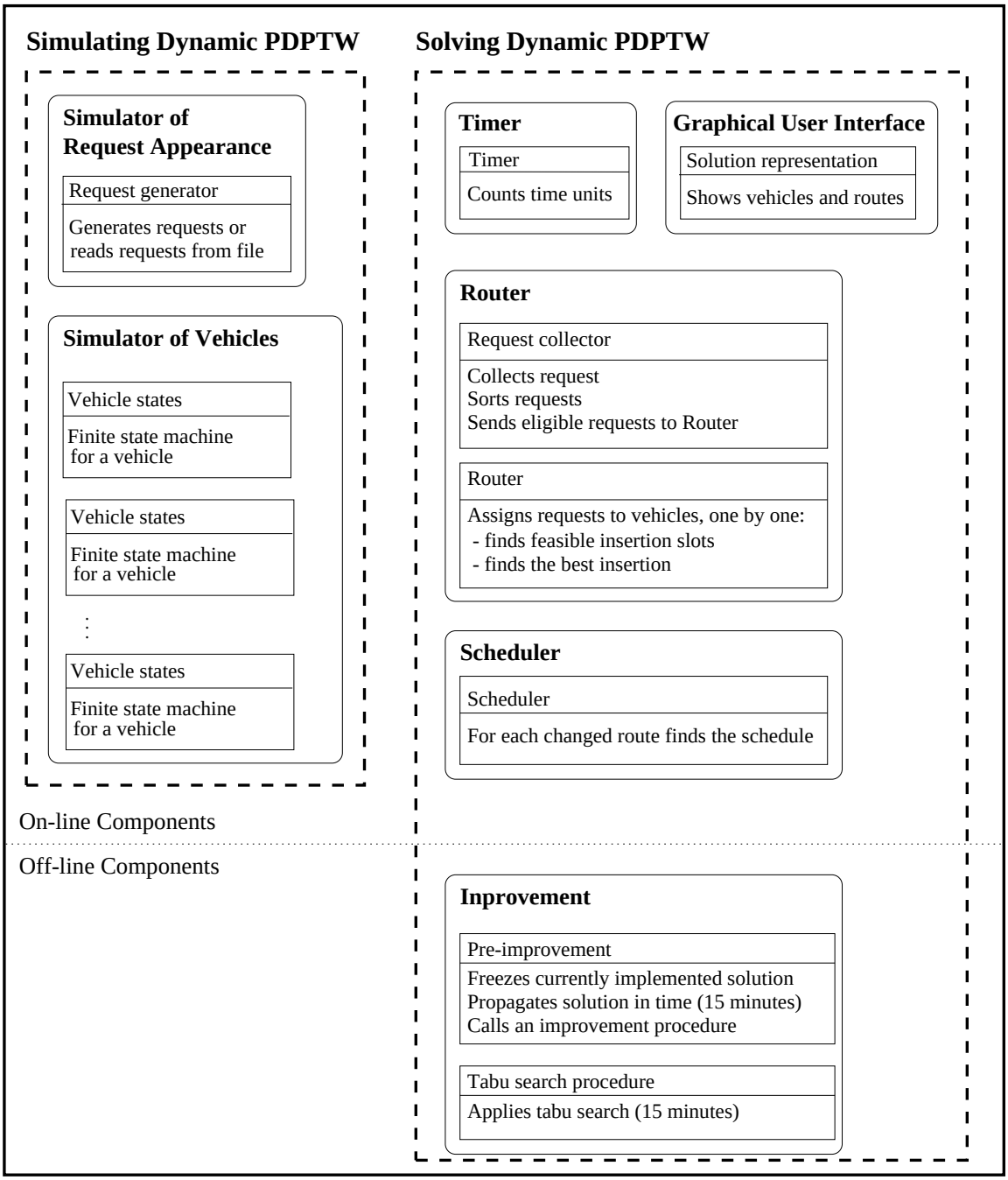


Figure 3.8: More details about components of the on-line system for simulating and solving a dynamic vehicle routing problem.

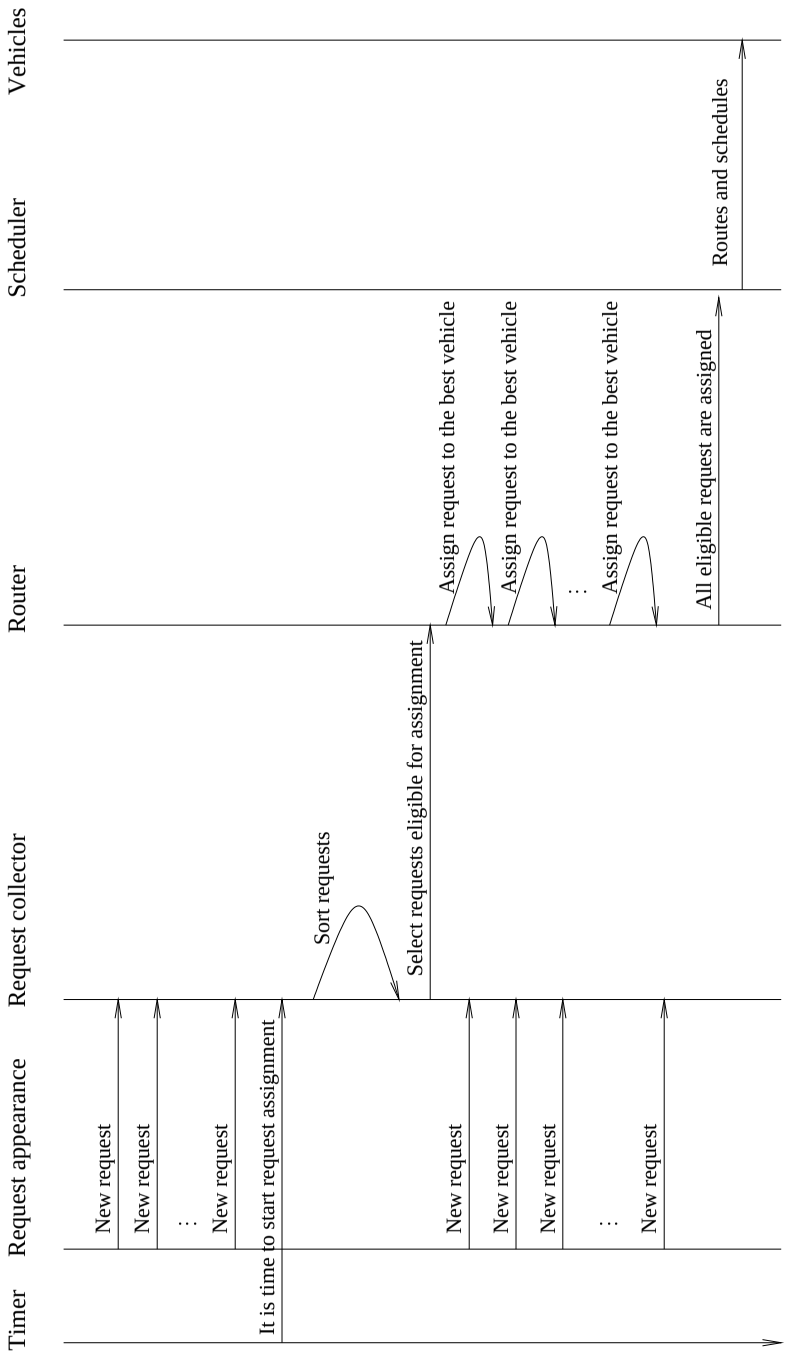


Figure 3.9: Sequence diagram of interactions between system components.

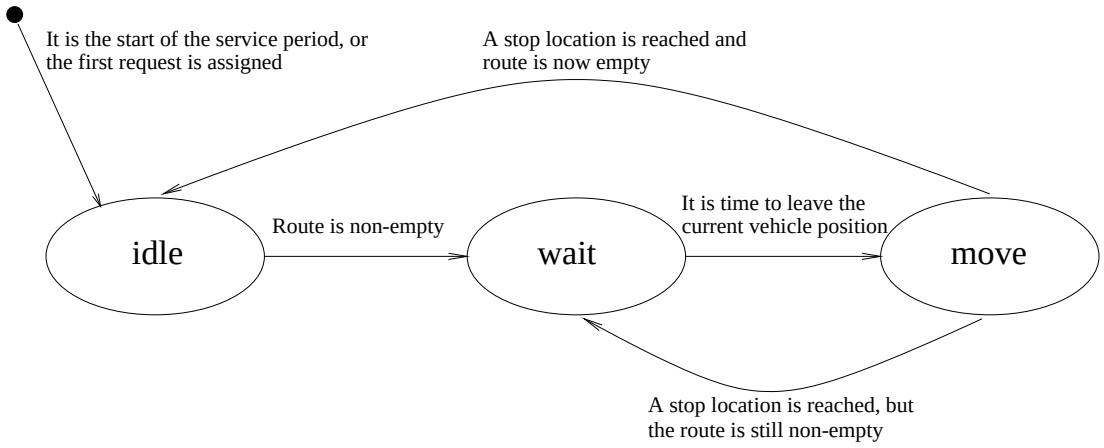


Figure 3.10: Finite state machine representing the vehicle states.

when the first request is assigned to them (this can only happen if currently active vehicles cannot feasibly serve the new request). All active vehicles remain active till the end of the problem service period. An active vehicle is simulated by its movements along the route, and by the change of its states. The vehicle states are:

- moving state - when the vehicle moves toward a stop location,
- waiting state - when the vehicle is waiting for the scheduled departure time,
- idle state - when the vehicle route is empty.

The finite state machine representing a vehicle is given in Figure 3.10. The initial state of a vehicle is the idle state. An initially active vehicle is in the idle state from the very beginning of the service period. Any other vehicle gets into the idle state (immediately followed by waiting or moving state) when the first request is assigned to the vehicle. Since the vehicle route contains only unvisited locations, a vehicle is in the moving state or in the waiting state, while the vehicle route is non-empty. When the vehicle route becomes empty, the vehicle gets into the idle state. (An active vehicle in the waiting or in the idle state is still an active vehicle.)

3.3.2.3 Graphical representation of a solution

We provide here a graphical representation of the PDPTW solution used in the graphical user interface of our on-line system. Figure 3.11 represents a simple solution of a routing problem, comprising one route. The figure depicts three windows, of which one represents the plane view, and other two represent time. The biggest (middle) window represent the route $\{1, 2, 3, 4, 5, 6, 7, 8, 9\}$ in the plane. The big bold numbers close to the edges are the travel times between the stop locations. The other two smaller windows show the route in time: the route in the plane is first projected on one axis, and than the projection is shown in time. The upper window gives the route in time after it was projected on x-axis, while the left window gives the route in time after it was projected on y-axis. In these two windows the position of a location number indicates when the location was visited. The three windows are actually the three projections in the three planes of a solution represented in three-dimensional space. The first two coordinates are the (x,y) plane coordinates, and the third coordinate is time.

Figure 3.12 gives the same route in the case when all locations have time windows. For the sake of simplicity almost all time windows are of the equal size. Waiting occurs only once before serving location 5, as shown in the figure by the dashed line.

3.3.2.4 Improvement component

Pre-improvement steps. The improvement component of our on-line system (see Subsection 3.3.2) works off-line, and improves the current solution of the dynamic PDPTW as if it is a static solution. Thus, the initial work to be done before the improvement procedure is applied consists of:

(1) 'freezing' the currently implemented solution,

(2) propagating it in time (for the time that we want to allow for running of the improvement procedure).

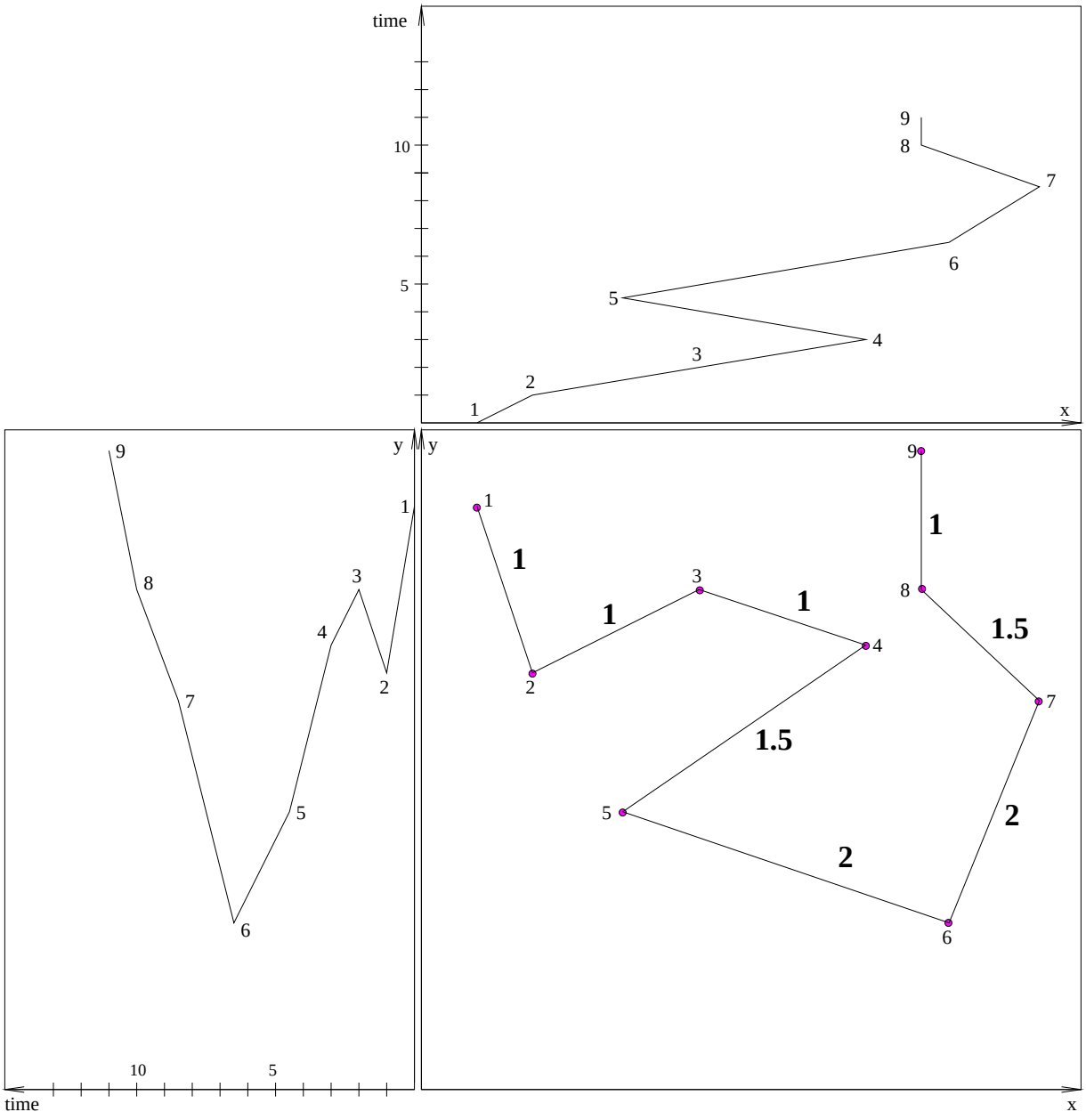


Figure 3.11: Graphical representation of one route.

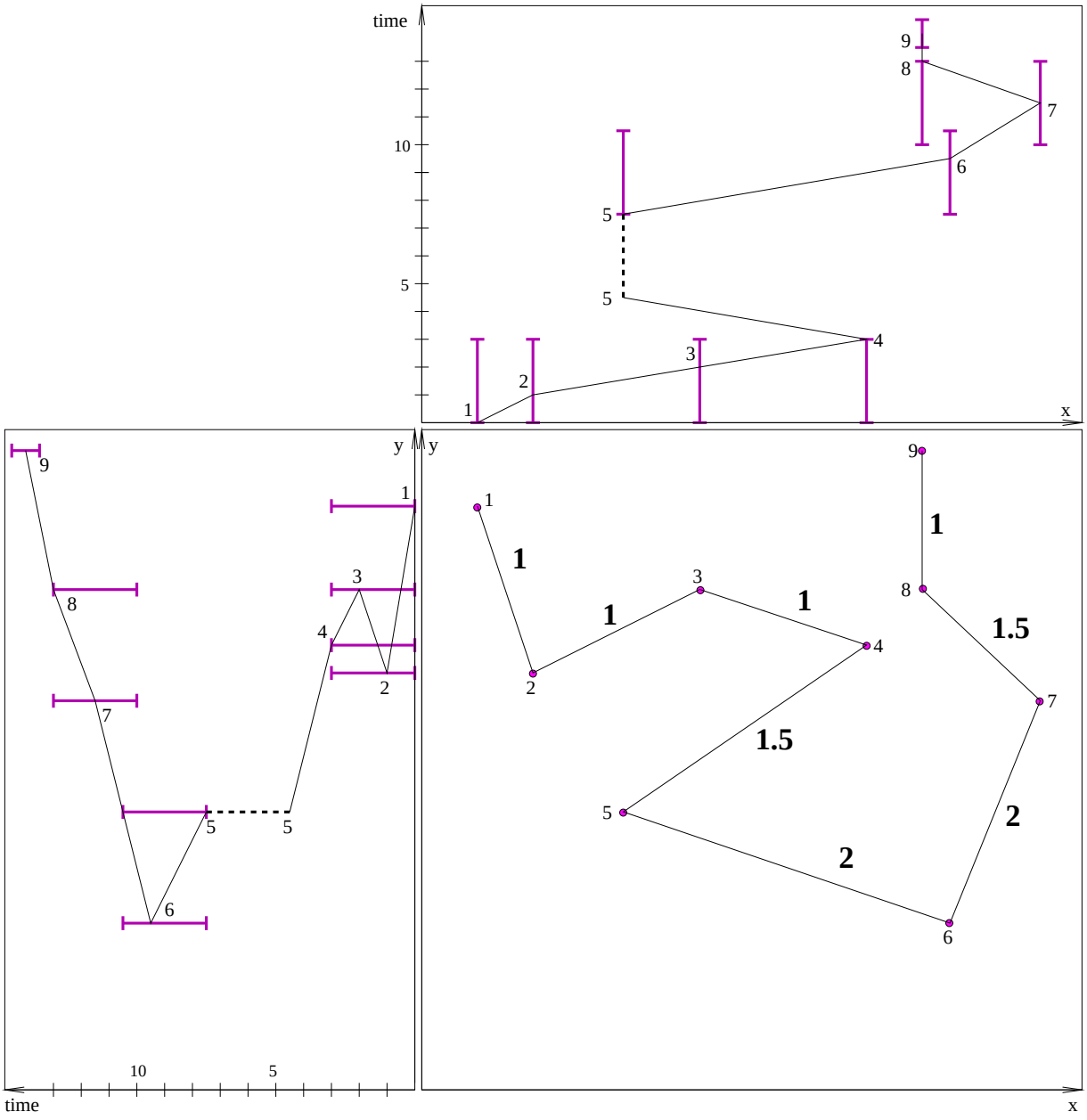


Figure 3.12: Graphical representation of one route. Each location has a time window.

To make things less complex, in an otherwise complex dynamic environment and on-line system, we recommend the postponing of any new request insertion during the running time of the improvement procedure.

Tabu search. Tabu search improvement procedure is included in our on-line system in order to thoroughly test waiting strategies (see Chapter 5) for solving the scheduling problem. Our preliminary empirical study (on the system without an improvement procedure) has shown that different waiting strategies achieved different quality of solutions. We wanted to check whether similar differences in solutions (in respect to the waiting strategies) is carried forward when more complex and more successful solution methods are used. Since the tabu search is one of the best solution methods for the dynamic PDPTW (as reported in literature), we have implemented a simple tabu search procedure. We anticipate that similar solution differences (in respect to the waiting strategies, as achieved in our empirical study on the system with tabu search improvement procedure) would be obtained when more complex tabu search procedures are used.

The tabu search is an improvement heuristic introduced by Glover (1989, 1990). As described in Garcia, Potvin, Rousseau (1994) at each step of a tabu search procedure, the neighborhood of the current solution is explored, and the best solution in the neighborhood is selected as the new current solution. As opposed to pure local search techniques, a tabu search procedure does not stop when no improvement is possible. The best solution in the neighborhood is selected as the current solution, even if it is worse than the current solution. This strategy allows the method to escape from a local optima and, as a consequence, to explore a larger fraction of the search space.

The simplest form of tabu search can be described by the following scheme taken from Gendreau, Hertz, Laporte, Stan (1998):

Step 1: Start from a solution S.

Step 2: The best solution found is equal to the current solution: $S^* = S$.

Step 3: While the stopping criteria is not met do:

Step 3.1: Generate a neighborhood of S through non-tabu moves.

Step 3.2: Select the best neighborhood solution S'.

Step 3.3: If S' is better than S^*, set $S^* = S'$.

Step 3.4: Set the current solution $S = S'$.

Step 4: Output solution S^*.

To prevent cycling, the tabu search procedure forbids repeated moves into recently selected solutions. These recently selected (visited) solutions are inserted into a list called 'tabu list'. Instead of keeping the entire solutions visited in the tabu list, a more common implementation is to keep some characteristics of the solutions. For example, the list can contain an element such as: "vehicle v serves request i", therefore forbidding vehicle v to serve request i for a certain number of iterations.

Sometimes a tabu move can be overridden, if an overall best solution is produced by doing so. Explicitly, each element of the tabu list can be checked to see if its violation produces an overall best solution, better than S^*. For example, if serving request i by vehicle v will achieve a solution better than S^*, this tabu move can be overridden.

We have chosen to implement a simplified variation of the tabu search method proposed by Gendreau, Guertin, Potvin, Séguin (1998) for solving the dynamic pickup and delivery problem with soft time windows. We use similar definition of the neighborhood based on ejection chains: a request (with both its stop locations) 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 chain may be of any length smaller than the number of routes, and it may be cyclic or not. A greedy heuristic is used to extend the current ejection chain. (The best non-tabu coupled insertion

of the request is chosen. Details about the coupled insertion of a request are given page 86.) Solution feasibility is maintained at all times, despite the fact that the best results in solving vehicle routing problems have been achieved when the tabu search procedure was allowed to explore both, feasible and infeasible solutions, as neighbors. The moves into infeasible solutions have been shown as being effective for moving out of local optima, and for exploring different areas of the search space. Unfortunately, in the case of time-constrained routing problems (like PDPTW, VRPTW and TSPTW) it is highly recommended that the solution method maintain, at all times, the feasibility of solutions because it is quite difficult to regain feasible solution after an infeasible move (due to the time windows constraints) (as cited in Garcia, Potvin, Rousseau (1994); Gendreau, Hertz, Laporte, Stan (1998)). Thus, our tabu search procedure moves into the best feasible neighboring non-tabu solution. One visited solution is tabu for a random number of iterations. The random number is chosen from the interval $[5, 10]$ (Gendreau, Laporte, Potvin, 1997), which generates a relatively short tabu list. After certain time the tabu search procedure terminates, and the best visited solution is implemented as the current solution.

Since the tabu search implemented is an intra-route improvement procedure, one has to be careful when applying such a procedure in practice. Dispatchers are reluctant to take a request already assigned to a driver in order to assign it to another driver. The problem can be resolved by informing drivers only about requests that have to be picked up in the near future, and permitting in the procedure only reassignment of other requests (not known to drivers). We believe that reassignment will be used more often in practice, because the advancements in telecommunication industry (the two-way pagers, the global positioning systems) have made reassignment possible and easy.

Chapter 4

On-line insertion heuristic

4.1 Introduction

This chapter describes an on-line greedy insertion heuristic. The heuristic has been used within the router component of our on-line system (as described in Subsection 3.3.2). The heuristic solves the problem of assigning requests to vehicles, and the problem of routing vehicles. Section 4.2 gives a general scheme of the heuristic with a detailed description of several criteria. Testing the feasibility of inserting a request into a route is discussed in Section 4.3. New graphs, called precedence graphs, are introduced in Section 4.4. The precedence graphs are used for checking the feasibility of a vehicle serving a new request. In the case of m-TSPTW, the precedence graphs can be used for finding a lower and an upper bound on the minimum number of vehicles needed to serve all locations. Section 4.5 analyzes the best request insertions with its complexity.

4.1.1 A route of a dynamic time-constrained routing problem

A *route* is a sequence of distinct locations $(1, 2, ..., i-1, i, i+1, ..., n)$ that are scheduled to be served by one vehicle. Each location is either a pickup location or a delivery location. A pickup location is labeled by i^+ when it needs to be distinguished from a delivery location i^-. The

other labels that will be used are:

$[a_i, b_i]$ - the time window of location i; a_i is release time and b_i is deadline.

$t_{i,j}$ - the travel time between locations i and j

s_i - the service time or duration of service at location i

A_i - scheduled arrival time at location i

$\underline{A_i}$ - the earliest arrival time at location i

$\overline{A_i}$ - the latest arrival time at location i

D_i - scheduled departure time from location i

$\underline{D_i}$ - the earliest departure time from location i

$\overline{D_i}$ - the latest departure time from location i

T_i - the start time of service at location i

$\underline{T_i}$ - the earliest start time of service at location i

Note that the earliest arrival time at location i is the earliest arrival time at the location when it is a part of some route. It is not the earliest possible arrival time at this location (before it is assigned to some route). Similarly, the other earliest and latest times of a location are the times determined only when the location belongs to a route.

4.2 On-line insertion procedure

This section gives general scheme of the on-line greedy insertion heuristic implemented in the router of the on-line system. The problem of assigning requests to vehicles and the problem of routing vehicles are solved simultaneously by finding the best insertion of a new request. The requests are inserted into routes by a *sequential decision making heuristic*. (The term was first used by Dror, Laporte, Trudeau (1989).) A request is inserted so as to minimize the increase in route lengths. There are several criteria that define details of the procedure: criterion for

start assigning of unassigned requests, criterion of request eligibility for assignment at a specific moment in time, criterion for sorting of unassigned requests, criterion for selecting vehicles which will be evaluated for serving an unassigned request.

On-line insertion heuristic:

Step 1: Wait until the criterion for request insertion is satisfied.

Step 2: Select unassigned requests eligible to be assigned to routes.

Step 3 (Optional): Sort eligible requests.

Step 4: For each eligible request (i^+, i^-) do:

Step 4.1: Find active vehicles that can feasibly serve request (i^+, i^-).

Step 4.2: Find the best insertion of request (i^+, i^-). Insert request (i^+, i^-).

Step 4.3: If no active vehicle can serve request (i^+, i^-), assign request to a new vehicle.

Step 5: Make a schedule for each vehicle whose route was altered in Step 4.

Step 6: Goto Step 1.

There are many versions of this on-line insertion heuristic, depending mostly on the chosen criteria and the measures. The criteria and the measures employed herein are presented in the second part of this section.

4.2.1 Request collecting, sorting and eligibility for assignment

New requests are being collected during the whole service period. Whenever the criterion for request insertion is satisfied, the eligible requests are selected from unassigned requests, sorted and assigned to vehicles.

The criterion for starting request insertion is:

- the appearance of a new request, or
- the passing of a certain time since the last request insertion.

The latter criterion sets the on-line algorithm to resolve insertion of eligible requests regularly, every k minutes. Unassigned requests eligible for insertion are either:

- all unassigned requests, or
- all unassigned requests with an impending pickup deadline (the end of the pickup time window is close to the current time).

The eligible requests can be sorted by the following criteria:

- the request deadline in nondecreasing order (the request deadline is the end of the delivery time window), or
- the difficulty of the request (so that the most difficult requests are assigned first).

A request is more difficult to serve if its time windows are narrow (the deadline of the delivery time window is close to the release time of the pickup time window), and the distance between the pickup location and delivery location is large. Thus, a difficulty degree can be equal to the 'free time' h_i of the request:

$$h_i = (b_{i^-} - a_{i^+}) - t^*_{i^+,i^-}, \tag{4.1}$$

where b_{i^-} is the end of the delivery time window, a_{i^+} is the beginning of the pickup time window, and $t^*_{i^+,i^-}$ is the shortest travel time between the stop locations i^+ and i^-. A shorter 'free time' implies the request is more difficult to serve. Therefore, we sort requests in nondecreasing order of the request 'free time'.

Also there is a preprocessing step, called the *time window restrictions*, done on each new request. The restrictions affect the end time of a pickup time window, and the start time of a delivery time window. The first restriction is:

$$b_{i^+} = \min(b_{i^+}, b_{i^-} - t^*_{i^+,i^-}), \tag{4.2}$$

and it possibly reduces the size of the pickup time window $[a_{i^+}, b_{i^+}]$ by incorporating the minimal travel time $t^*_{i^-,i^+}$ into the pickup time window. The restriction is based on the fact that a vehicle that is due to be at i^- before b_{i^-} has to leave i^+ at $b_{i^-} - t^*_{i^+,i^-}$ at the latest. The restriction for the start time of a delivery time window is:

$$a_{i^-} = \max(a_{i^-}, a_{i^+} + t^*_{i^+,i^-}). \tag{4.3}$$

This restriction possibly reduces the size of the delivery time window $[a_{i^-}, b_{i^-}]$ by incorporating the minimal travel time $t^*_{i^+,i^-}$ into the delivery time window. The restriction is based on the fact that a vehicle that is due to be at i^+ after a_{i^+} can visit i^- at $a_{i^+} + t^*_{i^+,i^-}$ at the earliest.

4.2.2 Restricted evaluation of vehicles

Solving large-scale dynamic PDPTW can be characterized by the limited time allowed for decision making. If one can predict which subset of vehicles would be the most appropriate for serving a new request, one can evaluate only vehicles belonging to this subset. If a feasible insertion is found the new request can be assigned. Such a procedure represents a *restricted evaluation of vehicles*, and it can be applied when there is not enough time for evaluating all the vehicles. In this section we describe a prediction of the closest vehicle being the best vehicle for serving a new request. The distance between a request and a vehicle is the sum of two distances: the distance between the pickup location and the vehicle route, and the distance between the delivery location and the vehicle route. The distance between a location and a route is the distance between the location and the route service zones. (The service zones are described in Section 5.4.)

Inserting a request using a restricted evaluation of vehicles:

Step 1: Sort all vehicles by the distance from a new request.

Step 2: Partition the set of vehicles into subsets according to distance. Sort subsets. (Details are given below the algorithm.)

Step 3: Take the first subset of vehicles.

Step 4: For each vehicle from this subset, find the best insertion of the new request.

Step 5: If there is a feasible insertion, assign the request to the best vehicle. Stop.

Step 6: If there are no feasible insertions, take the next subset of vehicles. Go to Step 4.

The set of vehicles is partitioned into subsets in two ways:

(1) the first subset contains vehicles that are at distance no greater than (constant) D from the request, the second subset contains the vehicles that are at a distance ranging from D to $D + E$ (E is also a constant) from the request, and so on; or

(2) the first 10% of the closest vehicles are in the first subset, the second 10% of the closest vehicles are in the second subset, and so on.

4.2.3 Complexity of an on-line heuristic implementation

In implementation of an on-line insertion algorithm, many difficulties arose that may not arise in the implementation of off-line insertion algorithms. One of the problems is that decisions about a new location insertion has to be made while the states and positions of vehicles are changing. (The insertion procedure takes time, too.) Each active vehicle is either waiting, moving or idling. When a vehicle is moving it might be impossible to schedule the new location as the first location of the vehicle route. On the safe side, good practice would be to avoid inserting

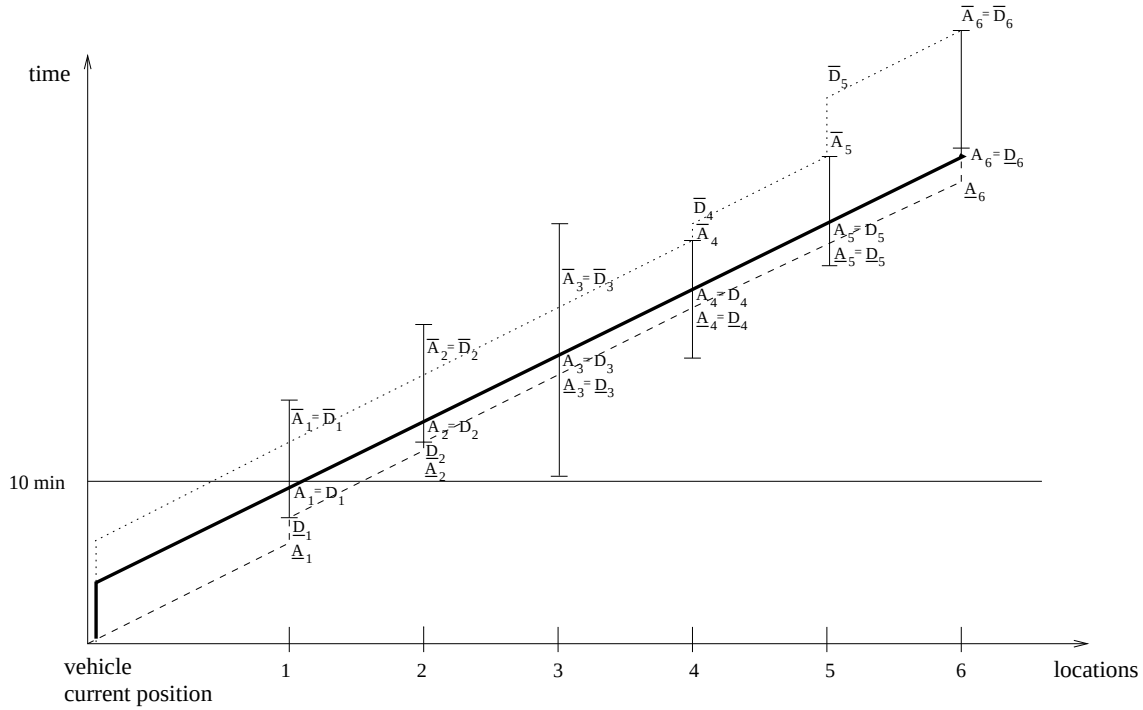


Figure 4.1: The assignment of a new location to a vehicle that drives along the route r: $\{1, 2, 3, \ldots, k\}$: During the assignment the vehicle is waiting for the time to leave the current location and to drive towards the location 1. Figure shows that insertion of a new location is allowed after location 2 whose scheduled departure time is more than ten minutes away from the current time.

the new location before a location (in the route) whose scheduled departure time is close to the current time. As in Figure 4.1 the insertion of a new location is allowed after location 2.

4.3 Feasible request insertion

4.3.1 Testing feasibility

Feasibility of inserting a request into a route depends on the time window constraints and the travel time constraints. This section introduces several feasibility tests. Two of these tests can be used to eliminate vehicles that cannot feasibly serve the request. There is one test that is sufficient for proving that the vehicle can feasibly serve the request, that is the upper-bound-request test. Also introduced are some test that generates a subset of the route locations after which the insertion can be feasible. After this point we have used enumeration over this restricted set of the route locations to evaluate whether the request can feasibly be served by the vehicle.

We discuss insertion of request (i^+, i^-) into the route of vehicle v. Label by V the set of all unserved route locations of vehicle v, the two stop locations i^+ and i^-, and the current position of vehicle v.

Testing feasibility of inserting a request (i^+, i^-) into the route of vehicle v:

The direct-path test: Check if it is feasible for vehicle v to serve request (i^+, i^-) by directly travelling from its current position to pickup location i^+, and then directly travelling to delivery location i^-. If this test fails, vehicle v cannot feasibly serve request (i^+, i^-). On the other hand, the test success is not sufficient for proving that vehicle v can feasibly serve request (i^+, i^-). The complexity of the direct-path test is $O(1)$.

Upper bound on the number of vehicles: Find an upper bound on the minimum number of vehicles needed to serve all locations in set V. If the upper bound is one, vehicle v can

feasibly serve request (i^+, i^-). The complexity of this test depends on the complexity of the algorithm used for finding an upper bound.

Lower bound on the number of vehicles: Find a lower bound on the minimum number of vehicles needed to serve all locations in set V. If the lower bound exceeds one, vehicle v cannot feasibly serve request (i^+, i^-). The lower bound equal one, is not sufficient for proving that vehicle v can feasibly serve request (i^+, i^-). The complexity of this test depends on the complexity of the algorithm used for finding a lower bound.

The restricted connected set of prospective route locations: Separately, for the pickup location and for the delivery location of request (i^+, i^-) do the following: Find the first route location f after which the new stop location can be inserted, and find the last route location l before which the new stop location can be inserted. (Location f for the pickup location i^+ is the last location j in the route for which $b_j \leq a_{i^+}$. Location l for the pickup location i^+ is the first location j in the route for which $a_j \geq b_{i^+}$.) This test is based only on the time windows constraints. If there is no location between f and l, vehicle v cannot feasibly serve request (i^+, i^-). The existence of a route location between f and l does not guarantee that vehicle v can feasibly serve request (i^+, i^-). The complexity of this step is $O(|V|)$, or $O(\lg |V|)$ if the times are kept in sorted order.

Enumeration for finding route locations after which it is feasible to insert request (i^+, i^-):

Separate request's stop locations insertion: Find all route locations between f and l, after which the new location can be inserted, by considering the time window constraints and the travel time constraints. This set of route locations will be called the restricted set or prospective route locations. Label the restricted set after which it is feasible to insert i^+ by F_{i^+}, and the restricted set after which it is feasible to insert i^- by F_{i^-}. If any of

the two sets, F_{i^+} and F_{i^-}, is empty, vehicle v cannot feasibly serve request (i^+, i^-). The complexity of this test is $O(|V|)$.

Coupled request's stop locations insertion: Check each pair of locations from $F_{i^+} \times F_{i^-}$ for the feasibility of the insertion of i^+ and i^-. The increase in route length could be calculated throughout, so that the best insertion could be registered.

The tests just described can be used in design of a gradual feasibility testing procedure. The gradual feasibility testing would allow testing feasibility of inserting a request into a route in steps such that the first tests are less complex than the last, and such that even though some steps may not eliminate the need for further checking of feasibility, they can be used as effective filters to reduce the number of complete feasibility checks required to obtain a feasible insertion. One version of gradual feasibility testing procedure is given in Figure 4.2. Experimental results on a similar gradual feasibility testing procedure are given in Section 7.4. In these experiments the direct-path test discarded 11.93% of vehicles, and the lower bound test discarded additional 44.88% of vehicles. The building of the restricted connected set, and the test of the separate location insertion discarded an additional 27.13% of vehicles. Therefore, the enumerative exploration of route locations (after which it is feasible to insert stop locations of a new request) were done on less than 20% of vehicles.

4.4 Bounds on vehicles, and precedence graphs

One version of the upper-bound test and the lower-bound test of the previous section is presented, by introducing precedence graphs of PDPTW. These precedence graphs can be used for checking whether a new request can be served by a vehicle. The same precedence graphs can be built for m-TSPTW, and used for finding bounds on the number of vehicles needed for serving locations of m-TSPTW. (When dealing with PDPTW it is assumed that the time window reductions (Formula 4.2 and Formula 4.3) have been performed.)

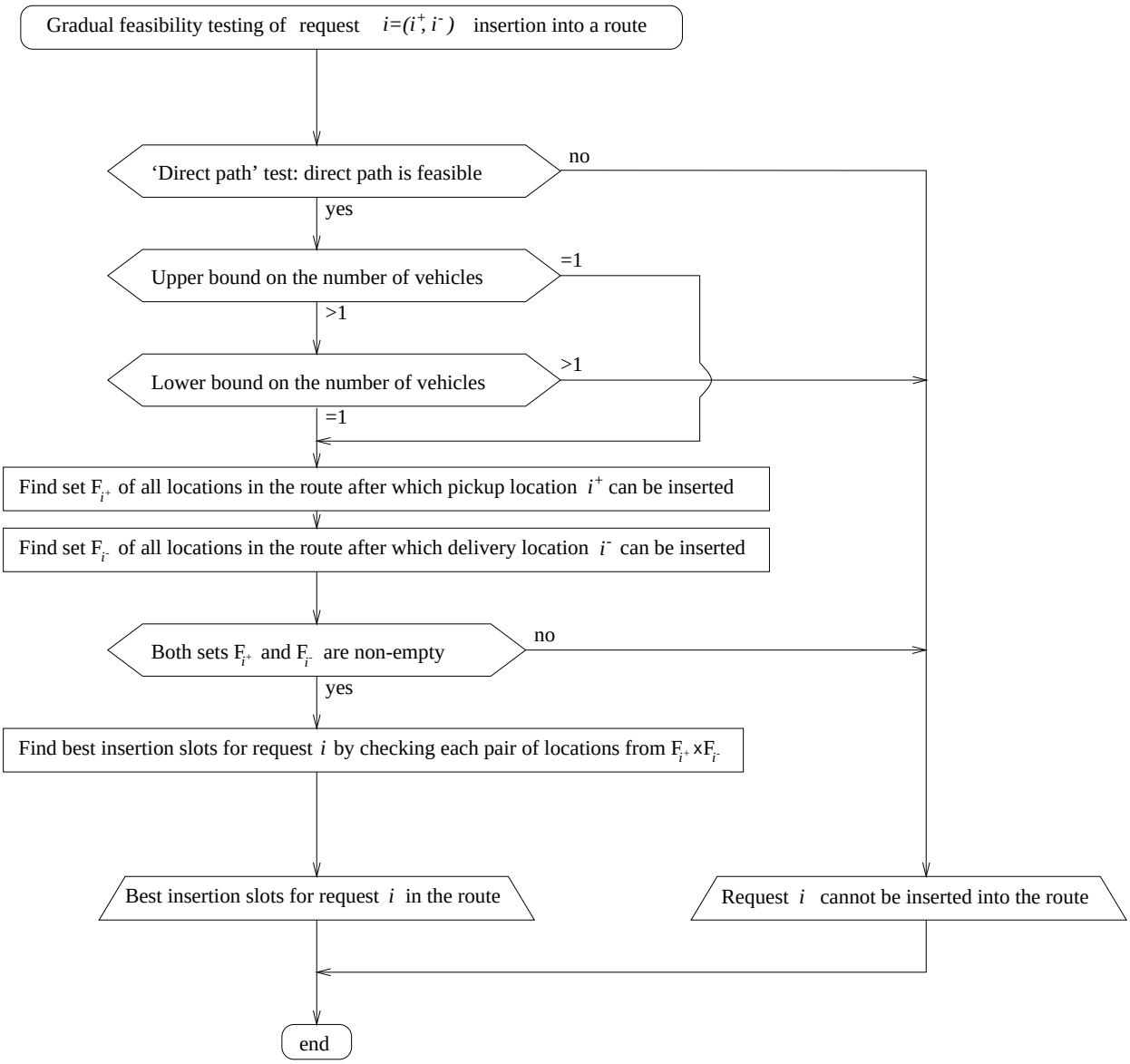


Figure 4.2: A gradual testing of feasibility for the request $i = (i^+, i^-)$ insertion into a route.

4.4.1 Precedence graphs of time-constrained routing problem without capacity constraints

Suppose that $G = (V, E)$ is a graph of a time-constrained routing problem without capacity constraints. Set V contains stop locations and set E contains edges that represent roads between the stop location pairs. Each two stop locations are distinct. Function $t : E \rightarrow R^+$ maps each edge to a strictly positive value that represents travel time between the two corresponding locations. Each stop location i has a corresponding time window $[a_i, b_i]$, which defines the time frame in which i has to be visited. Graph $G^* = (V, E^*)$ is built, where $E^* = \{(i, j)|$ there is a path in G between i and $j\}$. Define function $t^* : E^* \rightarrow R^+$ by which the shortest travel time between two nodes is associated with the corresponding edge. Note that if the problem has k depots, graph G can have at most k components for the existence of a feasible solution to the problem. In graph G^* each component is a complete subgraph.

Graph G^* is equal to G if (1) all components of G are complete subgraphs and if (2) the triangular inequality is satisfied in G. Otherwise, graph G^* can be constructed from G by solving the all-pairs shortest path problem. The all-pairs shortest path problem can be solved by applying Dijkstra's algorithm $|V|$ times (Cormen, Leiserson, Rivest, 1993). A linear-array implementation of the priority queue, gives $O(|V|^3)$ running time. The binary-heap implementation of the priority queue yields a running time of $O(|V|\,|E| \lg |V|)$, which is an improvement if graph G is sparse. Alternatively, the priority queue with a Fibonacci heap can be implemented, yielding a running time of $O(|V|^2 \lg |V| + |V|\,|E|)$.

Now, we construct a *precedence graph* that is a directed graph which mirrors the precedence relations among stop locations. The time window constraints and the travel times are taken into account. We present two types of precedence graphs that can be built from graph G^*.

Definition 5 *The start-time precedence graph, $G_a = (V, E_a)$, of graph $G = (V, E)$ is a digraph of precedence relations among stop locations V with regard to the start time of their time windows,*

i.e.,

$$E_a = \{(i,j) \in (V \times V) \mid a_i + t^*_{i,j} \leq b_j\}.$$

The label G_a is due to the widespread use of label a for the start of a time window. Note that the existence of $(i,j) \in E$ is not mandatory for the existence of $(i,j) \in E_a$, but the existence of $(i,j) \in E^*$ is necessary.

Set E_a has been known in literature as the set of admissible edges.

Definition 6 *The end-time precedence graph, $G_b = (V, E_b)$, of graph $G = (V, E)$ is a digraph of precedence relation among stop locations V with regard to the end time of their time windows, i.e.,*

$$E_b = \{(i,j) \in (V \times V) \mid b_i + t^*_{i,j} \leq b_j\}.$$

The label G_b is due to the widespread use of label b for the end of a time window. Note that the existence of $(i,j) \in E$ is not mandatory for the existence of $(i,j) \in E_b$, but the existence of $(i,j) \in E^*$ is necessary. It is clear that $E_b \subset E_a$.

An example is given in Figures 4.3 and 4.4. Locations of a TSPTW instance are given by graph $G = (V, E)$ whose nodes are $V = \{1, 2, 3, 4, 5\}$. The time windows are: $[a_1, b_1] = [1, 2]$, $[a_2, b_2] = [1, 3]$, $[a_3, b_3] = [2, 3]$, $[a_4, b_4] = [3, 4]$, $[a_5, b_5] = [0, 0]$, and the matrix of the shortest travel times is:

$$D = \begin{bmatrix} 0 & 2 & 3 & 1 & 1 \\ 2 & 0 & 1 & 3 & 3 \\ 3 & 1 & 0 & 2 & 3 \\ 1 & 3 & 2 & 0 & 1 \\ 1 & 3 & 3 & 1 & 0 \end{bmatrix}$$

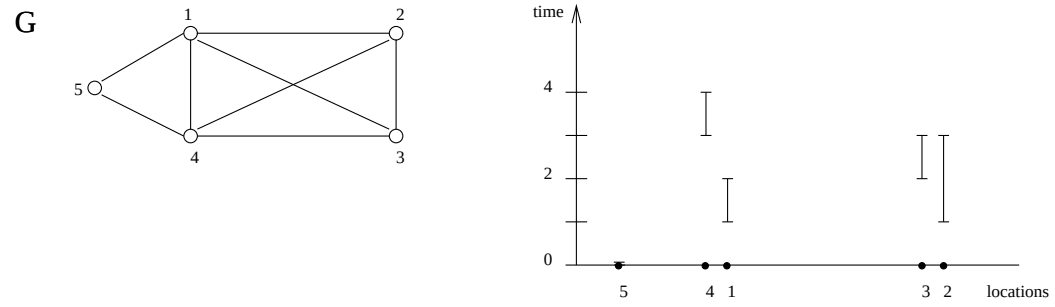


Figure 4.3: Graph G of a time-constrained routing problem, and the time windows of the locations.

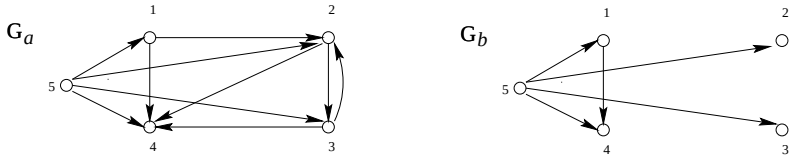


Figure 4.4: The start-time precedence graph G_a, and the end-time precedence graph G_b.

4.4.2 Precedence graphs of PDPTW

In this subsection we restrict our discussion to precedence graphs of PDPTW, and their characteristics in the respect to one PDPTW request. Assume that the time window restrictions (see page 64) have been performed. Consider request i and its two stop locations (i^+, i^-). It is evident that edge (i^+, i^-) is in E_a, because otherwise the request i would be infeasible to serve.

Lemma 1 *The end-time precedence graph G_b contains the edge (i^+, i^-).*

Proof: For each request i it is true (by 4.2):

$$b_{i^-} - t^*_{i^+,i^-} \geq b_{i^+} \quad \Rightarrow \quad b_{i^+} + t^*_{i^+,i^-} \leq b_{i^-} \quad \Rightarrow \quad (i^+, i^-) \in E_b. \qquad \square$$

Lemma 2 *The end-time precedence graph G_b does not contain edge (i^-, i^+).*

Proof: Under assumption $0 < t^*_{i^+,i^-}$, for each request i it is true (by 4.2):

$$
\begin{aligned}
& b_{i^-} - t^*_{i^+,i^-} \geq b_{i^+} \\
& b_{i^-} + t^*_{i^+,i^-} \geq b_{i^+} + 2t^*_{i^+,i^-} \\
& b_{i^-} + t^*_{i^+,i^-} > b_{i^+} \\
& \Rightarrow (i^-, i^+) \notin E_b.
\end{aligned}
$$

□

Theorem 1 *The PDPTW precedence constraints are satisfied in any path of G_b precedence graph.*

Proof: If the travel times are positive this theorem is a corollary of Lemma 1 and Lemma 2. □

Lemma 3 *The start-time precedence graph G_a can contain the edge (i^-, i^+) of some request i.*

Proof: For each i it is true (by 4.3):

$$
\begin{aligned}
& a_{i^-} + t^*_{i^-,i^+} \geq a_{i^+} + t^*_{i^+,i^-} + t^*_{i^-,i^+} \quad \Rightarrow \\
& a_{i^-} + t^*_{i^-,i^+} > b_{i^+} \qquad \text{only if} \quad b_{i^+} - a_{i^+} < t^*_{i^+,i^-} + t^*_{i^-,i^+} \quad \Rightarrow \\
& (i^-, i^+) \notin E_a \qquad\quad \text{only if} \quad b_{i^+} - a_{i^+} < t^*_{i^-,i^+} + t^*_{i^+,i^-}.
\end{aligned}
$$

□

Theorem 2 *PDPTW precedence constraints are not necessarily satisfied in all paths of G_a precedence graph.*

Proof: This theorem is a corollary of Lemma 3. □

4.4.3 Starting vehicle positions

If the time-constrained routing problem (without capacity constraints) has a depot, or if the current vehicle position is considered as a depot, the start-time precedence graph can mirror tighter precedence relations between the locations. The procedure of building the start-time precedence graph G_a starts with finding the earliest arrival time $\underline{A}_i$ at each location i assuming that vehicle v leaves the depot at the earliest possible time $a_{s(v)}$, *i.e.*, $\underline{A}_i = a_{s(v)} + t^*_{s(v),i}$. Graph G_a contains only edges starting at the depot, and ending at the locations. Afterward, for each

location i outgoing edges are to be drawn only toward a location j that can be reached before the end of its time window, assuming that location i was left at the earliest departure time $\underline{D}_i = \max(\underline{A}_i, a_i)$

$$E_a = \{(s(v), i) \text{ where } i \in V \mid \underline{A}_i \leq b_i\} \quad \cup \quad \{(i, j) \in (V \times V) \mid \underline{D}_i + t^*_{i,j} \leq b_j\}.$$

Since $t^*_{i,j}$ is the shortest travel time from location i to location j, and the $t^*_{s(v),i}$ is the shortest travel time from start vehicle position to location i, no further iterations would decrease the earliest arrival times at locations (nor would any edges be deleted in G_a).

If the problem has more depots (or no depot, in which case the starting vehicle positions could be considered as separate depots), the building of the start-time precedence graph will be similar. The earliest arrival time at a location will be the earliest arrival time from the closest depot.

In the example given in Figures 4.3 and 4.4, the stop location 5 acts like a depot because its time window is very narrow, and precedes all other time windows. If we consider it as a depot (or starting vehicle position), we can make the location precedence relation tighter, and build smaller start-time precedence graph (as given in Figure 4.5) than the one shown in Figure 4.4.

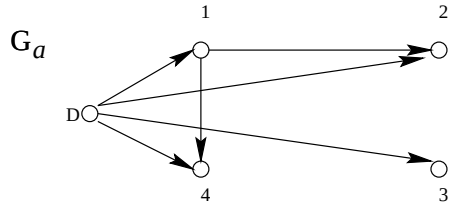


Figure 4.5: Tighter precedence relation and the corresponding start-time precedence graph G_a.

When the depots, or the ending vehicle positions, have to be reached at the end of the service period, before a certain time $e(v)$, the end-time precedence relations can be tightened. The latest departure times for all locations will be calculated first, and then the admissible edges

would be drawn by

$$E_b = \{(j, e(v)) \text{ where } j \in V \mid e(v) - t^*_{j,e(v)} = \overline{D}_i \geq a_j\} \qquad \cup$$

$$\{(i,j) \in (V \times V) \mid b_i + t^*_{i,j} \leq \min(b_j, \overline{D}_j) = \overline{A}_j)\}.$$

4.4.4 m-TSPTW: Bounds on the number of vehicles

This subsection describes how the precedence graphs can be used for finding bounds of the minimum number of vehicles needed to serve all m-TSPTW locations. Denote the minimum number of vehicles needed to serve all locations by v_{min}, and denote a lower bound by $\underline{v_{min}}$, and an upper bound by $\overline{v_{min}}$.

4.4.4.1 Number of vehicles needed to serve locations at one path of precedence graph

Suppose that i and j are two locations of m-TSPTW. It is evident that the two locations can be served by one vehicle iff (i, j) is in E_a. (Note that if waiting is not allowed the statement does not hold.)

Theorem 3 *For serving all m-TSPTW locations that belong to one path of graph G_a more than one vehicle may be needed.*

Proof: Each location in a path of G_a can surely be reached from its predecessor (on the path) if the predecessor is left at the start of its time window. Therefore, to ensure that all locations in one path are served by one vehicle the vehicle should leave each location at the start of its time window. This might not be possible because $a_i + t^*_{i,j}$ might be greater than a_j, for two locations i and j on the path. This implies that more than one vehicle might be needed to serve all locations on the path. □

Theorem 4 *All locations on a path of graph G_b can be served by one vehicle, i.e., the upper bound on the number of vehicles needed to serve all locations that belong to a path in G_b is one.*

Proof: Existence of path L in G_b means that each node in L can be reached from its predecessor if the predecessor was left at the end of its time window, *i.e.*, even if $\underline{D}_i = b_i$. Since all locations has to be served within their time windows, all locations along one G_b path can be served by one vehicle. □

4.4.4.2 A lower and an upper bound

The problem of finding vehicle bounds for m-TSPTW corresponds to the problem of covering all nodes of a precedence graph by the minimum number of paths. We will refer to this problem as the *minimum cover by paths* problem (MCP). The minimum-cover-by-paths (MCP) is the problem of finding the minimum number of directed paths in graph G that covers all nodes of G. Assume that graph $G = (V, E)$ contains all m-TSPTW locations, excluding depot(s) (or vehicle starting and ending positions).

Theorem 5 *The minimum cover by paths of the start-time precedence digraph $G_a(G)$ gives a lower bound $\underline{v_{min}}(G)$ on the minimum number of vehicles needed to serve all locations in G.*

Proof: The minimum cover by paths is a lower bound for the number of vehicle needed to serve all locations since: (a) algorithm gives the minimum number of paths to cover all locations and (b) all locations in one path are considered to be served by one vehicle (Theorem 3 on page 77). □

Theorem 6 *The minimum cover by paths of the end-time precedence graph $G_b(G)$ gives an upper bound $\overline{v_{min}}(G)$ on the minimum number of vehicles needed to serve all locations in G.*

Proof: Any feasible solution of covering-by-paths problem gives a number of vehicles that can be used to serve all locations, because all nodes of a path in G_b can be served by one vehicle.

The minimum cover-by-paths gives the smallest number of vehicles to serve all locations. (The bound is upper because better can be done when some of the locations are left before the end of their time windows.) □

The introduced bounds do not consider starting positions of vehicles. In practical traveling salesman problems where the starting positions of vehicles are important, the bounds can be useful under the following circumstances:

Post-evaluation for a dynamic m-TSPTW: At the end of the problem service period the vehicle bounds can be calculated in order to evaluate the possible best value for the minimum number of vehicles needed to serve all customers. The minimum number of vehicles needed would have been within the bounds, if we knew all requests in advance, and if we had positioned vehicles at certain starting locations.

Stochastic m-TSPTW: If the future requests can be modelled stochastically, we can find good starting vehicle positions.

An on-line algorithm: By finding vehicle bounds for serving unassigned requests, good starting vehicle positions can be determined and used to reroute vehicles in such a way that they converge to these positions, before even assigning those unassigned requests.

4.4.5 Minimum cover by paths

In this subsection we prove (Theorem 7 on page 80) that if the precedence graph of graph G is acyclic the problem of minimum covering by paths is equivalent to the problem of decomposition by chains from the theory of partially ordered sets. The decomposition-by-chains problem is addressed by Dilworth's theorem (Trotter, 1992) A method, proposed by Ford, Fulkerson (1962), for solving the problem on graph $G = (V, E)$ is based on solving the maximum bipartite matching problem on the graph $B = (V, V; A)$ where the set of edges A comprises edge $(i, j) \in A$ iff $(i, j) \in E$. The problem is solvable in polynomial time.

4.4.5.1 Partially ordered sets and chain decomposition

Definition 7 (Robinson, Foulds (1980)) *A* binary relation *defined over a set of elements V is a pair (V, ρ) where $\rho \subseteq V \times V$ is a set of ordered pairs of elements from V. A binary relation is a partial order iff it is:*

Reflexive	$(\forall x \in V)$	$x \rho x$
Antisymmetric	$(\forall x, y \in V)$	$(x \rho y \wedge y \rho x) \Rightarrow x = y$
Transitive	$(\forall x, y, z \in V)$	$(x \rho y \wedge y \rho z) \Rightarrow x \rho z$

Two elements x, y are said to be comparable if $x \rho y$.

Definition 8 (Robinson, Foulds (1980)) *A* chain *is a subset $Y \subset V$ whose every two elements are comparable.*

Definition 9 (Robinson, Foulds (1980)) *An* antichain *is a subset $Y \subset V$ whose no two distinct elements are comparable.*

Theorem 1 (Robinson, Foulds (1980)) *Every acyclic directed graph uniquely determines a partially ordered set.*

Theorem 2 (Robinson, Foulds (1980)) *Dilworth's theorem: The minimum number of chains (totally ordered sets) which cover a partially ordered set is equal to the maximum size of an antichain (set of incomparable elements).*

Theorem 7 *For an acyclic, transitive graph, the problem of finding a minimum cover by paths is equivalent to the problem of finding a decomposition by chains.*

Proof: When graph H is acyclic, it uniquely determines a partially ordered set (by Theorem 1 (Robinson, Foulds, 1980)). The partially ordered set is represented by the transitive closure of H. Therefore, when graph H is acyclic and transitive, it represents a partially ordered set.

The decomposition by chains decomposes a partially ordered set into a minimum number of pairwise disjoint chains. On the other hand, the minimum cover by paths does not restrict the cover to comprise of disjoint paths. However, each non-disjoint cover of a transitive graph can be transformed into a disjoint cover without increasing the number of paths. Therefore, the cover of a transitive graph that corresponds to the minimum number of chains, in which the graph is decomposed, is an optimal cover of the graph. Thus, an optimal cover by (disjoint) chains of a transitive graph is an optimal cover of the graph.

□

4.4.5.2 Precedence graphs and transitivity

A transitive closure of an acyclic directed graph represents a partially ordered set. The precedence graphs are directed by construction. We show that the end-time precedence graph G_b is transitive. We also show that the start-time precedence graph G_a can be made transitive (such that the minimum-cover-by-paths on the new transitive G_a has smaller optimal solution value than on the initial G_a).

Theorem 8 *The end-precedence graph G_b is transitive.*

Proof: Suppose that $(i,j) \in E_b \ \wedge \ (j,k) \in E_b$.

$(i,j) \in E_b \ \Rightarrow \ b_i + t^*_{i,j} \leq b_j$

$(j,k) \in E_b \ \Rightarrow \ b_j + t^*_{j,k} \leq b_k$

Since, the times are the shortest travel times we have

$$b_i + t^*_{i,k} \ \leq \ b_i + t^*_{i,j} + t^*_{j,k} \ \leq \ b_j + t^*_{j,k} \ \leq \ b_k \quad \Rightarrow \quad (i,k) \in E_b.$$

□

The start-time precedence graph is not transitive. A TSPTW instance whose G_a is not transitive is: The TSPTW graph is $G = (V, E)$, $V = \{1, 2, 3\}$, $E = \{(1, 2), (2, 3)\}$. The location distances are: $d_{1,2} = 1$, $d_{2,3} = 1$, $d_{1,3} = 2$. The time windows are: $[a_1, b_1] = [0, 0]$, $[a_2, b_2] = [0, 1]$,

$[a_3, b_3] = [0, 1]$. It is clear that the start time precedence graph has edges $(1, 2)$ and $(2, 3)$, but not edge $(1, 3)$.

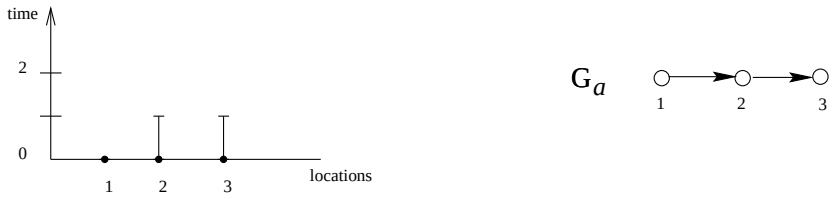


Figure 4.6: A TSPTW problem with collinear locations that is characterized by non-transitive start-time precedence graph.

It is evident that the transitive closure of a graph increases the number of the edges of the original graph. Thus, the transitive closure of a graph can be covered by a smaller number of paths than the original (non-transitive) graph. Since graph G_a is used for finding a lower bound, we can use the transitive closer of G_a for finding a lower bound, too.

4.4.5.3 Precedence graphs and cycles

Definition 10 (Robinson, Foulds (1980)) *If i and j are nodes in V, and there is a $(i, \ldots, j)$ walk, then it is said that j is reachable from i.*

The *inreach* $R^I(i)$ is the set of all nodes in V such that i is reachable from each of them. Similarly, the *outreach* $R^O(i)$ is the set of all nodes that are reachable from i. Note that $i \in R^O(i)$ and $i \in R^I(i)$.

Theorem 3 (Robinson, Foulds (1980)) *(1) $R^O(j) \subset R^O(i)$ iff there is a walk $(i, \ldots, j)$, (2) $R^I(i) \subset R^I(j)$ iff there is a walk $(i, \ldots, j)$.*

Definition 11 (Robinson, Foulds (1980)) *If there is walk $(i, \ldots, j)$ and if there is walk $(j, \ldots, i)$ then it is said that i is equivalent to j.*

Equivalence is an equivalence relation. The set of nodes equivalent to i is called the *component* and it is denoted $C(i)$.

Theorem 4 (Robinson, Foulds (1980)) $C(i) = R^O(i) \cap R^I(i)$

Theorem 5 (Robinson, Foulds (1980)) *If i and j are equivalent then $R^I(i) = R^I(j)$ and $R^O(i) = R^O(j)$.*

Theorem 6 (Robinson, Foulds (1980)) *There is a closed walk passing through nodes i, j iff they are equivalent.*

Definition 12 (Robinson, Foulds (1980)) *Let $D = (V, E)$ be a digraph with components $C_1, ..., C_c$. Let $V^c = \{C_1, ..., C_c\}$ and $E^c = \{(C_u, C_v), u \neq v \mid \exists i \in C_u$ and $\exists j \in C_v$ such that $(i, j) \in E\}$. Then $D^c = (V^c, E^c)$ is called condensation of D.*

If each node of a digraph is a component, the digraph is *acyclic*. Condensation of any digraph is an acyclic digraph.

Therefore, condensation of the start-time precedence graph G_a is acyclic graph. Given that the start-time precedence graph G_a is used for finding a lower bound on the number of vehicles, the condensation G_a^c of G_a can be used, too. The new lower bound can be worse because it assumes that each component of G_a is served by one vehicle, even though this might not be feasible (Theorem 3 on page 77). Intuitively, the lower bound will be better if the average size of components is small, *i.e.*, if the number of components is close to the original number of locations. Set the time window of a component C to

$$[a_C, b_C] = [\min_{i \in C}\{a_i\}, \max_{i \in C}\{b_i\}] \tag{4.4}$$

It is worth mentioning that the average size of components of G_a depends on the number of overlapping time windows, the distances between stop locations, and the size of time windows

(especially overlapping ones). Note that if all the time windows of the locations are mutually disjoint, then the corresponding G_a is acyclic.

The following two lemmas prove that the end-precedence graph G_b is acyclic.

Lemma 4 *If the time restriction has been applied, there is no cycle of size 2 in G_b.*

Proof: The lemma is a corollary of Lemma 1 and Lemma 2. □

Lemma 5 *There is no cycle of any size in G_b.*

Proof: By contradiction: Suppose that $C = (i_1, i_2, ..., i_k)$ is a cycle (of distinct locations) in G_b. Then it is true that $b_{i_1} < b_{i_2} < ... < b_{i_k} < b_{i_1}$. This implies $b_{i_1} < b_{i_1}$ which is false for any b_{i_1}.

4.4.5.4 Algorithm for finding bounds on vehicles of m-TSPTW

Algorithm for finding a lower (an upper) bound on the number of vehicles needed to serve locations of an m-TSPTW:

Step 1: Find the shortest travel times between all pairs of locations.

Step 2: Create precedence graph G_a (G_b).

Step 3: Make transitive closure of G_a. Condense the resulting graph.

Step 4: Apply a bipartite matching algorithm for solving the decomposition-by-chains problem (see discussion at the beginning of Subsection 4.4.5.

Step 5: The number of chains is a lower bound (an upper bound) on the minimum number of vehicles needed to serve all locations of m-TSPTW.

Complexity of the algorithm for finding a lower bound is $O(V^3)$ due to the complexity of making the graph transitive closure (Step 3). Complexity of finding an upper bound is $O(VE)$ due to the complexity of the maximum bipartite matching (Step 4).

4.4.6 PDPTW: Lower bound on the number of vehicles. 1-PDPTW: Feasibility of a request insertion

Above it was proved that the start-time precedence graph and the end-time precedence graph can be used for finding a lower and an upper bound, respectively, on the number of vehicles needed to serve all locations of an m-TSPTW. In the case of PDPTW the situation is a bit more complex. Because the precedence constraints of PDPTW are not enforced in the start-time precedence graph G_a, some of paths that are in the solution of the minimum-cover-by-paths problem of G_a can contain infeasible edges (from the perspective of PDPTW). Nevertheless, the minimum cover by paths will give a lower bound on vehicles for PDPTW, because the start-time precedence graph gives a lower bound for the corresponding m-TSPTW which is a relaxation of PDPTW. Therefore any lower bound for m-TSPTW is a lower bound for PDPTW. Regarding the end-time precedence graph, the precedence constraints are enforced, but the pairing constraints are not. Since the end-precedence graph is used to find an upper bound, both the constraints have to be satisfied by all paths that would cover graph nodes. Therefore, an upper bound of the corresponding m-TSPTW cannot be used as an upper bound for PDPTW.

Examine now a subproblem of a dynamic PDPTW that considers one active vehicle. The subproblem is the single-vehicle PDPTW. One can check the feasibility of serving a new request (by the vehicle) using the upper-bound test and the lower-bound test of Section 4.3. Build graph G whose vertices comprise the vehicle position, the two stop locations of the new request, and all unserved route locations. The precedence graphs of graph G can be used to find a lower and an upper bound on the number of vehicles needed to serve G. (Pairing constraints in G_b are present because of single-vehicle case.)

Note that testing of feasibility of serving a new request in a dynamic 1-PDPTW is equivalent to the problem of finding the longest path in the precedence graphs. Associate a unit length to each edge of the two precedence graphs. Find the longest paths in the two graphs. If the longest path contain all nodes, the minimum number of paths needed to cover all graph nodes is one.

The problem of finding the longest path in a directed acyclic graph can be solved in polynomial time (Garey, Johnson, 1979).

4.5 Best request insertion

The best insertion of a request is resolved by finding the slots to insert the two stop locations such that the increase in the total route lengths is minimized. The most common approach is to first find the best insertion slot for the pickup location, to insert the pickup location, and then to find the best insertion slot for the delivery location. We use the approach of finding the *best coupled insertion*, *i.e.*, finding the best insertion slots for both stop locations together. Subsection 4.5.1 provides an analysis of a route in a dynamic environment. Subsection 4.5.2 explains changes in the route duration due to one location insertion, and shows that the feasibility check of one location insertion can be done in constant time. In the last section we prove that the best coupled insertion of a PDPTW request can be done in $O(n^2)$ time. A similar analysis for the case of TSPTW has been published by Savelsbergh (1985), showing that the verification of 2-optimality can be done in $O(n^2)$ time.

Despite the fact that a routing problem with non-zero service times can be transformed to the problem with 0-service times, we have chosen to carry the service times through this section in order to allow easier software implementation of the ideas proposed.

4.5.1 Arrival and departure times of route locations

Assume that the vehicle's starting position is denoted by 0, and the time when the vehicle leaves position 0 is denoted by t. Additional labels that will be used in this section are:

$\overrightarrow{S_i}$ - slack time for the insertion of a new location after location i

$\overleftarrow{S_i}$ - slack time for the insertion of a new location before location i

$F_{i,p,i+1}$ - forward increase of the earliest times - increase in $\underline{A_{i+1}}$ caused by the insertion of location p between locations i and $i+1$

$B_{i,p,i+1}$ - backward decrease of the latest times - decrease of $\overline{D_i}$ caused by the insertion of location p between locations i and $i+1$.

4.5.1.1 Arrival and departure times

A route for any dynamic routing problem is characterized by specific times assigned to each of its stop locations. The two main time variables assigned to location i are the scheduled arrival time A_i and the scheduled departure time D_i. Based on these values, additional variables can be derived: the start time of service at the location, the end time of service at the location, the wait time at the location before serving the location, and the wait time at the location after serving the location. The lower bound of A_i will be named the earliest arrival time and denoted by $\underline{A_i}$. The latest arrival time $\overline{A_i}$ is the upper bound of A_i. Similar notation will be used for the earliest departure time $\underline{D_i}$ and the latest departure time $\overline{D_i}$. Hence,

$$A_i \in [\underline{A_i}, \overline{A_i}] \tag{4.5}$$

$$D_i \in [\underline{D_i}, \overline{D_i}]. \tag{4.6}$$

The *earliest arrival time* $\underline{A_i}$ at location i is the earliest time when the vehicle *can* reach location i, assuming that all locations preceding i are served within their time windows. (Vehicle leaves location 0 at time t.) Equations for the earliest arrival times and the earliest departure times are:

$$\underline{A_1} = t + t_{0,1} \tag{4.7}$$

$$\underline{D_i} = \max(a_i, \underline{A_i}) + s_i, \qquad \forall i \in \{1, ..., n\} \tag{4.8}$$

$$\underline{A_{i+1}} = \underline{D_i} + t_{i,i+1} = \max(a_i, \underline{A_i}) + s_i + t_{i,i+1}, \qquad \forall i \in \{1, ..., n-1\}. \tag{4.9}$$

The earliest arrival time and the earliest departure time for one location are calculated based on the earliest times of the previous location in the route. The equations show that the earliest arrival times and the earliest departure times for all locations in one route can be calculated in $O(n)$ time, where n is the number of the locations in the route.

We assume, as is usually the case, that the start of service time is equal to the earliest possible start of service:

$$T_i = \underline{T_i} \tag{4.10}$$

where the earliest start of service time is:

$$\underline{T_i} = \max(a_i, \underline{A_i}). \tag{4.11}$$

The *latest departure time* $\overline{D_i}$ from location i is the latest time when the vehicle *has to* leave location i, in order to serve on time all locations following i. The latest arrival times and the latest departure times can be calculated by equations:

$$\overline{A_n} = b_n - s_n \tag{4.12}$$

$$\overline{D_i} = \overline{A_{i+1}} - t_{i,i+1}, \qquad \forall i \in \{1, ..., n-1\} \tag{4.13}$$

$$\overline{A_i} = \min(\overline{D_i}, b_i) - s_i = \min(\overline{A_{i+1}} - t_{i,i+1}, b_i) - s_i, \qquad \forall i \in \{1, ..., n-1\}. \tag{4.14}$$

The latest arrival and departure times for one location are calculated based on the latest times of location following the location i. The latest times of all locations in a route can be found in $O(n)$ time, where n is the number of locations in the route.

The following subsections describe how to update the earliest times and the latest times of locations along a route. The earliest arrival time is specified as a function of time t, where t is the time when the vehicle leaves its current position. The function can be used to update the earliest arrival time when (1) the vehicle has been delayed at its current location or when (2)

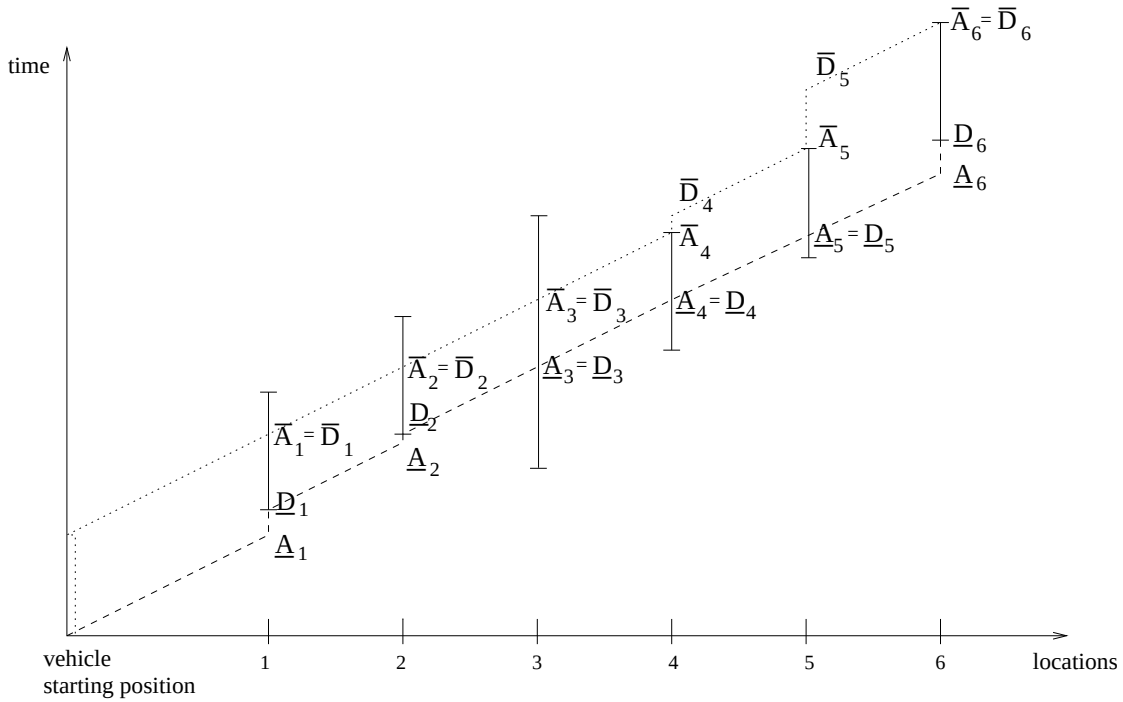


Figure 4.7: The earliest arrival times $\underline{A_i}$, the latest arrival times $\overline{A_i}$, the earliest departure times $\underline{D_i}$ and the latest departure times $\overline{D_i}$ at locations of one route. The intervals are the time windows for each of six locations in the route. The route locations are on the line. Service time at each location is zero. The dashed line represents the route through time if the vehicle route is scheduled according to the earliest times. The dotted line represents the route through time if the vehicle route is scheduled according to the latest times.

a new location has been inserted in the route. The latest arrival time is defined as a function of time b, where b is the end time of the time window of the route's last location. The function can be used to update the latest departure times when (1) the time b has been changed or when (2) a new location has been inserted in the route.

4.5.1.2 Dynamic earliest times

Practical dynamic PDPTW can be often characterized by increased service time at a location, increased waiting time at a location, or worsened traffic conditions. Each of these changes as well as insertion of a new location into the route could be modelled as the vehicle being delayed at the corresponding location. This subsection describes the changes in the route's earliest times if the vehicle is delayed at its starting (current) position. The same equations can be used to determine the consequences of a vehicle delay at any position along the route.

Assume that travel times and service times are constant, and that variable is time t at which the vehicle leaves its current position, *i.e.*, $\underline{A_i} = \underline{A_i}(t)$ and $\underline{D_i} = \underline{D_i}(t)$. Suppose now that the vehicle is delayed at its current position 0 for time $\Delta > 0$. The earliest arrival time at location i is $\underline{A_i}(t + \Delta)$, and, similarly, the earliest departure time is $\underline{D_i}(t + \Delta)$. We call $\underline{A_i}(t + \Delta)$ and $\underline{D_i}(t + \Delta)$ the *dynamic earliest times*.

Theorem 9 *If vehicle is delayed at its current position by time $\Delta > 0$, the earliest arrival time at the first route's location changes according to equation:*

$$\boxed{\underline{A_1}(t + \Delta) = \underline{A_1}(t) + \Delta.}$$

Proof: Based on Equation (4.7), we have $\underline{A_1}(t + \Delta) = t + \Delta + t_{0,1} = \underline{A_1}(t) + \Delta$. □

Theorem 10 *The earliest arrival time $\underline{A_i}(t)$ is a monotone non-decreasing function of time:*

$$\Delta > 0 \;\Rightarrow\; \underline{A_i}(t + \Delta) \geq \underline{A_i}(t).$$

Proof by induction on i:

For $i = 1$ the theorem holds as a corollary of Theorem 9. Suppose that theorem is valid for $i = k$

$$\Delta > 0 \Rightarrow \underline{A_k}(t+\Delta) \geq \underline{A_k}(t). \tag{4.15}$$

Prove that theorem is valid for $i = k + 1$

$$\begin{aligned}
\underline{A_{k+1}}(t+\Delta) &\overset{\text{Equation (4.9)}}{=} \max(a_k, \underline{A_k}(t+\Delta)) + s_k + t_{k,k+1} \geq \\
&\overset{\text{Equation (4.15)}}{\geq} \max(a_k, \underline{A_k}(t)) + s_k + t_{k,k+1} = \\
&= \underline{A_{k+1}}(t).
\end{aligned}$$

□

Lemma 6 *For* $\forall i \in \{2, ..., n\}$

$$\underline{A_i}(t+\Delta) = \begin{cases} \underline{A_i}(t), & \text{if } \underline{A_{i-1}}(t+\Delta) < a_{i-1} \\ \underline{A_{i-1}}(t+\Delta) + s_{i-1} + t_{i-1,i}, & \text{if } \underline{A_{i-1}}(t+\Delta) \geq a_{i-1}. \end{cases}$$

Proof: For $\forall i \in \{2, ..., n\}$

$$\begin{aligned}
\underline{A_i}(t+\Delta) &\overset{\text{Equation (4.9)}}{=} \max(a_{i-1}, \underline{A_{i-1}}(t+\Delta)) + s_{i-1} + t_{i-1,i} = \\
&= \begin{cases} a_{i-1} + s_{i-1} + t_{i-1,i}, & \text{if } \underline{A_{i-1}}(t+\Delta) < a_{i-1} \\ \underline{A_{i-1}}(t+\Delta) + s_{i-1} + t_{i-1,i}, & \text{if } \underline{A_{i-1}}(t+\Delta) \geq a_{i-1} \end{cases} \\
&\overset{\text{Theorem 10}}{=} \begin{cases} \max(a_{i-1}, \underline{A_{i-1}}(t)) + s_{i-1} + t_{i-1,i}, & \text{if } \underline{A_{i-1}}(t+\Delta) < a_{i-1} \\ \underline{A_{i-1}}(t+\Delta) + s_{i-1} + t_{i-1,i}, & \text{if } \underline{A_{i-1}}(t+\Delta) \geq a_{i-1} \end{cases} \\
&\overset{\text{Equation (4.9)}}{=} \begin{cases} \underline{A_i}(t), & \text{if } \underline{A_{i-1}}(t+\Delta) < a_{i-1} \\ \underline{A_{i-1}}(t+\Delta) + s_{i-1} + t_{i-1,i}, & \text{if } \underline{A_{i-1}}(t+\Delta) \geq a_{i-1}. \end{cases}
\end{aligned}$$

□

Theorem 11 *If vehicle leaves its current position at time $t + \Delta$, the earliest arrival times of locations on its route change according to the following equation:*

$$\boxed{\underline{A_i}(t+\Delta) = \begin{cases} \underline{A_i}(t), & (\exists j < i)\underline{A_j}(t+\Delta) < a_j \\ \underline{A_i}(t) + \Delta, & (\forall j < i)\underline{A_j}(t) \geq a_j. \end{cases}}$$

The equation is valid for $\forall i \in \{2, ..., n\}$.

Proof by induction on $i > 1$:

For $i = 2$

$$\underline{A_2}(t+\Delta) \overset{\text{Lemma 6}}{=} \begin{cases} \underline{A_2}(t), & \text{if } \underline{A_1}(t+\Delta) < a_1 \\ \underline{A_1}(t+\Delta) + s_1 + t_{1,2}, & \text{if } \underline{A_1}(t+\Delta) \geq a_1. \end{cases}$$

Second part of equation is

$$\begin{aligned} \underline{A_1}(t+\Delta) + s_1 + t_{1,2} &\overset{\text{Theorem 9}}{=} \underline{A_1}(t) + \Delta + s_1 + t_{1,2} = \\ &= \max(a_1, \underline{A_1}(t)) + s_1 + t_{1,2} + \Delta, \qquad \text{if } \underline{A_1}(t) \geq a_1 \\ &\overset{\text{Equation (4.9)}}{=} \underline{A_2}(t) + \Delta. \end{aligned}$$

Hence,

$$\begin{aligned} \underline{A_2}(t+\Delta) &= \begin{cases} \underline{A_2}(t), & \text{if } \underline{A_1}(t+\Delta) < a_1 \\ \underline{A_2}(t) + \Delta, & \text{if } \underline{A_1}(t+\Delta) \geq a_1 \ \wedge \ \underline{A_1}(t) \geq a_1 \end{cases} \\ &\overset{\text{Theorem 10}}{=} \begin{cases} \underline{A_2}(t), & \text{if } \underline{A_1}(t+\Delta) < a_1 \\ \underline{A_2}(t) + \Delta, & \text{if } \underline{A_1}(t) \geq a_1, \end{cases} \end{aligned}$$

i.e., the theorem is valid for $i = 2$.

Suppose that the theorem is valid for $i = k$

$$\underline{A_k}(t+\Delta) = \begin{cases} \underline{A_k}(t), & (\exists j < k)\underline{A_j}(t+\Delta) < a_j \\ \underline{A_k}(t) + \Delta, & (\forall j < k)\underline{A_j}(t) \geq a_j. \end{cases} \tag{4.16}$$

Prove that the theorem is true for $i = k + 1$.

$$\underline{A_{k+1}}(t+\Delta) \overset{\text{Lemma 6}}{=} \begin{cases} \underline{A_{k+1}}(t), & \text{if } \underline{A_k}(t+\Delta) < a_k \\ \underline{A_k}(t+\Delta) + s_k + t_{k,k+1}, & \text{if } \underline{A_k}(t+\Delta) \geq a_k \end{cases}$$

$$\overset{\text{Equation (4.16)}}{=} \begin{cases} \underline{A_{k+1}}(t), & \text{if } \underline{A_k}(t+\Delta) < a_k \\ \underline{A_k}(t) + s_k + t_{k,k+1}, & \text{if } (\underline{A_k}(t+\Delta) \geq a_k) \wedge \\ & \quad \wedge (\exists j < k)\underline{A_j}(t+\Delta) < a_j \\ \underline{A_k}(t) + \Delta + s_k + t_{k,k+1}, & \text{if } (\underline{A_k}(t+\Delta) \geq a_k) \wedge \\ & \quad \wedge (\forall j < k)\underline{A_j}(t) \geq a_j. \end{cases}$$

It is valid:

$$\underline{A_k}(t) + s_k + t_{k,k+1} = \max(a_k, \underline{A_k}(t)) + s_k + t_{k,k+1}, \quad \text{if } \underline{A_k}(t) \geq a_k$$
$$\overset{\text{Equation (4.9)}}{=} \underline{A_{k+1}}(t),$$

$$\underline{A_k}(t) + \Delta + s_k + t_{k,k+1} = \max(a_k, \underline{A_k}(t)) + \Delta + s_k + t_{k,k+1}, \quad \text{if } \underline{A_k}(t) \geq a_k$$
$$\overset{\text{Equation (4.9)}}{=} \underline{A_{k+1}}(t) + \Delta,$$

thus

$$\underline{A_{k+1}}(t+\Delta) = \begin{cases} \underline{A_{k+1}}(t), & \text{if } (\underline{A_k}(t+\Delta) < a_k) \vee \{(\underline{A_k}(t+\Delta) \geq a_k) \wedge \\ & \quad \wedge [(\exists j < k)\underline{A_j}(t+\Delta) < a_j] \wedge (\underline{A_k}(t) \geq a_k)\} \\ \underline{A_{k+1}}(t) + \Delta, & \text{if } (\underline{A_k}(t) \geq a_k) \wedge [(\forall j < k)\underline{A_j}(t) \geq a_j] \wedge \\ & \quad \wedge (\underline{A_k}(t+\Delta) \geq a_k). \end{cases} \tag{4.17}$$

Label the conditions of Equation (4.17) by C_1 and C_2, respectively.

$$\begin{aligned} C_1 \quad &= \quad (\underline{A_k}(t+\Delta) < a_k) \quad \vee \\ &\quad \vee \quad \{(\underline{A_k}(t+\Delta) \geq a_k) \wedge [(\exists j < k)\underline{A_j}(t+\Delta) < a_j] \wedge (\underline{A_k}(t) \geq a_k)\} = \\ &\overset{\text{Theorem 4.15}}{=} \quad (\underline{A_k}(t+\Delta) < a_k) \quad \vee \quad \{[(\exists j < k)\underline{A_j}(t+\Delta) < a_j] \wedge (\underline{A_k}(t) \geq a_k)\}. \end{aligned}$$

$$\begin{aligned} C_2 \quad &= \quad (\underline{A_k}(t) \geq a_k) \wedge [(\forall j < k)\underline{A_j}(t) \geq a_j] \wedge (\underline{A_k}(t+\Delta) \geq a_k) = \\ &\overset{\text{Theorem 4.15}}{=} \quad (\underline{A_k}(t) \geq a_k) \wedge [(\forall j < k)\underline{A_j}(t) \geq a_j] = \\ &= \quad [(\forall j < k+1)\underline{A_j}(t) \geq a_j]. \end{aligned}$$

We now derive $\underline{A_{k+1}}(t+\Delta)$ for condition

$$C_3 = [(\exists j < k)\underline{A_j}(t+\Delta) < a_j] \wedge (\underline{A_k}(t) < a_k)$$

$$\begin{aligned} \underline{A_{k+1}}(t+\Delta) \quad &\overset{\text{Equation (4.9)}}{=} \quad \max(a_k, \underline{A_k}(t+\Delta)) + s_k + t_{k,k+1} = \\ &\overset{C_3 \text{ and Equation (4.16)}}{=} \quad \max(a_k, \underline{A_k}(t)) + s_k + t_{k,k+1} = \\ &= \quad \underline{A_{k+1}}(t). \end{aligned}$$

Because $\underline{A_{k+1}}(t+\Delta) = \underline{A_{k+1}}(t)$ for C_1 and for C_3, we simplify $C_1 \vee C_3$

$$C_1 \vee C_3 = (\underline{A_k}(t+\Delta) < a_k) \quad \vee \quad \{[(\exists j < k)\underline{A_j}(t+\Delta) < a_j] \wedge (\underline{A_k}(t) \geq a_k)\} \quad \vee$$

$$\vee \quad \{[(\exists j < k)\underline{A_j}(t+\Delta) < a_j] \wedge (\underline{A_k}(t) < a_k)\} =$$

$$= (\underline{A_k}(t+\Delta) < a_k) \quad \vee \quad \{[(\exists j < k)\underline{A_j}(t+\Delta) < a_j] \wedge [(\underline{A_k}(t) \geq a_k) \vee (\underline{A_k}(t) < a_k)]\} =$$

$$= (\underline{A_k}(t+\Delta) < a_k) \ \vee \ [(\exists j < k)\underline{A_j}(t+\Delta) < a_j] =$$

$$= [(\exists j < k+1)\underline{A_j}(t+\Delta) < a_j].$$

So Equation (4.17) becomes

$$\underline{A_{k+1}}(t+\Delta) = \begin{cases} \underline{A_{k+1}}(t), & \text{if } C_1 \vee C_3 \\ \underline{A_{k+1}}(t)+\Delta, & \text{if } C_2 \end{cases}$$

$$= \begin{cases} \underline{A_{k+1}}(t), & (\exists j < k+1)\underline{A_j}(t+\Delta) < a_j \\ \underline{A_{k+1}}(t)+\Delta, & (\forall j < k+1)\underline{A_j}(t) \geq a_j \end{cases}$$

thus proving the theorem. □

The same equation can be used for the earliest arrival time updates after a new location is inserted. In this case, Δ will be equal to $F_{i,p,i+1}$ (see Subsection 4.5.2 for details) and the earliest arrival times have to be updated for all locations following new location p.

4.5.1.3 Dynamic latest times

The latest arrival time and the latest departure time at location i depend on the latest times of the location that follows i (see Subsubsection 4.5.1.1). Labeling $b = b_n$, and assuming that the travel times and the service times are constant, we can write $\overline{A_i} = \overline{A_i}(b)$ and $\overline{D_i} = \overline{D_i}(b)$. This subsection describes changes in the route when b is decreased by $\Delta > 0$. The latest arrival and departure times from location i are $\overline{A_i}(b-\Delta)$ and $\overline{D_i}(b-\Delta)$, and we call them the *dynamic latest times*.

Similar equations and theorems, as given in Subsubsection 4.5.1.2 for the earliest arrival times, can be deduced and proved for the dynamic latest times. Listed here are theorems about the dynamic latest times without proofs.

For the sake of completeness it is assumed that the latest departure time from the last location in the route is equal to the end of the time window of the last location

$$\overline{D_n} = \overline{D_n}(b) = b_n = b. \tag{4.18}$$

Theorem 12 *Due to changes in b, the latest arrival time at the last location in the route changes according to equation:*

$$\boxed{\overline{A_n}(b-\Delta) = \overline{A_n}(b) - \Delta.}$$

Theorem 13 *Due to changes in b, the latest departure time at the last location in the route changes according to equation:*

$$\boxed{\overline{D_n}(b-\Delta) = \overline{D_n}(b) - \Delta.}$$

Theorem 14 *The latest departure time $\overline{D_i}(t)$ $(\forall i < n)$ is a monotone non-increasing function of time, i.e.*

$$\Delta > 0 \;\Rightarrow\; \overline{D_i}(b-\Delta) \leq \overline{D_i}(b).$$

Theorem 15 *If the latest arrival time at the last location is changed to $b-\Delta$, the latest departure times of all locations in the route change according to equation:*

$$\boxed{\overline{D_i}(b-\Delta) = \begin{cases} \overline{D_i}(b), & (\exists j > i)\overline{D_j}(b-\Delta) > b_j \\ \overline{D_i}(b) - \Delta, & (\forall j > i)\overline{D_j}(b) \leq b_j. \end{cases}}$$

The equation is valid for $\forall i \in \{1, ..., n-1\}$.

The same equations can be used for the latest departure time updates after inserting a new location. The latest times of all locations preceding the new location has to be updated. In this case Δ will be equal to $B_{i,p,i+1}$ (see Subsection 4.5.2 for details).

Note that if the vehicle is delayed at its current position and leaves at $t+\Delta$, the latest times of the route locations remain unchanged.

4.5.2 One location insertion

Each location of PDPTW has to be served within its time window. When scheduled to a route, location i can be served only within time interval $[\max(\underline{A_i}, a_i), \min(\overline{D_i}, b_i)]$. This time interval is called the *actual time window* of location i. Now it is obvious that a route is feasible if for each location i the *size of the actual time window*, denoted by

$$I_i = \min(\overline{D_i}, b_i) - \max(\underline{A_i}, a_i) \tag{4.19}$$

is at least equal to the location's service time

$$(\forall i) I_i \geq s_i. \tag{4.20}$$

This section describes the changes in a route due to insertion of one new location, and shows that the feasibility check of one location insertion can be done in constant time. Variables $\overrightarrow{S_i}$ and $\overleftarrow{S_i}$ for slack times, $F_{i,p,i+1}$ for forward increase of the earliest times, and $B_{i,p,i+1}$ for backward decrease of the latest times, with the following definitions will be used extensively:

$$F_{i,p,i+1} \stackrel{\text{def}}{=} t_{i,p} + t_{p,i+1} - t_{i,i+1} + s_p + \max(0, a_p - \underline{A_p}) \tag{4.21}$$

$$B_{i,p,i+1} \stackrel{\text{def}}{=} t_{i,p} + t_{p,i+1} - t_{i,i+1} + s_p + \max(0, \overline{D_p} - b_p) \tag{4.22}$$

$$\overrightarrow{S_i} \stackrel{\text{def}}{=} \overline{D_i} - \max(\underline{A_i}, a_i) - s_i \tag{4.23}$$

$$\overleftarrow{S_{i+1}} \stackrel{\text{def}}{=} \min(\overline{D_{i+1}}, b_{i+1}) - s_{i+1} - \underline{A_{i+1}}. \tag{4.24}$$

Theorem 16 *For any location $i \in \{1, ..., m\}$ the following relation holds:*

$$\overrightarrow{S_i} = \overleftarrow{S_{i+1}} .$$

Proof:

$$
\begin{aligned}
\overrightarrow{S}_i &\overset{\text{Definition (4.23)}}{=} \overline{D_i} - \max\left(\underline{A_i}, a_i\right) - s_i = \\
&\overset{\text{Equation (4.13)}}{=} \overline{A_{i+1}} - t_{i,i+1} - \max\left(\underline{A_i}, a_i\right) - s_i = \\
&\overset{\text{Equation (4.14)}}{=} \min\left(\overline{D_{i+1}}, b_{i+1}\right) - s_{i+1} - t_{i,i+1} - \max(\underline{A_i}, a_i) - s_i \\
&\overset{\text{Equation (4.8)}}{=} \min\left(\overline{D_{i+1}}, b_{i+1}\right) - s_{i+1} - t_{i,i+1} - \underline{D_i} \\
&\overset{\text{Equation (4.9)}}{=} \min\left(\overline{D_{i+1}}, b_{i+1}\right) - s_{i+1} - \underline{A_{i+1}} = \\
&\overset{\text{Definition (4.24)}}{=} \overleftarrow{S}_{i+1} \ .
\end{aligned}
$$

□

Theorem 17 *After the insertion of location p between locations i and $i+1$ the value of $\underline{A_{i+1}}$ is increased by the value of $F_{i,p,i+1}$:*

$$\underline{A^*_{i+1}} = \underline{A_{i+1}} + F_{i,p,i+1},$$

where $\underline{A^*_{i+1}}$ is the new value for the earliest arrival time at location $i+1$ after the insertion.

Proof:

$$
\begin{aligned}
F_{i,p,i+1} &\overset{\text{Definition (4.21)}}{=} t_{i,p} + t_{p,i+1} - t_{i,i+1} + s_p + \max\left(0, a_p - \underline{A_p}\right) \\
\underline{A_{i+1}} &\overset{\text{Equation (4.9)}}{=} \max\left(a_i, \underline{A_i}\right) + s_i + t_{i,i+1} \\
\underline{A_p} &\overset{\text{Equation (4.9)}}{=} \max\left(a_i, \underline{A_i}\right) + s_i + t_{i,p} \\
\underline{A^*_{i+1}} &\overset{\text{Equation (4.9)}}{=} \max\left(a_p, \underline{A_p}\right) + s_p + t_{p,i+1}
\end{aligned}
$$

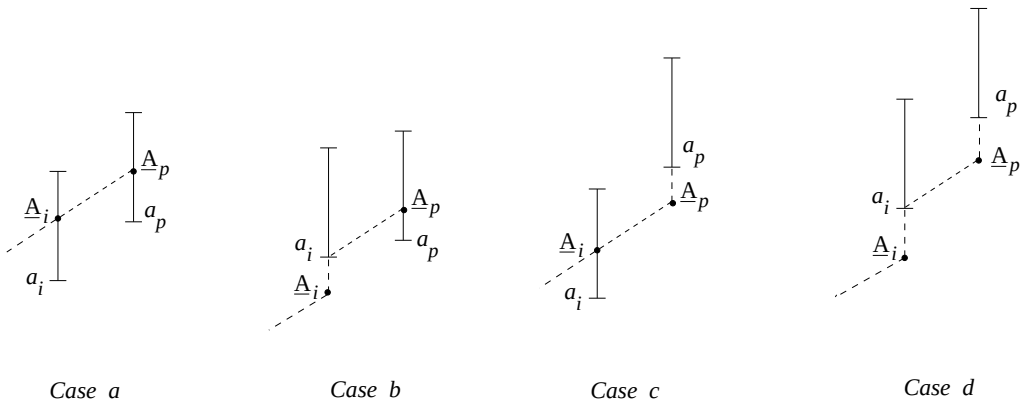


Figure 4.8: Different cases of the relation among the earliest arrival times and the start of the time windows for two successive locations in a route. These cases are used in the proof of Theorem 17.

$$\begin{aligned}\underline{A^*_{i+1}} - \underline{A_{i+1}} &= \max\left(a_p, \max\left(a_i, \underline{A_i}\right) + s_i + t_{i,p}\right) + s_p + t_{p,i+1} - \\ &- \max\left(a_i, \underline{A_i}\right) - s_i - t_{i,i+1}\end{aligned}$$

$Case\ a:\quad (a_p < \underline{A_p}) \wedge (a_i < \underline{A_i})$

$$\begin{aligned}\underline{A^*_{i+1}} - \underline{A_{i+1}} &= \underline{A_i} + s_i + t_{i,p} + s_p + t_{p,i+1} - \underline{A_i} - s_i - t_{i,i+1} = \\ &= t_{i,p} + t_{p,i+1} - t_{i,i+1} + s_p = \\ &= F_{i,p,i+1}.\end{aligned}$$

$Case\ b:\quad (a_p < \underline{A_p}) \wedge (a_i \geq \underline{A_i})$

$$\begin{aligned}\underline{A^*_{i+1}} - \underline{A_{i+1}} &= a_i + s_i + t_{i,p} + s_p + t_{p,i+1} - a_i - s_i - t_{i,i+1} = \\ &= t_{i,p} + t_{p,i+1} - t_{i,i+1} + s_p = \\ &= F_{i,p,i+1}.\end{aligned}$$

$Case\ c:\quad (a_p \geq \underline{A_p}) \wedge (a_i < \underline{A_i})$

$$\begin{aligned}
\underline{A^*_{i+1}} - \underline{A_{i+1}} &= a_p + s_p + t_{p,i+1} - \underline{A_i} - s_i - t_{i,i+1} = \\
&= t_{p,i+1} - t_{i,i+1} + s_p - (\underline{A_i} + s_i + t_{i,p}) + t_{i,p} + a_p = \\
&= t_{p,i+1} - t_{i,i+1} + s_p - (max(\underline{A_i}, a_i) + s_i + t_{i,p}) + t_{i,p} + a_p = \\
&= t_{p,i+1} - t_{i,i+1} + s_p - \underline{A_p} + t_{i,p} + a_p = \\
&= t_{i,p} + t_{p,i+1} - t_{i,i+1} + s_p + (a_p - \underline{A_p}) = \\
&= t_{i,p} + t_{p,i+1} - t_{i,i+1} + s_p + \max(0, a_p - \underline{A_p}) = \\
&= F_{i,p,i+1}.
\end{aligned}$$

$Case\ d:\quad (a_p \geq \underline{A_p}) \wedge (a_i \geq \underline{A_i})$

$$\begin{aligned}
\underline{A^*_{i+1}} - \underline{A_{i+1}} &= a_p + s_p + t_{p,i+1} - a_i - s_i - t_{i,i+1} = \\
&= t_{i,p} + t_{p,i+1} - t_{i,i+1} + s_p + [a_p - (a_i + s_i + t_{i,p})] = \\
&= t_{i,p} + t_{p,i+1} - t_{i,i+1} + s_p + [a_p - (\max(a_i, \underline{A_i}) + s_i + t_{i,p})] = \\
&= t_{i,p} + t_{p,i+1} - t_{i,i+1} + s_p + [a_p - \underline{A_p}] = \\
&= t_{i,p} + t_{p,i+1} - t_{i,i+1} + s_p + \max(0, a_p - \underline{A_p}) = \\
&= F_{i,p,i+1}.
\end{aligned}$$

□

Note that insertion of the new location p into the route between locations i and $i+1$ does not influence the earliest arrival or the earliest departure times of locations $\{1, ..., i\}$.

Theorem 18 ***Feasibility of a location insertion - Test 1:*** *After the insertion of p between i and $i+1$, the route section $\{i+1, ..., n\}$ will remain feasible if $F_{i,p,i+1} \leq \overleftarrow{S}_{i+1}$ is valid, i.e.,*

$$(\forall k > i)\ I_k = \min(\overline{D_k}, b_k) - \max(\underline{A_k}, a_k) \geq s_k \quad \wedge \quad F_{i,p,i+1} \leq \overleftarrow{S}_{i+1} \quad \Longrightarrow$$

$$\Longrightarrow \quad (\forall k > i)\ I^*_k = \min(\overline{D_k}, b_k) - \max(\underline{A^*_k}, a_k) \geq s_k.$$

Proof by induction on $k \geq i+1$:

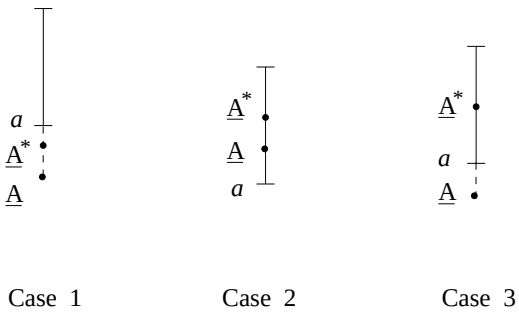


Figure 4.9: Different cases of the relation among the old earliest arrival time, the new earliest arrival time, and the start of the time window for a location in a route. The old earliest arrival time is the earliest arrival time prior to the insertion of a new location in the route. The new earliest arrival time is the earliest arrival time after the insertion of the new location in the route. The new location has been inserted before the location whose time window is shown in the figure. The three cases shown are used in the proof of Theorem 18.

Proof that the theorem is valid for $k = i + 1$

Case 1: $(\underline{A_{i+1}} < a_{i+1}) \wedge (\underline{A^*_{i+1}} < a_{i+1})$

$$\begin{aligned} I^*_{i+1} &= I_{i+1} \geq && \text{because} \quad \max(\underline{A^*_{i+1}}, a_{i+1}) = \max(\underline{A_{i+1}}, a_{i+1}) = a_{i+1} \\ &\geq s_{i+1}. \end{aligned}$$

Case 2: $(\underline{A_{i+1}} \geq a_{i+1}) \wedge (\underline{A^*_{i+1}} \geq a_{i+1})$

$$\begin{aligned} I^*_{i+1} &\overset{\text{Figure 4.9}}{=} I_{i+1} - (\underline{A^*_{i+1}} - \underline{A_{i+1}}) = \\ &\overset{\text{Theorem 17}}{=} I_{i+1} - F_{i,p,i+1} \geq \\ &\overset{\text{this theorem assumption}}{\geq} I_{i+1} - \overleftarrow{S}_{i+1} = \\ &\overset{\text{Equation (4.19)}\ \text{Definition (4.24)}}{=} (\min(\overline{D_{i+1}}, b_{i+1}) - \underline{A_{i+1}}) - (\min(\overline{D_{i+1}}, b_{i+1}) - s_{i+1} - \underline{A_{i+1}}) = \end{aligned}$$

$$= \qquad s_{i+1}.$$

Case 3: $(\underline{A_{i+1}} < a_{i+1}) \wedge (\underline{A^*_{i+1}} \geq a_{i+1})$

$$\begin{aligned}
I^*_{i+1} &\overset{\text{Figure 4.9}}{=} I_{i+1} - (\underline{A^*_{i+1}} - a_{i+1}) = \\
&\overset{\text{Theorem 17}}{=} I_{i+1} - (\underline{A_{i+1}} + F_{i,p,i+1} - a_{i+1}) = \\
&\overset{\text{this theorem assumption}}{\geq} I_{i+1} - \underline{A_{i+1}} - \overleftarrow{S}_{i+1} + a_{i+1} = \\
&\overset{\text{Equation (4.19)}}{=} \min(\overline{D_{i+1}}, b_{i+1}) - a_{i+1} - \underline{A_{i+1}} - \overleftarrow{S}_{i+1} + a_{i+1} = \\
&\overset{\text{Definition (4.24)}}{=} \min(\overline{D_{i+1}}, b_{i+1}) - \underline{A_{i+1}} - \min(\overline{D_{i+1}}, b_{i+1}) + s_{i+1} + \underline{A_{i+1}} = \\
&= s_{i+1}.
\end{aligned}$$

Suppose that the theorem is valid for $k > i+1$

$$I_k \geq s_k \quad \wedge \quad F_{i,p,i+1} \leq \overleftarrow{S}_{i+1} \quad \Longrightarrow \quad I^*_k \geq s_k.$$

Proof that the theorem is valid for $k+1$, when $k > i+1$

$$I_{k+1} \geq s_{k+1} \quad \wedge \quad F_{i,p,i+1} \leq \overleftarrow{S}_{i+1} \quad \overset{?}{\Longrightarrow} \quad I^*_{k+1} \geq s_{k+1}.$$

Case 1: $(\underline{A_{k+1}} < a_{k+1}) \wedge (\underline{A^*_{k+1}} < a_{k+1})$

$$\begin{aligned}
I^*_{k+1} &= I_{k+1} \geq \qquad \text{because} \quad \max(\underline{A^*_{k+1}}, a_{k+1}) = \max(\underline{A_{k+1}}, a_{k+1}) = a_{k+1} \\
&\geq s_{k+1}.
\end{aligned}$$

Case 2: $(\underline{A_{k+1}} \geq a_{k+1}) \wedge (\underline{A^*_{k+1}} \geq a_{k+1})$

$$
\begin{aligned}
I^*_{k+1} &\overset{\text{Figure 4.9}}{=} I_{k+1} - (\underline{A^*_{k+1}} - \underline{A_{k+1}}) = \\
&\overset{\text{Equation (4.19)}}{=} (\min(\overline{D_{k+1}}, b_{k+1}) - \underline{A_{k+1}}) - (\underline{A^*_{k+1}} - \underline{A_{k+1}}) = \\
&= \min(\overline{D_{k+1}}, b_{k+1}) - \underline{A^*_{k+1}} = \\
&= \min(\overline{D_{k+1}}, b_{k+1}) - s_{k+1} + s_{k+1} - \underline{A^*_{k+1}} = \\
&\overset{\text{Equation (4.14)}}{=} \overline{A_{k+1}} + s_{k+1} - \underline{A^*_{k+1}} = \\
&\overset{\text{Equation (4.9)}}{=} \overline{A_{k+1}} + s_{k+1} - (\underline{D^*_k} + t_{k,k+1}) = \\
&\overset{\text{Equation (4.8)}}{=} \overline{A_{k+1}} + s_{k+1} - (\max(\underline{A^*_k}, a_k) + s_k + t_{k,k+1}) \geq \\
&\overset{\text{Assumption for } k}{\geq} \overline{A_{k+1}} + s_{k+1} - (\min(\overline{D_k}, b_k) + t_{k,k+1}) = \\
&= \overline{A_{k+1}} - t_{k,k+1} - \min(\overline{D_k}, b_k) + s_{k+1} = \\
&\overset{\text{Equation (4.13)}}{=} \overline{D_k} - \min(\overline{D_k}, b_k) + s_{k+1} = \\
&= \max(0, \overline{D_k} - b_k) + s_{k+1} \geq \\
&\geq s_{k+1}.
\end{aligned}
$$

Case 3: $(\underline{A_{k+1}} < a_{k+1}) \wedge (\underline{A^*_{k+1}} \geq a_{k+1})$

$$
\begin{aligned}
I^*_{k+1} &\overset{\text{Figure 4.9}}{=} I_{k+1} - (\underline{A^*_{k+1}} - a_{k+1}) = \\
&\overset{\text{Equation (4.19)}}{=} (\min(\overline{D_{k+1}}, b_{k+1}) - a_{k+1}) - (\underline{A^*_{k+1}} - a_{k+1}) = \\
&= \min(\overline{D_{k+1}}, b_{k+1}) - \underline{A^*_{k+1}} \geq \\
&\overset{\text{as in Case 2}}{\geq} s_{k+1}.
\end{aligned}
$$

□

Theorem 19 *After the insertion of location p between locations i and $i+1$ the value of $\overline{D_i}$ is decreased by the value of $B_{i,p,i+1}$:*

$$\overline{D^*_i} = \overline{D_i} - B_{i,p,i+1}$$

$\overline{D_i^*}$ *is the new value for the latest departure time at location* i*, i.e., after the insertion.*

The proof of this theorem is omitted as being symmetric to the proof of Theorem 17. It is evident that insertion of the new location p into the route between locations i and $i+1$ does not influence the latest departure or the latest arrival times of locations $\{i+1, ..., n\}$.

Theorem 20 ***Feasibility of a location insertion - Test 2:*** *After the insertion of* p *between* i *and* $i+1$*, the route segment* $\{1, ..., i\}$ *will remain feasible if* $B_{i,p,i+1} \leq \vec{S}_i$ *is valid, i.e.,*

$$(\forall k \leq i)\ I_k = \min(\overline{D_k}, b_k) - \max(\underline{A_k}, a_k) \geq s_k \ \ \wedge \ \ B_{i,p,i+1} \leq \vec{S}_i \ \ \Longrightarrow$$

$$\Longrightarrow \quad (\forall k \leq i)\ I_k^* = \min(\overline{D_k^*}, b_k) - \max(\underline{A_k}, a_k) \geq s_k.$$

The proof of this theorem can be done by backward induction and it is very similar to the proof of Theorem 18.

Theorem 21 *Checking insertion feasibility of the new location* p *between locations* i *and* $i+1$ *can be done in* $O(1)$ *time by checking the validity of two inequalities:*

$$F_{i,p,i+1} \leq \overleftarrow{S}_{i+1} \tag{4.25}$$

$$B_{i,p,i+1} \leq \vec{S}_i = \overleftarrow{S}_{i+1}\ . \tag{4.26}$$

Proof: This theorem is a corollary of Theorem 18, Theorem 20 and Theorem 16. □

4.5.3 One request insertion

One PDPTW request consists of two locations: a pickup location and a delivery location. The earliest arrival times and the earliest departure times can be calculated or updated in $O(n)$ time for a route that covers n locations. Of equal complexity is the calculation or update of the latest arrival and departure times. The feasibility of one location insertion can be checked in constant time, if certain information pertaining to route locations are maintained (see Subsection 4.5.2).

Therefore, the best coupled insertion of one request can be found in $O(n^2)$ time, when objective is minimization of route lengths.

In practical implementations Theorem 11 and Theorem 15 can make the updates of dynamic times $\underline{A_i}, \underline{D_i}, \overline{A_i}, \overline{D_i}$ faster, despite the fact that the worst case scenario still remains of $O(n)$ complexity. Theorem 11 and introduction of two new labels for each route location would increase the speed of updates of earliest times in the route.

The first label is *arrival label R*. For stop location i, the arrival label R_i is equal to one if $\underline{A_i} \geq a_i$, *i.e.*, if the scheduled arrival time at i is not smaller than the start time of its time window.

$$R_i = \begin{cases} 0, & \text{if } \underline{A_i} < a_i \\ 1, & \text{if } \underline{A_i} \geq a_i \end{cases} \tag{4.27}$$

The other label is *dynamic label Q*. Location i has $Q_i = 1$ if $\underline{A_i}(t + \Delta) \neq \underline{A_i}(t)$, *i.e.*, if the earliest arrival time at i is influenced by a delay on the route section that precedes i.

$$Q_i = \begin{cases} 1, & \text{if } \underline{A_i}(t + \Delta) \neq \underline{A_i}(t) \\ 0, & \text{otherwise} \end{cases} \tag{4.28}$$

The following holds:

$$Q_1 = 1 \tag{4.29}$$

$$Q_i = 1 \Leftrightarrow R_i = 1 \wedge Q_{i-1} = 1. \tag{4.30}$$

The dynamic label Q is such that only the locations belonging to a consecutive sequence (of route locations), starting at the first route location, can have its dynamic labels equal to one.

The labels introduced guide the updates of the earliest times due to insertion of a new location. Also, these labels can be used for updates of location earliest times in situations that require regular updates. One such a situation is an unscheduled delay of a vehicle. Another is the usage of the wait-first waiting strategy (Subsection 5.3.2) for solving route scheduling. The

times of all locations whose $Q_i = 1$ are to be changed during the regular time updates. We call the location variables that have to be updated regularly *the dynamic location variables*, and they include: the earliest arrival time, the earliest departure time, the slack times, the arrival label and the dynamic label. The dynamic location variables have to be updated at least: before scheduling a new request, and after inserting of a new location.

Chapter 5

Scheduling

5.1 Introduction

This chapter concentrates on the scheduling aspect of a dynamic pickup and delivery problem with time windows. The problem consists of constructing a schedule for a vehicle when requests are already assigned and the route is already known. The importance of scheduling in a dynamic vs. a static problem is first explained, following by strategies that can be used for building a schedule in a dynamic environment.

The first two strategies introduced are the simplest strategies. They also reflect the two most extreme situations that can be imposed on vehicles with respect to waiting. Later, two more complex strategies are designed over a new model of a route. Each route, as per dynamic clustering introduced in Section 5.4, can be seen as a series of three dimensional boxes each of which contains a sequence of consecutive route locations that are close to each other. This route clustering is the underlying structure upon which the two complex strategies are built. The last strategy, the advanced dynamic waiting has achieved the best results in all experiments. All other strategies have advantages and disadvantages which will be addressed in the following sections.

5.2 Schedule for a fixed route

Constructing a schedule for a fixed route requires finding the exact values for the scheduled arrival time and the scheduled departure time for each route location. (The start of service time is equal to the earliest possible start of service having the scheduled arrival time at the location (see Equations 4.10 and 4.11).) Once the schedule is determined, it also defines the distribution of waiting times along the route.

In the static vehicle routing problem, scheduling can be (and usually is) done after the routing problem is completely resolved. In the dynamic vehicle routing, scheduling has to be done on-line because currently available routes have to be implemented. The implementation of the routes requires the route schedules to be known.

In the static routing problem with hard time windows, the schedule influences the duration of the route, while the length of the route is unchanged. In the dynamic environment however, the schedule can influence even the route length. The schedule determines the distribution of waiting times, which can greatly influence the feasibility of future location insertions. The vehicle's waiting time is more than a commodity on which the route duration depends. The waiting time can be, and should be, seen as a resource. Consequently, the scheduling is even more important in the dynamic time-constrained routing problem than in the static problem.

As per a vehicle schedule the vehicle can leave its starting position at: (1) the earliest possible time, or (2) the earliest possible time such that the first stop location would be served as soon as the vehicle arrives (without any waiting), or (3) the latest possible time such that each location will be served within its time window, or (4) at any time instant between the extremes given in (1) and (3).

A policy that guides schedule development will be called the *waiting strategy*. The waiting strategy determines the scheduled arrival and the scheduled departure times of route locations. Consequently, the waiting strategy determines the distribution of waiting times along the route. It actually defines where a vehicle will spend its waiting time periods. The waiting period is, in

fact, the time interval that can be used for the possible insertion of a new location, because it increases the slack time between route locations. (The slack time after location i is the difference between the latest departure time from i and the earliest end of the service time at i.)

After discussing two simple waiting strategies, more complex waiting strategies are presented.

5.3 Two simple waiting strategies

5.3.1 Drive-first waiting strategy

The drive-first strategy requires the vehicle to drive as soon as it is feasible. This strategy recommends waiting at a stop location *before* serving it. When a vehicle schedule is built according to the drive-first waiting strategy the vehicle will wait only if it reaches a location before the start of the location's time window. Figure 5.1 shows the vehicle using the drive-first strategy. The route locations in the figure are collinear.

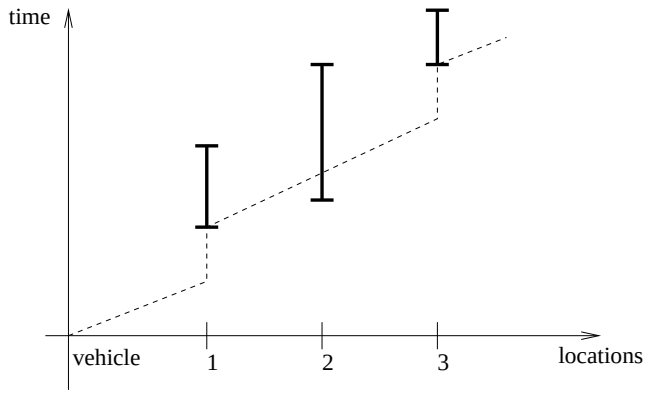


Figure 5.1: The drive-first waiting strategy: the vehicle drives as soon as possible and waits only if reaches a location before the start of the location's time window. The route locations are collinear and positioned along the horizontal axis. The three vertical intervals are the location's time windows. The dashed broken rising line shows the vehicle route in time.

The scheduled arrival time at a location is the earliest possible arrival time, *i.e.*, $A_i = \underline{A_i}$. The scheduled departure time from a location is the earliest departure time, *i.e.*, $D_i = \underline{D_i}$. The waiting time at location i before serving it, is equal $\max(0, a_i - A_i)$, while the waiting time after serving is zero.

The drive-first waiting strategy is the most commonly used waiting strategy. This might be because the drive-first strategy is the only appropriate strategy for static vehicle routing (when transporting goods and parcels). It is also worth mentioning that in a dynamic environment, maintaining the earliest arrival and departure times at route locations is the easiest with the drive-first strategy. The earliest times remain the same if no new location is inserted in the route.

5.3.2 Wait-first strategy

The wait-first strategy requires the vehicle to wait at the current location for as long as it is feasible. This strategy recommends waiting at a stop location *after* serving it. The vehicle waits for the latest time to leave the current location and to drive toward the next stop location. The next location will be served immediately after being reached. Figure 5.2 shows the vehicle using the wait-first strategy. The route locations in the figure are collinear.

The scheduled arrival time at a location is the latest feasible arrival time, *i.e.*, $A_i = \overline{A_i}$. The scheduled departure time from a location is the latest departure time, $D_i = \overline{D_i}$. The waiting time at a location before serving it, is zero. The waiting time at location i, after serving it, is $\max(0, D_i - b_i)$.

The wait-first strategy opens possibility of building better (shorter) routes, because it allows serving, by the same vehicle, new future locations close to the first route locations. These insertions might be infeasible when the drive-first strategy is used. Empirical study have shown that the routes built by the wait-first strategy were shorter (in the total length). Unfortunately, a disadvantage of the wait-first strategy is that it requires more vehicles than the drive-first

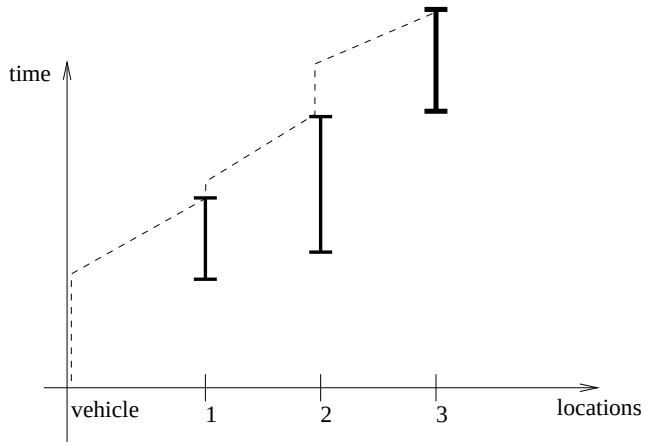


Figure 5.2: The wait-first strategy: the vehicle waits at the current location for as long as it is feasible. The route locations are collinear and positioned along the horizontal axis. The three vertical intervals are the location's time windows. The dashed broken rising line shows the vehicle route in time.

strategy, to serve the same set of requests.

Note also that the implementation of the wait-first strategy is more complex than the implementation of the drive-first strategy. The wait-first strategy requires maintenance of the earliest arrival/departure times because their values constantly change while the vehicle is waiting.

5.4 Dynamic clustering of a route in time and space

Consider partitioning a route locations into subsets such that each subset contains a number of consecutive locations close to each other. Since the problem is time-constrained, one subset of locations can be modelled by a three-dimensional box. We named it *the service zone*. Geometrically, a service zone is considered to be a cylinder or a prism.

The base of the service zone is a figure that encloses the positions of all the locations. Assume choosing the minimal enclosing rectangle.

The height of the service zone is a time span representing the time dimension of the locations

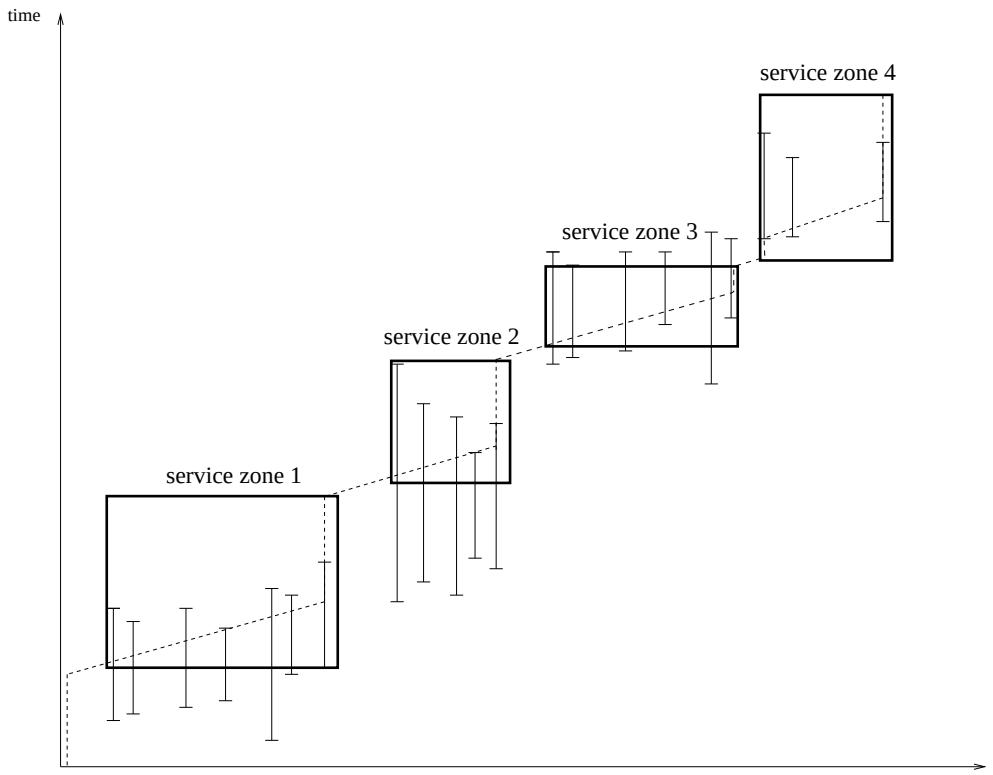


Figure 5.3: The route that consists of four service zones. The route locations are collinear.

that comprise the service zone. The service zone time span is to be determined by the scheduled arrival time at the first location in the service zone and the scheduled departure time from the last location in the zone. When the scheduled times are not known, the time span starts at the earliest start of service time of the first zone's location, and ends at the earliest end of service time of the last zone's location. The earliest end of service time is used for the zone time span, instead of the latest end of the service time, in order to avoid overlapping of time spans of two consecutive service zones of a route. (The latest end of service time at one location can be larger than the earliest start of service time at the following location in the route. The simplest example is when $\overline{D}_i = b_i$, $\underline{A}_{i+1} = a_i$ and $b_i > a_{i+1}$.)

A route now can be modelled by a series of the three-dimensional service zones. An example of route that contains collinear locations is shown in Figure 5.3. The location time windows are shown as vertical intervals. The rectangles in the figure represent the route service zones. They are two-dimensional figures because the route locations are on the line.

The introduced route clustering is referred to as dynamic because the route service zones change in time by insertion of locations, or simply by time passing (when some route locations are served, or when the earliest start of service time of a location being changed). Insertion of a new location in the route can cause one of the following situations:

(1) A new service zone is created containing the new location.

(2) The new location becomes part of an already existing service zone. The zone can be enlarged or it can remain unchanged.

(3) Existing service zone is split in two and the new location becomes a part of one of them.

(4) Existing service zone is split in two and a new service zone is created between them containing the new location.

The new service zone can merge with one of its two neighboring service zones. Similarly, location removal from the route causes rebuilding of service zones. The decisions in maintaining the route service zones depend on the pre-specified maximal size of the service zone base. The size of any service zone base cannot exceed that maximal value, thus occasionally new service zones are created or existing service zones are split in two. Our software implementation of the service zones is done in such a way that the service zones are updated whenever some change in the route appears.

Upon the idea of the dynamic route clustering, the next two waiting strategies are built. These waiting strategies deal better with arrangement of waiting intervals along a route (in respect of having a few larger waiting intervals in the route). The dynamic route clustering can also be used for measuring the distance between a route and a request.

5.5 Dynamic waiting strategy

5.5.1 Motivation

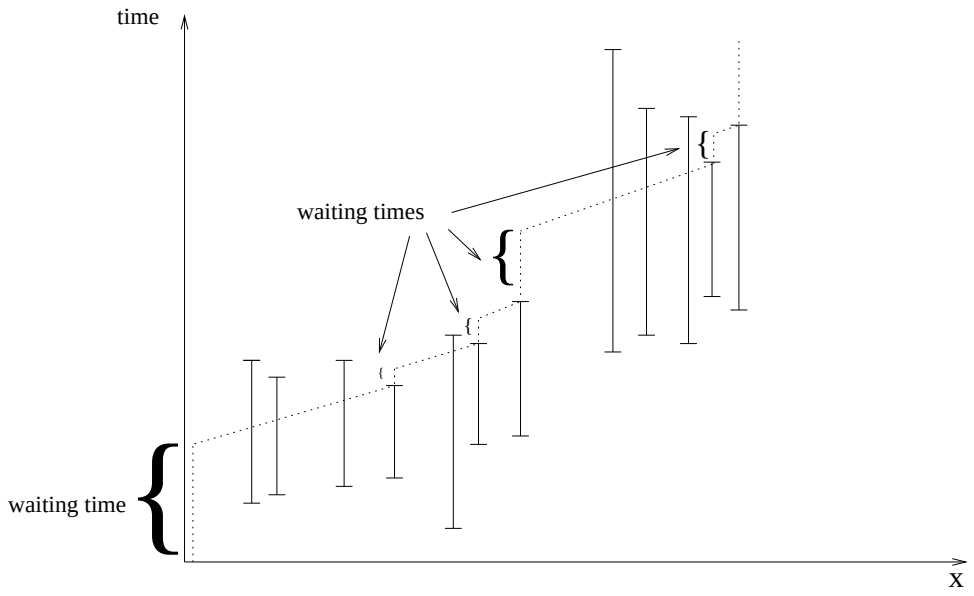


Figure 5.4: Waiting times generated by the wait-first strategy.

Our empirical study has shown a decrease in the total route lengths (the sum of the total distance travelled by all vehicles) when the wait-first strategy is used compared to the drive-first strategy (check the first rows of Table 7.8 on page 160), with the exception of experiments on 500-request instances. On the other hand, the drive-first strategy used up to 17% fewer vehicles. Below a reasoning is provided to explain inferior performance (in terms of vehicle count) of the wait-first strategy, which lies in tendency of the wait-first strategy to concentrate long waiting times in the first part of the route, leaving too little time (slack or waiting) in the rest of the route.

Figure 5.4 shows the waiting times generated by the wait-first strategy, and illustrates that the longest waiting period is at the beginning of the route, before the vehicle leaves its starting

position. Thus, while the vehicle waits for as many as possible new requests to appear for the sake of building better (shorter) route, the vehicle is spending a valuable portion of time doing nothing. When the vehicle finally leaves the starting position, it might well happen that the vehicle has to drive without any significant waiting period along the route. This might cause a situation where many of the requests appearing (after the time when vehicle leaves the starting position) cannot be served by that vehicle. This might result in an increase in the number of vehicles needed to serve all requests.

5.5.2 The strategy

The dynamic waiting strategy is another strategy that can be used to schedule a route. It is a mixture of the drive-first and the wait-first strategies, and represents a way of better arranging waiting times along the whole route. Underlying structure of this strategy is the dynamic route clustering described in Section 5.4. The dynamic waiting strategy constructs the vehicle schedule in the following manner:

(1) the vehicle drives and waits within each of its service zones according to the drive-first waiting strategy, and

(2) when the vehicle finishes serving all locations in one service zone, the vehicle uses the wait-first strategy.

In other words, the vehicle drives using the simplest and the most greedy strategy while serving close locations, and when all such locations are served and the vehicle has to serve the first far location, the vehicle is required to wait for as long as it is feasible.

Consider collapsing all route locations of one service zone into a super-location. Then, the dynamic waiting strategy is equivalent to scheduling all super-locations by the wait-first strategy, while the scheduling of all locations within one super-location is done by the drive-first strategy. The same route shown in Figure 5.3 is scheduled by the dynamic waiting strategy and given in

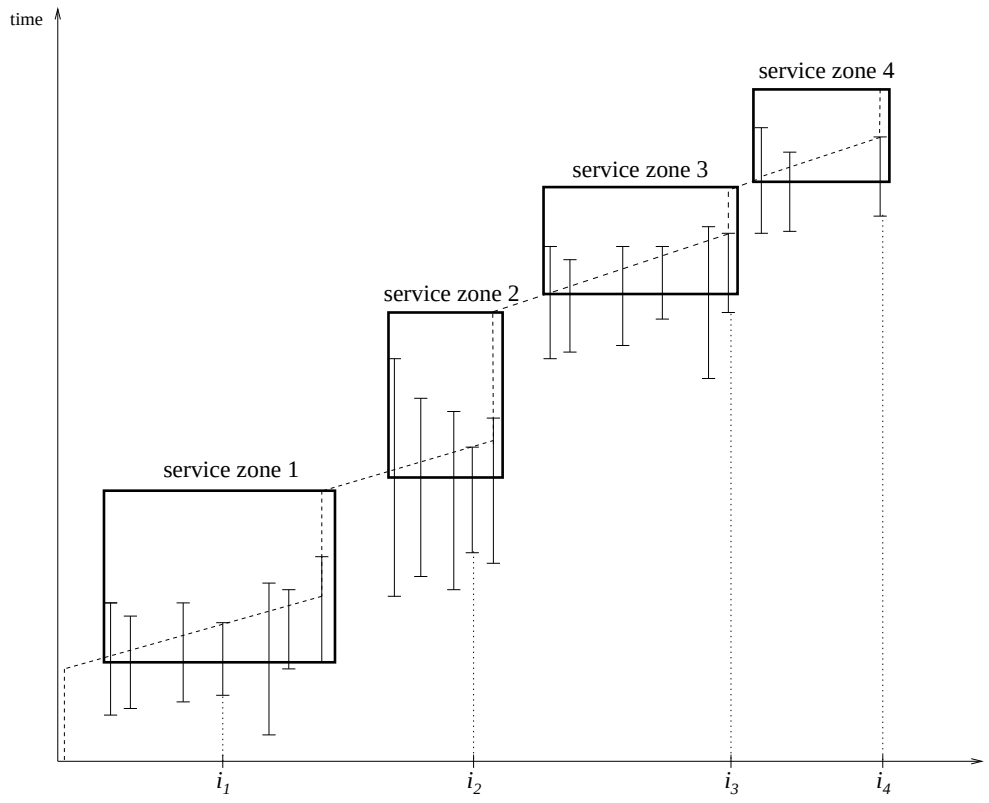


Figure 5.5: The route that consists of four service zones. The route locations are collinear. The scheduling is done by the dynamic waiting strategy.

Figure 5.5. The locations marked (i_1, i_2, i_3, i_4) dictate the wait-first strategy for the route super-locations. The time window of location i_j determines the latest departure time from service zone $j-1$. The time window of location i_1 determines the latest departure time from the starting vehicle position.

An advantage of the dynamic waiting strategy over the drive-first waiting strategy is in the increased waiting at some locations along the route, increasing the possibility of assigning of new requests to the closest service zone. A disadvantage of the dynamic waiting strategy compared with the drive-first waiting strategy is the usage of more vehicles in many problem instances

that were tested. An advantage of the dynamic waiting strategy over the wait-first strategy is in a better propagation of waiting times and a better arrangement of waiting periods along the route. The waiting time available in the route is divided (by the dynamic waiting strategy) in a few larger intervals, and they are distributed along the whole route (see Figures 5.4 and 5.6). This leaves more time for new request insertion in the second half of the service period (compared to the wait-first strategy).

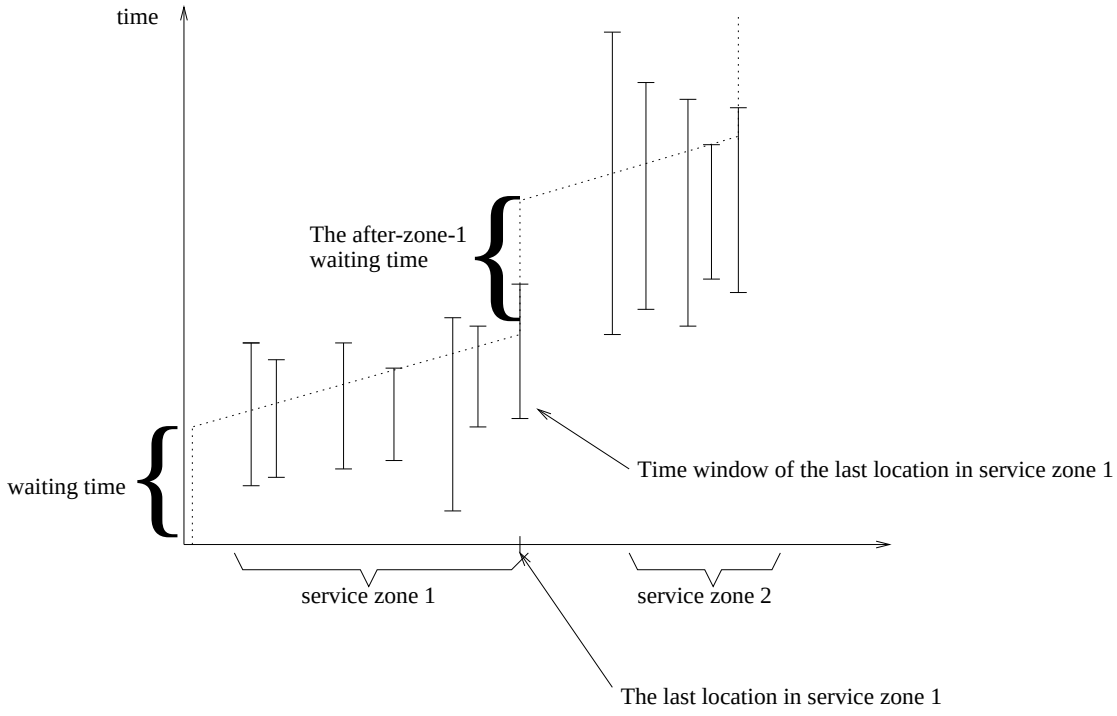


Figure 5.6: Waiting times generated by the dynamic waiting strategy.

5.6 Advanced dynamic waiting strategy

5.6.1 Motivation

The dynamic waiting strategy is shown to be better than the drive-first strategy: the total route lengths were up to 8% shorter. Compared to the wait-first strategy the total route lengths were shorter by up to 2.34% on the one-depot instances, and the total route lengths were longer by up to 4.63% on the no-depot instances. The dynamic waiting strategy used fewer number of vehicles compared to the wait-first strategy. Unfortunately the number of vehicles was still higher on many instances compared to the number of vehicles used by the drive-first strategy. (For more details check Table 7.13 on page 164 and Table 7.8 on page 160).

In the case of PDPTW where no requests were known before the start of the service period, the dynamic waiting strategy can generate a schedule such that the majority of waiting time is after the first service zone of the route. This situation is similar to the one when the wait-first strategy is used. The advanced dynamic waiting strategy propagates waiting times and arranges the waiting periods along a route in a better manner.

5.6.2 The strategy

The advanced dynamic waiting strategy strives to propagate waiting time available in the route along the whole route, and relies on clustering of the route in service zones. Inside each route service zone, the total waiting time remains the same as in the dynamic waiting strategy - as per the drive-first strategy. However, the waiting time at the last location of each service zone is distributed more evenly. The waiting time at the last location of a service zone is named herein the *after-zone waiting time*. In the advanced dynamic waiting strategy, the after-zone waiting time is just a portion of the longest feasible waiting. This portion can be defined in many different ways. We chose here to describe two possible ways that define different versions of the advanced dynamic waiting strategy.

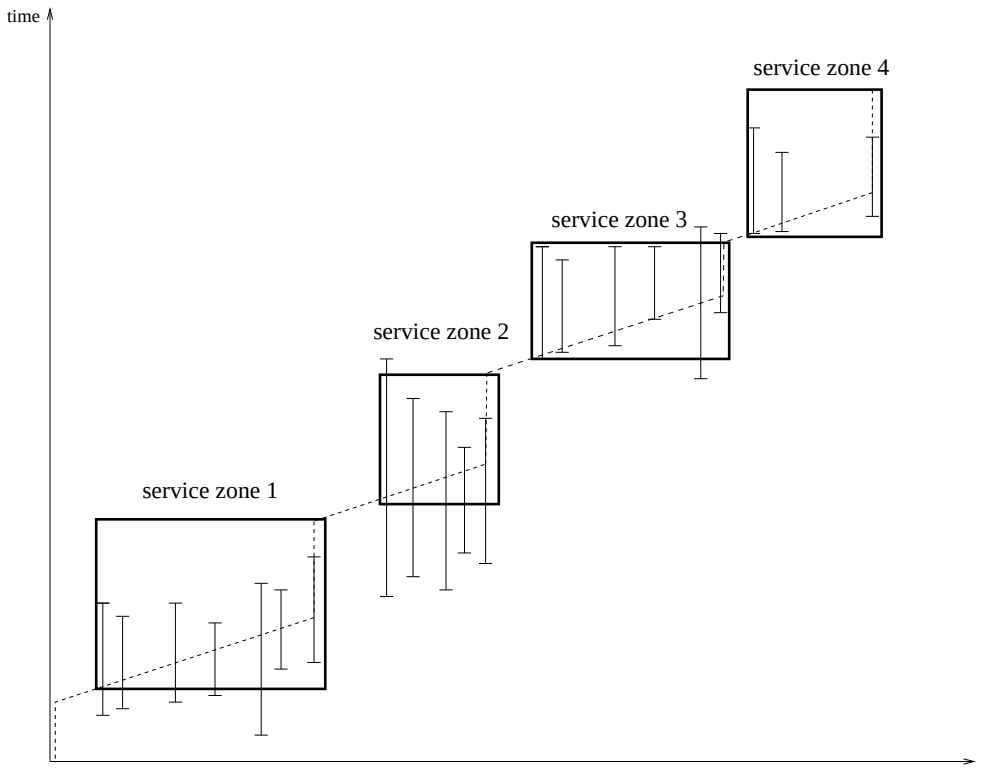


Figure 5.7: The route that consists of four service zones. The route locations are collinear. The scheduling is done by the advanced dynamic waiting.

Time span. Suppose that the number of future stop locations that will appear in a region is directly proportional to the duration of the observed future time interval. In terms of the route service zones, the assumption means that more new locations appear in the service zone in which the vehicle is scheduled to spend more time. This is an appropriate forecasting of future requests in problem instances with uniform distribution of stop locations in time and space. We design a waiting strategy based on this forecast. The after-zone waiting time is a function of the zone's time span. More precisely, the after-zone waiting time is directly proportional to the ratio of 'the width of the zone time span' to 'the sum of time spans of all service zones of the

route'.

Slack time. The slack time $\overrightarrow{S_i}$ (Definition 4.23 in Subsection 4.5.2) is the time available for the insertion of a new location after location i. Intuitively, if the slack times are large, new locations could be inserted. Thus, instead of forecasting a new location appearance (in the previous paragraph) one can turn attention toward the possibility of a new location being served by one vehicle. Assume that the more slack time is available in the route, the more likely is that new location could feasibly be inserted in the route. Based on this assumption, the after-zone waiting time in our advanced dynamic waiting will be determined. The after-zone waiting time should be directly proportional to the following ratio: 'the sum of slack times of all locations of the service zone' over 'the sum of slack times of all locations in the route'.

We have implemented the first version of the advanced dynamic waiting strategy.

5.7 Conclusions

The two simplest waiting strategies for scheduling a route are the drive-first and the wait-first strategies. Neither of them distribute waiting times good enough (in respect of future feasible location insertion) along the route. The dynamic waiting strategy and the advanced dynamic waiting strategy obtain a better arrangement of waiting times along a route.

Experimentally the other waiting strategies outperform the drive-first strategy. Unfortunately, the drive-first strategy is the easiest to implement while the others require additional effort. Specifically, in the case of the drive-first waiting strategy, the dynamic variables (Subsection 4.5.3 on page 106) have to be changed only after a location is inserted into a route, while in the case of the other waiting strategies the dynamic variables have to be updated regularly.

The dynamic waiting strategy and the advanced dynamic waiting strategies can do even better if further research is directed toward: (1) different ways of determining the after-zone wait, (2) definition of the base for service zones (another possible base is the minimal enclosing

circle, but intuitively the best choice would be the convex hull), (3) experimenting with different approaches when bounding the size of the service zone base, (4) measure of distance between a route and a request.

(The route service zones could also be used for inter-route improvements: by finding requests whose removal from one route, and insertion in another, would decrease the overlapping between the service zones of the two routes.)

Chapter 6

Dynamic problem, dynamic model, and dynamic solution method

6.1 Introduction

An empirical study (Chapter 7) have been conducted on the on-line system (described in Section 3.3). The aim was to test a number of the heuristic versions given in Chapter 4, and to evaluate effectiveness of the waiting strategies introduced in Chapter 5. The study results have motivated the design of the two-goal model and of the two-strategy heuristic described in this chapter.

The two-goal model is suitable for modelling a pure dynamic pickup and delivery problem with time windows. The model splits an objective of the corresponding static optimization problem into two goals. In the dynamic environment each goal is striven for over different parts of the future planning horizon. One goal dictates the objective in the near future, and the other deals with the distant future. The attempt is to reach a dynamic solution as close as possible to a good static solution (assuming the initial objective function of the static problem). A formal description of the dynamic optimization problem, the optimal dynamic solution, the optimal static solution, and the value of information are given in the first part of this chapter. These

formalizations are a logical introduction to the two-goal model.

The last section of this chapter introduces the two-strategy heuristic for solving a pure dynamic PDPTW. The idea was also motivated by the empirical study. The two-strategy heuristic is actually a framework in which any heuristic can be embedded. It can be used for solving the two-goal model.

6.2 Dynamic combinatorial optimization problem, dynamic model, and dynamic solution method

6.2.1 Combinatorial optimization problem

6.2.1.1 Static combinatorial optimization problem

A formal description of a combinatorial optimization problem is:

Definition 13 (Garey, Johnson (1979)) *A combinatorial optimization problem is either a minimization problem or a maximization problem that consists of the following three parts:*

(1) Set D of instances I.

(2) For each instance $I \in D$, a set of feasible solutions $X(I)$.

(3) Function f that assigns to each instance $I \in D$ and to each feasible solution $x \in X(I)$ a positive rational number $f(I, x)$ called the solution value for x.

Function f is the objective function *of the combinatorial optimization problem. If the problem is a minimization (maximization) problem, then an* optimal solution *for an instance $I \in D$ is a solution $x^* \in X(I)$ such that for all $x \in X(I)$, $f(I, x^*) \leq f(I, x)$ $(f(I, x^*) \geq f(I, x))$.*

The above definition captures all the aspects of a *static optimization problem* whose data is known before the process of problem solving starts. The problem is fully defined and modelled

with well defined objective function; its set of feasible solutions is determined and it remains fixed during the whole process of problem solving.

6.2.1.2 Dynamic combinatorial optimization problem

In dealing with a dynamic optimization problem the situation is quite different: some data may be known before the start of solving the problem, but much of the data will only be known during the process of solving the problem and during the process of implementing the solution. Three processes happen interchangeably or in parallel:

(1) the process of gathering and updating the problem data,

(2) the process of solving the problem, and

(3) the process of implementing the solution.

(In a static optimization problem all three processes exist, too, but they are strictly sequential and a researcher can concentrate on one process at a time.) The time interval corresponding to the duration of the process of implementing a solution is called *the problem service period.* The process of gathering and updating the data may start earlier than the start of the problem service period, and lasts till the end of the problem service period. The same is true for the process of solving the problem.

At each time instant t the dynamic optimization problem is determined by data known and valid at time t. Let us call this problem the *t-active problem* of the original dynamic problem. The t-active problem is like a snapshot of the dynamic problem at time t, of its data and current state of its implemented solution. We now introduce a formal description of the dynamic optimization problem:

Definition 14 *A dynamic combinatorial optimization problem is either a minimization problem or a maximization problem with the following parts:*

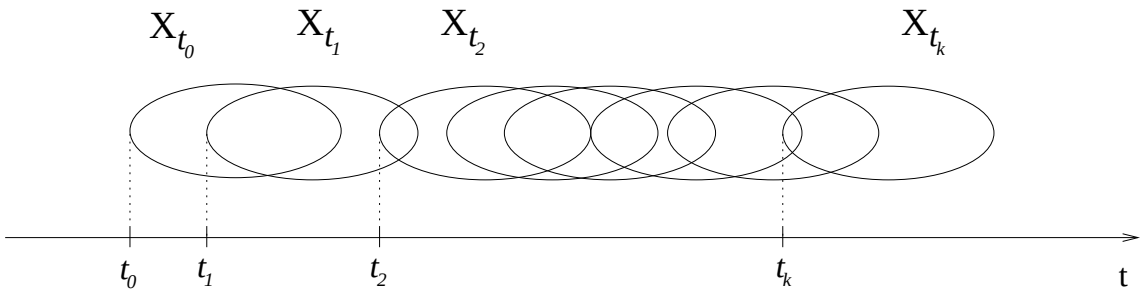


Figure 6.1: Active sets of feasible solutions at times: $t_0, t_1, t_2, ..., t_k$

(1) Set D of instances $\mathcal{I}$. Each instance $\mathcal{I}$ is a family $\{I_t : I_t$ is t-active instance of $\mathcal{I}\}$

(2.1) For each instance $\mathcal{I}$ there is a collection $\mathcal{X} = \{X_t\}$ of sets of feasible solutions.

(2.2) The set of feasible solutions X_t corresponds to the t-active instance I_t.

(2.3) Each element $x_t \in X_t$ is a feasible solution of the t-active instance.

(3) A t-active objective function f_t assigns to each instance I_t and to each feasible solution $x_t \in X_t$ a positive rational number $f_t(I_t, x_t)$ called the f_t-solution value for x_t.

The active objective function f_t can change over time, but it could, and usually would, be defined based on the objective function of the corresponding static optimization problem. The t-active objective function f_t determines the t-optimal solution $x_t^* \in X_t$.

Definition 15 *A t-solution x_t is a feasible solution of instance I_t proposed to be implemented at time t.*

Label $x_{t_i,t_{i+1}}$, $t < t_i < t_{i+1}$, is employed for the portion of x_t proposed to be implemented during the time interval (t_i, t_{i+1}).

Definition 16 *A solution implemented during time interval (t_i, t_{i+1}) is called the implemented solution, and it is labeled by $\dot{x}_{t_i,t_{i+1}}$.*

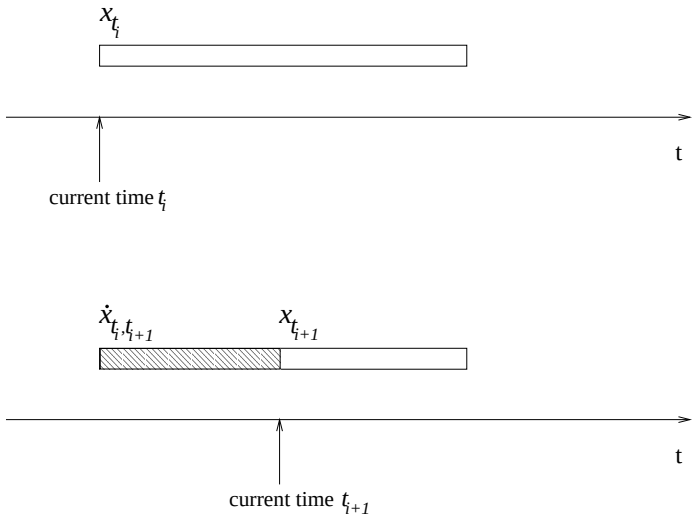


Figure 6.2: Dynamic optimization problem: proposed optimal solutions, and implemented solutions in time. At time t_i the solution x_{t_i} is proposed. The next solution is proposed at time t_{i+1}. The implemented solution $\dot{x}_{t_i,t_{i+1}}$ is equal to $x_{t_i,t_{i+1}}$ portion of x_{t_i}.

If the last solution found is x_{t_i}, and if the current time is t_{i+1}, the implemented solution $\dot{x}_{t_i,t_{i+1}}$ is the portion $x_{t_i,t_{i+1}}$ of x_{t_i}. The implemented solution $\dot{x}$ over the whole service period $(0, t_s)$ is a concatenation of the implemented solutions $\{\dot{x}_{0,t_1}, \dot{x}_{t_1,t_2}, \ldots, \dot{x}_{t_k,t_s}\}$, when the t-solutions are determined at time instants $\{0, t_1, \ldots, t_k\}$.

Therefore, a dynamic optimization problem has:

- changing active instances,
- changing sets of feasible solutions,
- changing t-solutions, and
- an implemented solution that 'grows',

which are the major distinct features of the dynamic problem compared to the static.

6.2.1.3 Optimal static solution and optimal dynamic solution

Suppose that one was attempting to solve a dynamic optimization problem, and had reached the end of the service period. All the problem data is known, and the resulting static problem instance I can be analyzed and solved. There is a set of feasible solutions X of I. The objective function f determines the optimal solution $x^* \in X$ of the static instance I.

Definition 17 *An optimal static solution of a dynamic instance $\mathcal{I}$ is an optimal solution $x^* \in X$ of static instance I generated from $\mathcal{I}$.*

Definition 18 *An optimal dynamic solution of a dynamic instance $\mathcal{I}$ is an optimal solution $\dot{x}^*$ of the instance $\mathcal{I}$.*

The following paragraphs describe a way of reaching an optimal dynamic solution.

Assume that we are solving instance $\mathcal{I}$ of a dynamic optimization problem. The service period is $(0, t_s)$. Suppose that the optimal solutions x_t^* are determined at equidistant time instants: $\{0, \delta, 2\delta, 3\delta, \ldots, (k-1)\delta\}$, where $(k-1)\delta < t_s \leq k\delta$. Implement these optimal solutions. For an implemented solution it is valid that $\dot{x}_{i\delta,(i+1)\delta} = x^*_{i\delta,(i+1)\delta}$, $i \in \{0, 1, \ldots, k-1\}$. The whole implemented solution is then $\dot{x}(\delta) = \{\dot{x}_{0,\delta}, \dot{x}_{\delta,2\delta}, \dot{x}_{2\delta,3\delta}, \ldots, \dot{x}_{(k-1)\delta,t_s}\}$. In many dynamic optimization problems there are a number of optimal solutions at each time instant. Thus, there are many possible instances $\{I^0_{i\delta}, I^1_{i\delta}, \ldots, I^j_{i\delta}, \ldots\}$ at time $i\delta$ depending on which optimal solution was proposed at time $(i-1)\delta$. Further, it is possible that for each active instance $I^j_{i\delta}$ many optimal solutions can be found. Therefore, the process of solving a dynamic optimization problem can be depicted by a tree shown in Figure 6.3. Each node of the tree is one active instance. The root of the tree is the initial active instance I_0 at the start of service period, *i.e.* at time 0. The outgoing edges of node I_0 represent different optimal solutions of the instance I_0 proposed at time 0. The i^{th} level of the tree contains all possible instances $\{I^0_{i\delta}, I^1_{i\delta}, \ldots, I^j_{i\delta}, \ldots\}$ at time $i\delta$. Each path of the tree from the root to one of the leaves represents one implemented

solution $\dot{x}(\delta)$. The path with the best implemented solution defines the best dynamic solution of the dynamic problem. We call this tree *the dynamic solution tree.*

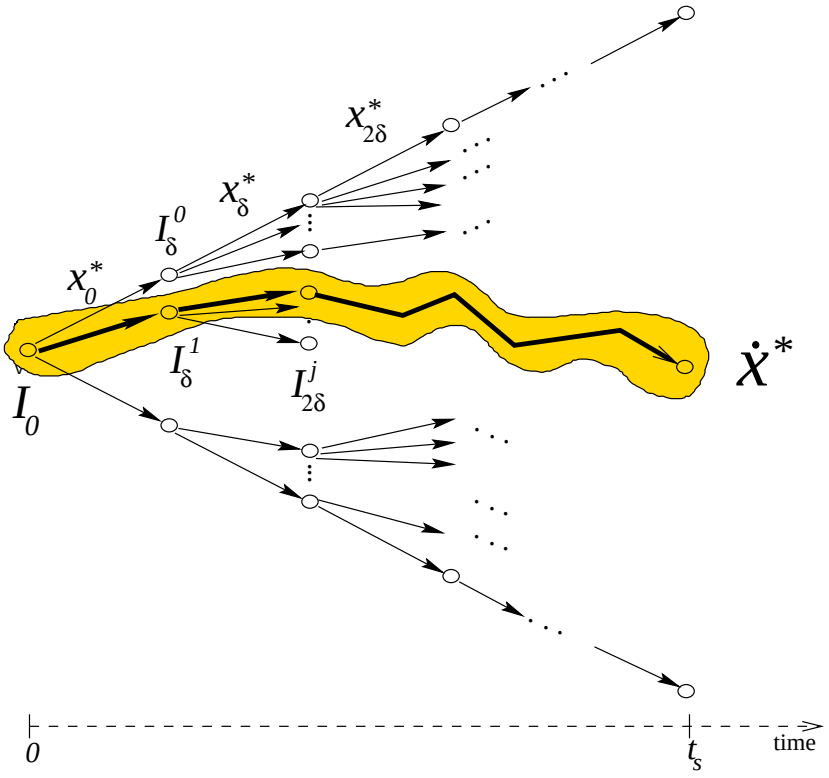


Figure 6.3: The dynamic solution tree of t-optimal solutions, and the dynamic optimal solution of a dynamic optimization problem.

Definition 19 *Assume that h is the objective function of a dynamic optimization problem, and that f is the objective function of the corresponding static optimization problem. An h-optimal dynamic solution (driven by f) is defined by:*

$$\dot{x}^*(h) = \lim_{\delta \to 0} f(\dot{x}(\delta, h)) \tag{6.1}$$

Functions h and f can be the same. The h-optimal dynamic solution is determined by the best path through the dynamic solution tree, when $\delta \to 0$.

Definition 20 *Assume that h is the objective function of a dynamic optimization problem, and that f is the objective function of the corresponding static optimization problem. An optimal dynamic solution (driven by f) is defined by:*

$$\dot{x}^* = \min_h f(\dot{x}^*(h)) \tag{6.2}$$

The function h^* for which the minimum is reached is called the optimal objective function of the dynamic problem driven by function f.

When $\delta \to 0$ the depth of the dynamic solution tree is unbounded. The number of possible nodes at each tree level can also be unbounded. In a combinatorial optimization problem the time variable is discretized, which generates the dynamic solution tree of finite depth, and with a finite number of nodes in one tree level. Let δ be equal to the time unit. If the time unit is small enough so that each data change is captured, the solution tree can produce an optimal dynamic solution. Otherwise the solution tree generates an upper bound of the optimal dynamic solution.

In Figure 6.4 a routing problem is shown in which four locations have to be visited. Locations 1 and 3 became known at time t', and locations 2 and 4 become known at time $t'' > t'$. The objectives of the dynamic problem and the corresponding static problem are equivalent: serve all locations and minimize route length. The dotted line represents the implemented solution when the drive-first strategy is used for solving the scheduling problem. The dashed line represents the implemented solution when the wait-first strategy is used. The wait-first strategy fails to serve location 4. There are many optimal dynamic solutions, three of which are shown by the bold lines in the figure. Each line between the lowest and the highest bold lines, parallel to the bold lines, is one optimal dynamic solution. (In an implementation, the number of optimal dynamic solutions depends on the value of time unit.) In this example the optimal static solutions are

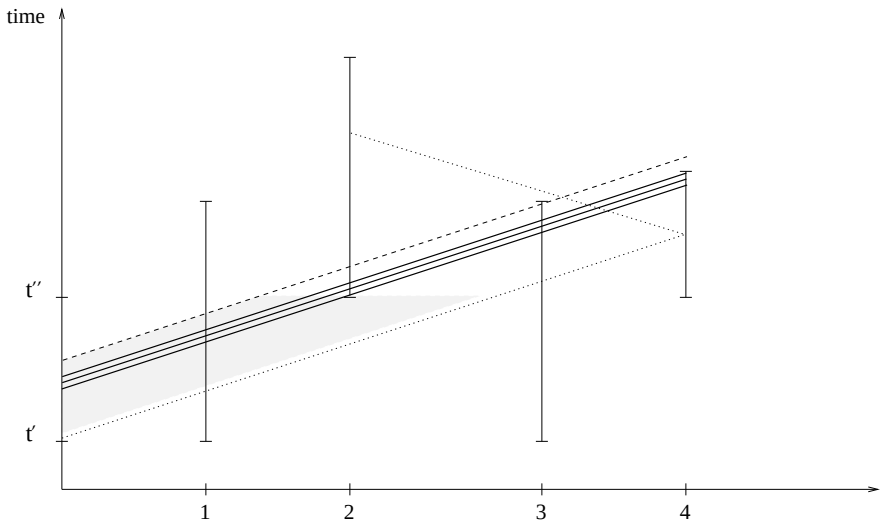


Figure 6.4: The bold lines show three optimal dynamic solutions. The optimal static solutions are equal to the optimal dynamic solutions.

equal to the optimal dynamic solutions. The shaded area shows the possible vehicle positions between time t' and t'', representing a number of different active instances between times t' and t''.

6.2.2 Value of information and solution method evaluation

Let f be an objective function of a static optimization problem. Consider a relation between the solution values of the optimal static solution and the optimal dynamic solution. This relation gives a measure of possible gain in solving a dynamic problem if all information about the problem have been known in advance.

Definition 21 *Assume that $\dot{x}^*$ is an optimal dynamic solution of dynamic problem instance $\mathcal{I}$, and that x^* is an optimal static solution of the same instance. The value of information V is:*

$$V = \frac{f(\dot{x}^*) - f(x^*)}{f(x^*)}. \tag{6.3}$$

The value of information is a measure of how close the optimal dynamic solution is to the optimal static solution. The solution value of the optimal static solution $f(x^*)$ is also called the posteriori lower bound of the optimal solution of the dynamic problem (Powell, Jaillet, Odoni, 1995).

When we know neither of the two optimal solutions we can (1) find lower or upper bounds on V by bounding each value, $f(\dot{x}^*)$ and $f(x^*)$, or (2) approximate V by solving the dynamic problem and the static problem using the same heuristic. When we approximate the value of information using a heuristic $\mathcal{H}$, we get *the value of information under heuristic* $\mathcal{H}$:

$$V(\mathcal{H}) = \frac{f(\dot{x}^{\mathcal{H}}) - f(x^{\mathcal{H}})}{f(x^{\mathcal{H}})} \tag{6.4}$$

In this context, the value of information is a measure of effectiveness of heuristic $\mathcal{H}$ for solving the dynamic problem.

In literature, measuring effectiveness of a method for solving dynamic problem appears in connection with problems whose static version is well-solvable optimally. In the area of vehicle routing, Powell, Jaillet, Odoni (1995) propose using the gap between $f(x^*)$ and $f(\dot{x}^{\mathcal{H}})$. Evaluations of on-line algorithms has been studied in other areas even more extensively: data structures (the list accessing problem, self-organizing data structures), computational topics (multiprocessor scheduling, memory paging in virtual memories, the k-server problem, the load balancing problem in parallel processing), game theory, and investment theory (assets allocation, trading). The traditional approach has been a distributional (or average-case) complexity, where one hypothesizes a distribution on events and studies the expected total cost or expected cost per event. In the last ten years the competitive analysis approach emerged which proposes measuring the quality of an on-line algorithm by comparing its performance to that of optimal off-line algorithm.

Definition 22 (Borodin, El-Yaniv (1998)) *On-line algorithm $\mathcal{A}$ is c-competitive if there is*

constant α such that for all dynamic problem instances $\mathcal{I}$,

$$f(\dot{x}^{\mathcal{A}}) \leq cf(x^*) + \alpha.$$

The problem of analyzing on-line algorithms can be viewed as a game between an on-line player and a malicious adversary. The on-line player runs the on-line algorithm on an input that is created by the adversary. The adversary, knowing the on-line algorithm, constructs the worst possible input so as to maximize the competitive ratio. Competitive analysis is comparable to the worst-case complexity of algorithms for solving static problems. Nevertheless, as cited by Borodin, Irani, Raghavan, Schieber (1992), practitioners voice reservations because competitive analysis, while being a useful mathematical tool, sometimes yields bounds that are unduly pessimistic. Furthermore, the analysis have shown as being unable to discern between two well-known algorithms (LRU(least-recently-used) and FIFO (first-in/first-out) for solving the paging problem) whose performances differ markedly in practice. Discussions have been raised on whether another approach should be used in evaluating on-line algorithms (Karp, 1992; Raghavan, 1992; Fiat, Woeginger, 1998).

We have chosen to use the value of information instead of the competitive analysis. The reasons are: (1) large-scale PDPTW cannot be solved optimally, and, (2) the already proven (Solomon, 1986) $\Omega(n)$ performance of many known off-line heuristics for solving static TSPTW.

6.2.3 Two-goal model

As mentioned earlier, research on dynamic problems, and the research on dynamic routing problems in particular, is fairly new, and many essential elements of the problem definition and solution methods are still unresolved. Goals and objective functions are often unclear. In the light of the value of information (as the evaluation technique of a method) possible goals in solving a dynamic optimization problem are discussed in this subsection. Assume that before the start of solving a dynamic problem, the best static solution $x^{\mathcal{M}}$ is known ($x^{\mathcal{M}}$ is the best solution that can be achieved when solving the corresponding static problem by method $\mathcal{M}$).

Label by $x^{\mathcal{M}}_{t_i,t_{i+1}}$ the portion of the static solution $x^{\mathcal{M}}$ that corresponds to the time interval (t_i, t_{i+1}). Solve the original dynamic optimization problem. Consider at time t two possible cases for the t-active instance I_t, $t < t_i < t_{i+1}$:

(1) The t-active instance I_t contains all data incorporated in $x^{\mathcal{M}}_{t_i,t_{i+1}}$, *i.e.*, all data incorporated in $x^{\mathcal{M}}_{t_i,t_{i+1}}$ are already known at time t.

(2) The t-active instance I_t does not contain all data incorporated in $x^{\mathcal{M}}_{t_i,t_{i+1}}$, *i.e.*, not all data incorporated in $x^{\mathcal{M}}_{t_i,t_{i+1}}$ are known at time t.

We distinguish between two cases regarding the position of time interval (t_i, t_{i+1}):

(a) Time interval (t_i, t_{i+1}) is close to the current time t.

(b) Time interval (t_i, t_{i+1}) is far away from the current time t.

Find the solution x_t of t-active instance of the problem, and assume that $x_{t_i,t_{i+1}}$ is a portion of it. Cases (1)(a) and (1)(b) are similar to solving static problem, while cases (2)(a) and (2)(b) are more interesting from the dynamic problem point of view. Case (2)(a) means that there is little time between now, time t, and the time t_i of possible implementation of the solution $x_{t_i,t_{i+1}}$. Consequently, there is a low probability that the solution $x_{t_i,t_{i+1}}$ changes because: there is no time for new data to appear, or there is a low probability that existing data will change, or there is not enough time to apply a solution method. Case (2)(b) represents the situation when there is a lot of time between now, time t, and the time t_i of possible implementation of the solution $x_{t_i,t_{i+1}}$. Data incorporated in the solution $x_{t_i,t_{i+1}}$ might change. New data may appear. To accommodate this data, it would be reasonable to maintain a special structure of the solution $x_{t_i,t_{i+1}}$.

This suggests a *two-goal approach* in solving a dynamic optimization problem, the result of which is *the two-goal model* of the problem. One goal is a *short-term goal* which builds the portion of the solution proposing to be implemented in the near future (Case (2)(a)). The other

goal is a *long-term goal* which builds the portion of the solution proposed to be implemented in the distant future (Case (2)(b)). Therefore the objective of a dynamic optimization model can be divided in:

Short-term goal: achieve low cost of the system operation, and

Long-term goal: maintain the system states such that the system can easily respond to future events.

As defined the short-term goal can be similar to the objective function of the corresponding static optimization problem, while long-term goal does not have to be similar. The best would be to strive for both goals simultaneously.

6.2.4 Dynamic on-line algorithm

An on-line algorithm is the most suitable for solving the two-goal model, because the solution procedure has to be applied repeatedly in time. *The dynamic on-line algorithm* is defined herein as a method for solving a dynamic optimization problem which allows the application of different modelling of t-active instances over time, and different solution procedures over time.
The dynamic on-line method for solving a dynamic optimization problem:

Step 0: Time $t = 0$.

Step 1: Take a snapshot of the problem at time t, and generate the t-active instance I_t.

Step 2: Choose an active problem model, an active objective function, and an active solution procedure.

Step 3: Solve the t-active instance by using the active solution procedure and the active objective function. The result is the solution x_t. Start implementation of x_t.

Step 4: At time $t = t + \Delta$ go to Step 1.

6.3 Two goals in solving dynamic PDPTW

6.3.1 Introduction

The dynamic PDPTW is characterized by requests that are coming-in during the whole service period. Data gathering, data update, and therefore the problem solving, can happen during the whole service period. The process of solving the problem comprises a series of decisions in choosing a vehicle that will serve a new request and in finding the best insertion of the request locations in to the vehicle's route. In order to find the best insertion slot for one location a measure has to be defined that evaluates the effectiveness of one insertion. In the case of static problem the measure is well defined by the objective function. In the case of dynamic problem the measure has to provide a way of finding a good current solution and, at the same time, has to allow building good solutions in the future (when solving future active instances). Defined in this section is one such measure in the form of dynamic utility function.

6.3.2 Two-goal model

The two-goal model of a dynamic routing problem is determined by: (1) the short-term goal that is to minimize route lengths, and (2) the long-term goal that is to maintain the routes during service period such that future request insertion is feasible, simple, and effective. The long-term goal in the dynamic routing problem with time windows is to maintain the route schedule such that the waiting periods and the slack times are grouped in larger units. The waiting time accumulation and distribution (along the route) can be handled by the dynamic route clustering (Section 5.4) and the advanced dynamic waiting (Section 5.6), respectively. The slack time accumulation can be achieved by using the utility function described in Subsection 6.3.3. The short-term goal in the dynamic routing problem with time windows could be to serve as many close locations as possible, despite possible fragmentation of the waiting periods and the slack times.

6.3.3 Slack time accumulation

In an on-line insertion procedure, the slack time accumulation can be achieved by using a time-utility function for evaluating the insertion of a new request. The goal is to sequence locations in the route such that the slack times are as large as possible, in order to allow more slack time for future location insertions. This calls for a restriction in the fragmentation of already existing slack times in the current routes.

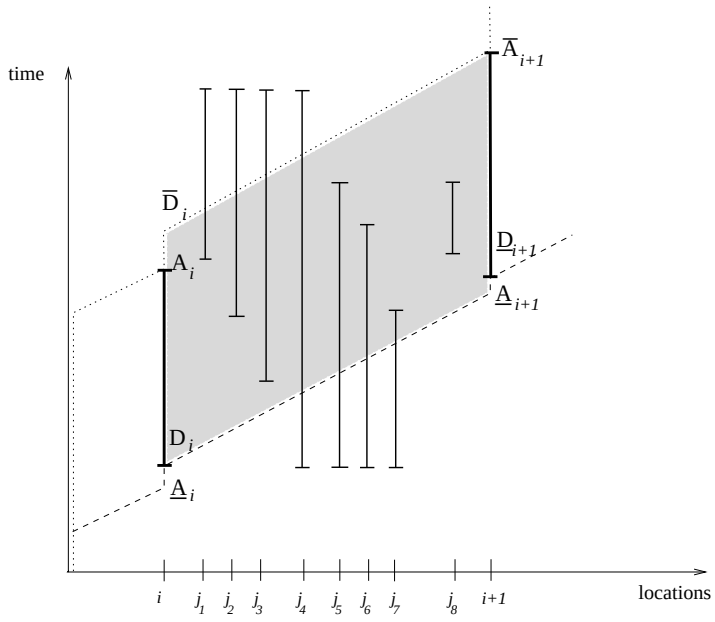


Figure 6.5: The slack time maintenance and the slack time fragmentation as a result of location j insertion between locations i and $i+1$. Eight examples of the time window of location j are shown. The slack time $\overrightarrow{S_i}\ (i+1)$ after location i, before insertion of new location, is shown by shaded area.

Suppose that we are inserting a new location j in the route $(1, 2, ..., i-1, i, i+1, ..., l)$, after location i. The variable $\overrightarrow{S_i}$ is the slack time available after location i. Extended labels for the slack times are introduced in order to differentiate between the slack time before the insertion of location j, denoted $\overrightarrow{S_i}\ (i+1)$, and the slack time after the insertion of location j after location i,

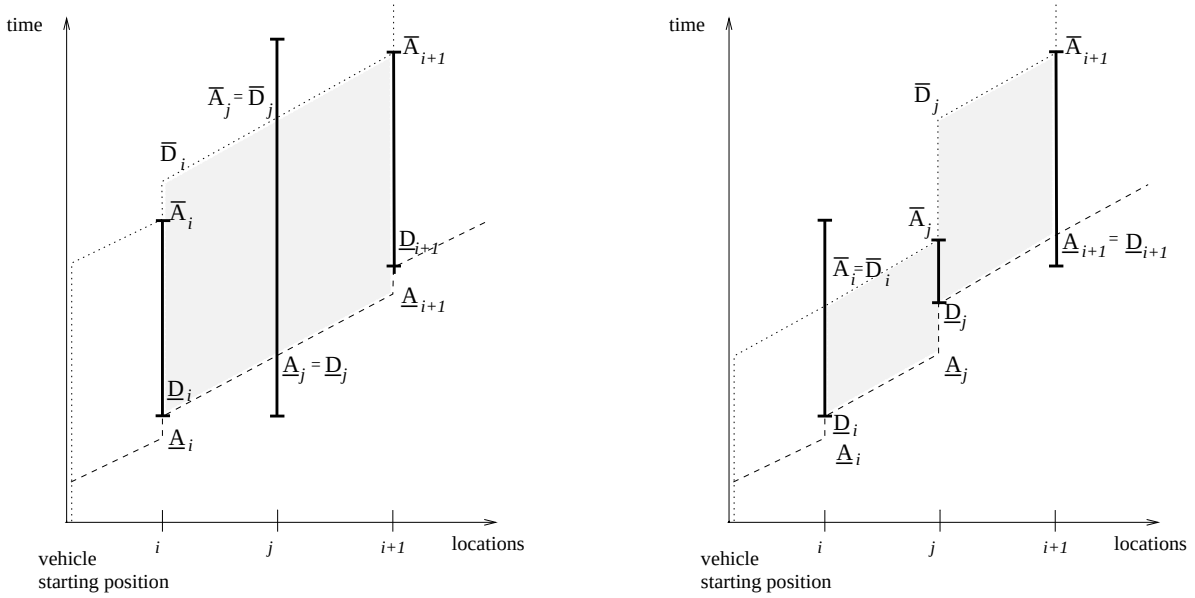


Figure 6.6: The slack time maintenance and the slack time fragmentation as a result of location j insertion between locations i and $i+1$. Two extreme cases: the left figure shows the unchanged slack time and the right figure shows the fragmented slack time as a result of location j insertion.

denoted $\overrightarrow{S}_i\ (j)$. In other words, $\overrightarrow{S}_i\ (j)$ is the slack time after location i if the location following i in the route is the location j. In Figure 6.5 eight possible time windows of location j are shown. The slack time $\overrightarrow{S}_i\ (i+1)$ is shown by the shaded area. The preferred location would be j_4, over locations j_1, j_7 and j_8. Figures 6.6 and 6.7 show four different cases of fragmentation and decrease of $\overrightarrow{S}_i$ due to the insertion of location j. Shaded areas illustrate the slack time $\overrightarrow{S}_i\ (j)$. The left graph of Figure 6.6 illustrates the situation where the time window of location j and the travel time constraints are such that the slack time $\overrightarrow{S}_i$ is neither fragmented nor decreased, *i.e.*, $\overrightarrow{S}_i\ (i+1) = \overrightarrow{S}_i\ (j) = \overrightarrow{S}_j\ (i+1)$. In the right graph of Figure 6.6 the slack time $\overrightarrow{S}_i$ is fragmented. The $\overrightarrow{S}_i$ is decreased from above due to insertion of j, and $\overrightarrow{S}_i\ (j) < \overrightarrow{S}_i\ (i+1)$. The slack time $\overrightarrow{S}_j\ (i+1) < \overrightarrow{S}_i\ (i+1)$. Figure 6.7 shows separately the slack time decrease from below, and the

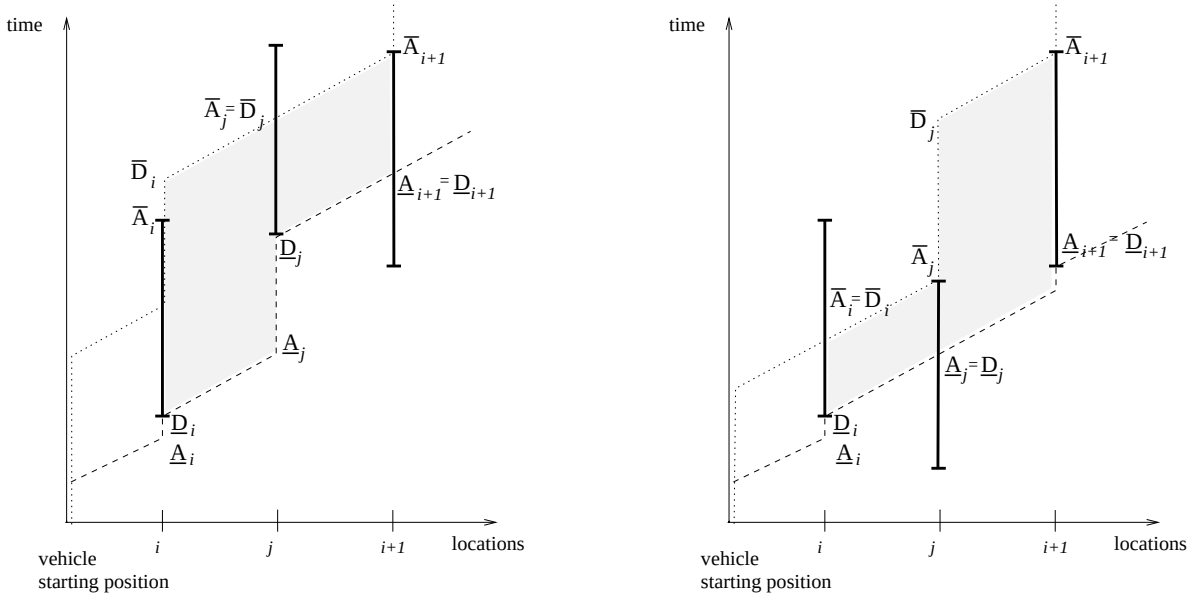


Figure 6.7: The slack time accumulation and the slack time fragmentation as a result of location j insertion between locations i and $i+1$. Two cases shows slack time decrease from below and from above.

decrease from above.

Assume that $\underline{D}_i$ is the earliest departure time from location i before the insertion of location j. Label by $A_j(\underline{D}_i)$ the arrival time at location j if i was left at $\underline{D}_i$. Let us introduce two functions, g' and g'' to represent the slack time decrease from below and the slack time decrease from above, respectively. From Figure 6.5 it is evident that function g' depends on the arrival time $A_j(\underline{D}_i)$ at location j. Compare $A_j(\underline{D}_i)$ to the time window $[a_j, b_j]$. The best situation is when the slack time is not reduced, and it happens when $A_j(\underline{D}_i) \geq a_j$. There is a symmetric case for the decrease of the slack time from above. Function g'' depends on the arrival time $A_j(\overline{D}_i)$ at location j, and the best situation is when $A_j(\overline{D}_i) \leq b_j$. The two functions are defined by:

$$g'(i, j, i+1) = \max(a_j - A_j(\underline{D}_i), 0) \tag{6.5}$$

$$g''(i,j,i+1) = \max(A_j(\overline{D}_i) - b_j, 0) \tag{6.6}$$

Functions g' and g'' have values zero when $A_j(\underline{D}_i) \in [a_j, b_j]$, and when $A_j(\overline{D}_i) \in [a_j, b_j]$, respectively. The maximal value for both functions is $\overrightarrow{S}_i\ (i{+}1)$. After reaching it, further strengthening of the time window constraints (by decreasing the size of the time window), or strengthening of the travel time constraints (by the decrease in the travel times) connected to location j, cause the infeasibility of sequence $(i,j,i+1)$ (see Figure 6.5). When g' reaches values larger than $\overrightarrow{S}_i\ (i+1)$ the serving of location $i+1$ becomes infeasible. When g'' reaches values larger than $\overrightarrow{S}_i\ (i{+}1)$ the serving of location j becomes infeasible. Hence, function g' is defined over interval $(a_j - \overrightarrow{S}_i\ (i+1), b_j)$, and function g'' is defined over interval $(a_j, b_j + \overrightarrow{S}_i\ (i+1))$. Formulas for new slack times, after insertion of location j between locations i and $i+1$, are:

$$\overrightarrow{S}_i\ (j) = \overrightarrow{S}_i\ (i+1) - g''(i,j,i+1) \tag{6.7}$$

$$\overrightarrow{S}_j\ (i+1) = \overrightarrow{S}_i\ (i+1) - g'(i,j,i+1) \tag{6.8}$$

The graphs of functions g' and g'' are given in Figure 6.8. The sum $g = g' + g''$ evaluates the total decrease of the slack times due to insertion of location j between locations i and $i+1$.

$$g(i,j,i+1) = g'(i,j,i+1) + g''(i,j,i+1) \tag{6.9}$$

Function g is a *time-utility function* that can be used as a measure of how good insertion of location j, with respect to accumulation (and fragmentation) of slack times. The objective is to minimize g.

6.3.4 Dynamic measure of location insertion

A utility function for the evaluation of inserting location j between locations i and $i+1$ could be a parameterized combination of two functions:

$$f = (1-\alpha)d(i,j,i+1) + \alpha g(i,j,i+1) \tag{6.10}$$

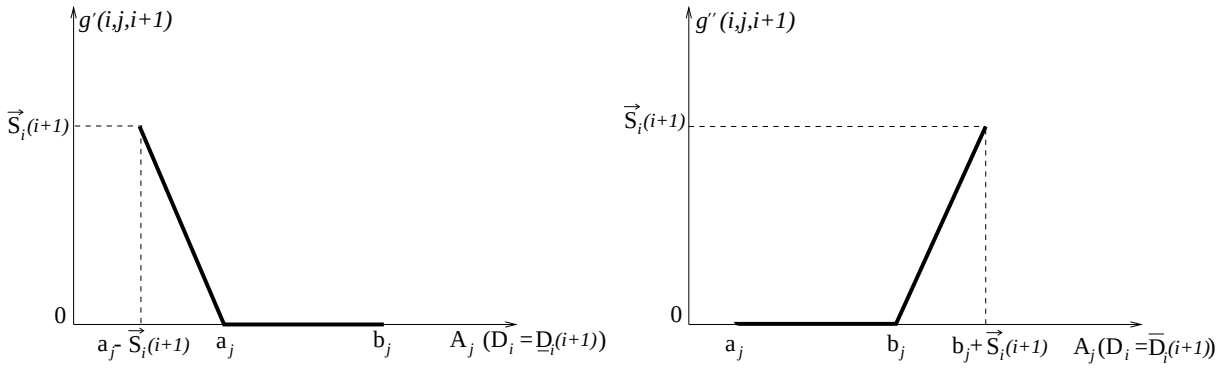


Figure 6.8: The problem is the dynamic pickup and delivery problem with time windows. Functions g' and g'' evaluate insertion of location j after location i in the route, regarding the pre-insertion and the post-insertion slack times. Both functions depend on the arrival time A_j at location j. In the case of g' it is assumed that i was left at the earliest departure time $\underline{D}_i(i+1)$ (the earliest departure time before insertion of j). In the case of g'' it is assumed that i was left at the latest departure time $\overline{D}_i(i+1)$ (the latest departure time before insertion of j). Both functions have the lowest values when A_j is within time window $[a_j, b_j]$, because in these situations the pre-insertion slack time $\vec{S}_i$, remains unchanged after j insertion. See Subsection 4.5.2 for the slack time definition.

where g is the time-utility function defined in Subsection 6.3.3

$$g(i,j,i+1) = g'(i,j,i+1) + g''(i,j,i+1)$$

and d is the route length increase due to j insertion. Functions d and g are also the measures of how good the sequence $(i,j,i+1)$ is, regarding the two goals described in Subsection 6.2.3. The function d reflects a short-term goal, and the function g reflects a long-term goal of the dynamic PDPTW. The utility function has to be minimized.

The utility function f would even better mirror a dynamic environment if parameter α is set to be time-dependent, *i.e.*, if it changes over time. In addition, parameter α could be different for each particular feasible insertion of a new location. Thus, we label such a *'dynamic parameter'* by $\alpha_{i,j,i+1}(t)$. Now, we have *the dynamic utility function* f:

$$f(t) = (1 - \alpha_{i,j,i+1}(t))d(i,j,i+1) + \alpha_{i,j,i+1}(t)g(i,j,i+1), \tag{6.11}$$

Because of the paradigm of 'short-term goal and long-term goal', and its connection with functions d and g, the 'dynamic parameter' has to be designed as a monotone decreasing function of the start of service T_j at new location j. Since T_j can only be calculated later, at the time of solving the scheduling problem for a fixed route, instead of T_j we use the earliest start of service time $\underline{T}_j$ at location j. The prospective value for $\underline{T}_j$ reflects prospective $(i,j,i+1)$ subsequence.

One possible $\alpha_{i,j,i+1}(t)$ is the distance, in time, from the current time t to the time of the earliest start of service $\underline{T}_j(i)$ at new location j, if j is inserted between i and $i+1$ at time t:

$$\alpha_{i,j,i+1}(t) = \underline{T}_j(i) - t.$$

To let one be the maximal value of α set:

$$\alpha_{i,j,i+1}(t) = \frac{\underline{T}_j(i) - t}{t_s - t}, \tag{6.12}$$

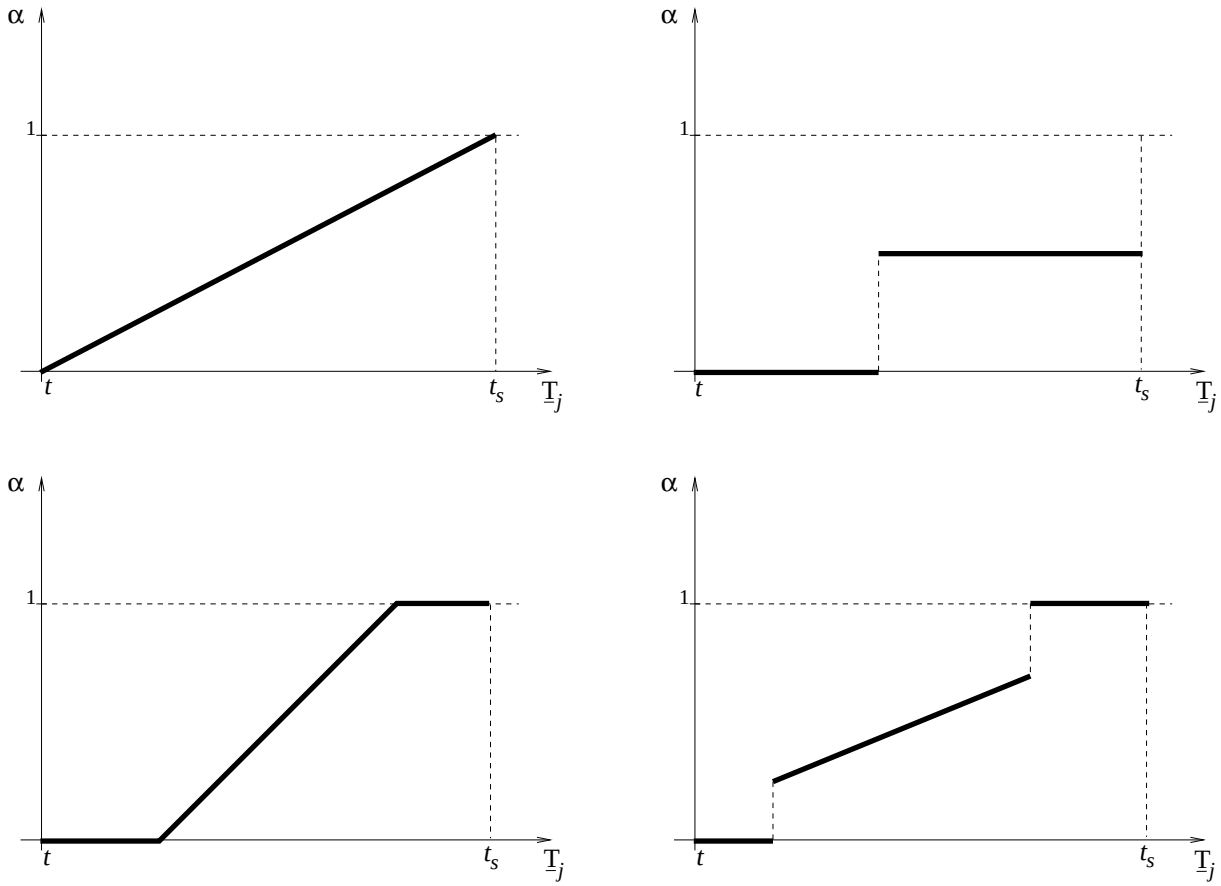


Figure 6.9: Different ways to define the 'dynamic parameter' α of the dynamic utility function $f(t)$. The figure shows four different graphs. The dynamic utility function $f(t)$ is aimed at pursuing two goals simultaneously in solving a dynamic routing problem. The utility function evaluates location j insertion between locations i and $i+1$ in time t. The 'dynamic parameter' α depends on the time distance between the current time t and the earliest start of service time $\underline{T}_j$ at location j. Time t_s is the end of the problem service period.

where t_s is the end of the problem service period. (See the first graph in Figure 6.9.) In order to avoid the steep slope of α later in time, the parameter can be set to:

$$\alpha_{i,j,i+1}(t) = \frac{\underline{T}_j(i) - t}{t_s}, \tag{6.13}$$

despite the fact that it does not reach the maximal value of one.

Three more examples of α graphs are given in Figure 6.9. The second graph reflects a two-goal model in which each of two goals are striven over one of two distinct time horizons. The third and the fourth graphs show combinations of the first two. The dynamic utility function $f(t)$ with such 'dynamic parameters' favors function d over function g for those insertions when a location is to be served in the near future, or function g over d otherwise.

The dynamic utility function $f(t)$ can be used as the t-active objective function f_t in the process of solving a dynamic optimization problem. Using of $f(t)$ calls for applying re-optimization during the whole service period, even when no new request appears, because of a change in the current time t. Note that since $f(t)$ is a measure of location insertion it can be used in any insertion procedure or generalized insertion procedure.

6.4 A dynamic on-line method: the two-strategy heuristic

6.4.1 Introduction

The two-strategy heuristic is a framework within which any known heuristic or method can be incorporated. It is based on three concepts:

- two-goal model of the dynamic optimization problem,
- two-strategy approach to problem solving, and
- the time horizons approach (Psaraftis, 1988) in solving a dynamic routing problems.

The two-strategy heuristic proposes solving dynamic PDPTW problem over two time horizons. The short-term time horizon covers the first sections of routes, and the long-term time horizon covers the rest of routes (or the whole routes). The strategies to be applied over each time horizon are called *the short-term strategy*, and *the long-term strategy*, respectively. (The first route section is the part of the route along which the vehicle will drive in the near future. Note that the first route section changes over time.)

Variations of the two-strategy heuristic are:

(1) The heuristic for solving the problem model with two different goals (the short-term goal, and the long-term goal) by using the same method for reaching both the goals. Each goal is to be striven for over one of the two time horizons: the short-term goal over the short-term time horizon, and the long-term goal over the long-term time horizon.

(2) The heuristic for solving the problem model with one goal, by using two different methods: one method, usually simple and fast, applied over the short-term time horizon, and the other method over the long-term time horizon.

(3) The heuristics for solving the problem model with two different goals, each one over one of the two time horizons, by using two different methods.

Heuristics (1) and (3) can be used for solving the two-goal model, while the dynamic utility function (Subsection 6.3.4) can be used within heuristic (2).

In solving the dynamic routing problem, the two-strategy heuristic can be used for solving one or both of the existing problems: routing and scheduling. An example of the two-strategy heuristic applied to the scheduling problem is the advanced dynamic waiting strategy (Section 5.6). The short-term goal is to serve all locations in the first route service zone as fast as possible, and it is realized by using the drive-first waiting strategy. The long-term goal is to distribute waiting times in larger portions along the route, which the advanced dynamic waiting does. In solving the routing problem, the short-term strategy could be a simple greedy strategy. It

can be insertion or re-optimization procedure. The long-term strategy can also be an insertion procedure or a re-optimization procedure. For the sake of simplicity, the following discussion is restricted to version of the two-strategy heuristic with:

- the long-term strategy being an insertion procedure, and
- the short-term strategy being a re-optimization procedure.

6.4.2 Time horizons

The sizes of the two time horizons and their mutual relation define variations of the two-strategy heuristic. The relation between the two time horizons could be: (1) the long-term horizon is the continuation of the short-term one, *i.e.*, the long-term time horizon starts where the short-term horizon ends, or (2) the short-term time horizon is embedded in the long-term horizon. The first is simpler and easier to implement, while the second (the embedded time-horizon structure) might achieve better solutions. We expect better solutions because the embedded time-horizon structure guarantees that the long-term strategy could influence the route section that belongs to the short-term time horizon. In other words, it allows the insertion of a new location in the first route section. Consecutive time horizons define the heuristic in which the first route section cannot be changed by the appearance of a new request.

The long-term horizon rolls. In the case when the long-term horizon covers the rest of the service period, the horizon still rolls because the start of the long-term horizon changes. The short-term time horizon can *roll* or *jump*. The short-term time horizon rolls if the procedure that implements the short-term strategy is called every K minutes and the short-term time horizon length is greater than K. The short-term time horizon jumps if its length is equal to K.

One of the simplest versions of the two-strategy heuristic, shown in Figure 6.10, has two consecutive time horizons: the jumping short-term time horizon and the rolling long-term time horizon. Horizontal intervals represent the time horizons. The figure shows changes over time in the time horizons' duration and position. The insertion procedure (long-term strategy) is called

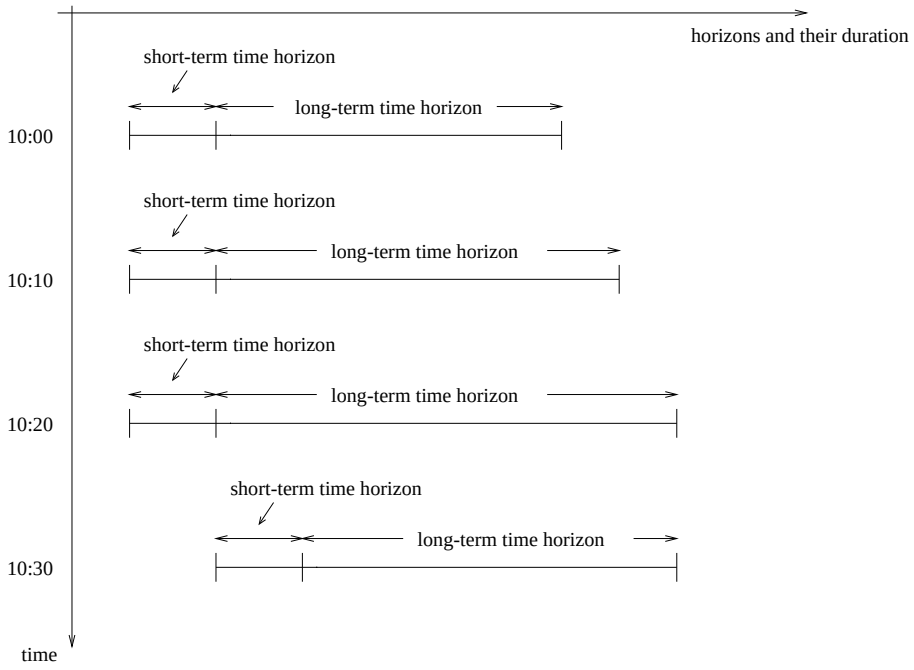


Figure 6.10: Two-strategy heuristic designed over two consecutive time horizons: the jumping short-term time horizon and the rolling long-term time horizon. The long-term time horizon starts where the short-term time horizon ends. The short-term horizon duration is 30 minutes, and the short-term strategy is applied every 30 minutes. The procedure of the long-term strategy is called every ten minutes, and does not influence the route section that belongs to the short-term horizon.

every ten minutes, and the re-optimization procedure (short-term strategy) is called every thirty minutes.

6.4.3 Two-strategy heuristic

Since the two-strategy heuristic is based on time horizons we present here the Psaraftis (1988) rolling horizons scheme modified so that can be applied to the dynamic PDPTW. (The Psaraftis algorithm was developed for solving the time-constrained problem of allocation of cargo ships to cargoes.)

Step 0: List M contains all unassigned requests.

Set time horizon length to L.

Set time F, $F < L$, that defines how often to call the assignment procedure.

Set $k = 1$, $t_k = 0$.

Step 1: Next horizon is $\mathcal{T} = (t_k, t_k + L)$.

Form set $S \subset M$ of requests eligible for assignment (*e.g.*, S contains all requests whose $b_{i^+} \in \mathcal{T}$).

Step 2: Assign requests from S to vehicles. This assignment determines the route in $(t_k, t_k + L + E)$. Value $E \geq 0$ due to the pairing constraints, the delivery time windows and the definition of the eligible request.

Step 3: Roll time horizon: Set $t_{k+1} = t_k + F$.

Update M by adding new requests.

Set $k = k + 1$ and go to Step 1.

Introduce notation L_l and F_l for the long-term horizon length and the frequency with which the insertion procedure (long-term strategy) is called. Similarly, L_s and F_s are the short-term

horizon length and the re-optimization (short-term strategy) frequency. The following is a version of the two-strategy heuristic for the dynamic routing problem.

Step 0: Initialization

List M contains all unassigned requests.

Set values for L_l, F_l, L_s, and F_l.

For the short-term horizon, set $k_s = 1$, $t_{k_s} = 0$.

For the long-term horizon, set $k_l = 1$, $t_{k_l} = 0$.

Do in parallel Steps 1, 2, and 3.

Step 1: The long-term strategy and the rolling long-term time horizon:

Step 1.0: Wait for time close to t_{k_l}.

Step 1.1: Next long-term horizon is $\mathcal{T}_l = (t_{k_l}, t_{k_l} + L_l)$.

Form set $S \subset M$ of requests eligible for assignment.

Step 1.2: Assign requests from S to vehicles by the long-term strategy $\mathcal{S}_l$.

Step 1.3: Roll long-term time horizon: set $t_{k_l+1} = t_{k_l} + F_l$.

Set $k_l = k_l + 1$.

Go to Step 1.0.

Step 2: The short-term strategy and the short-term time horizon:

Step 2.0: Wait for time close to t_{k_s}.

Step 2.1: Next short-term horizon is $\mathcal{T}_s = (t_{k_s}, t_{k_s} + L_s)$.

Step 2.2: Re-optimize route section within $\mathcal{T}_s$ by using the re-optimization procedure and strategy $\mathcal{S}_s$.

Step 2.3: Short-term time horizon jumps or rolls by setting $t_{k_s+1} = t_{k_s} + F_s$.

Set $k_s = k_s + 1$.

Go to Step 2.0.

Step 3: Handle new request appearance:

Step 3.0: Update M by adding new requests.

The short-term goal could be to serve as many route locations as possible close to the current vehicle position, and the method can be any fast greedy procedure (*e.g.*, such as the cheapest insertion). (The procedure could also consider the route locations that do not belong to the first route section.) For the long-term strategy, enough time is available for procedure application, and so even strategies as tabu search, whose running time directly determines the solution quality, can be successfully used. We note that to implement either of the two strategies any heuristic or optimization procedure can be used.

6.4.4 Advantages and variations

The two-strategy heuristic can be used for solving any dynamic routing problem, because a dynamic environment can be nicely modelled and utilized by the two goals and strategies. The two-strategy heuristic seems especially suitable for solving dynamic problems in which routes have to be planned for longer time periods. This is the case for the dynamic PDPTW, where the insertion of one request into a route can bind the route for quite some time into the future. The pairing constraints and the wide time windows can cause the vehicle schedule to be determined for the next six hours. In such a situation the two-goal concept will help to serve requests at lower costs while at the same time the distant (in time) route sections are maintained to allow future request insertions.

The two-goal model can be extended to a multi-goal model with many different combinations of short-term and long-term goals. The short-term goals can also cover sections other than the first route section. The two-strategy heuristic can appropriately extended to a multi-strategy heuristic. This methodology is useful in real-life applications which include: changing traffic conditions, different driver plans (lunch breaks, ending vehicle position), etc.

Chapter 7

Empirical study

7.1 Introduction

This chapter describes our empirical study that has led us through the process of engineering heuristics for solving the large-scale dynamic same-day pickup and delivery problem with time windows. The main results of the empirical study and the algorithm engineering are: (1) the recognition of the importance of schedule development for a fixed route in a dynamic routing problem, (2) the development of waiting strategies for scheduling a fixed route, and (3) motivation for the design of a two-goal model and a two-strategy heuristic for the dynamic time-constrained routing problem.

This empirical study contains experiments on the variations of an insertion heuristic described in Chapter 4, and experiments with all four waiting strategies outlined in Chapter 5. The most complex of the four waiting strategies, the advanced dynamic waiting, happens to be a version of the two-strategy heuristic of Section 6.4.

7.2 Test instances

Sets of test instances of the dynamic PDPTW have been generated by a request generator specially developed for this purpose. The instances have 100, 300, 500, or 1000 requests, and for each size ten instances have been generated. No requests are known in advance. The service area covers $30 \times 30\ km^2$, and the vehicle speed is $30\ km/h$. The service period lasts 10 or 13 hours, although the service period is simulated as a shorter time period in the majority of our experiments (for details see Subsection 7.2.3).

We have generated two sets of instances described in the next two subsections. In both sets, a request i is generated in the following manner:

(1) the time of request appearance is generated,

(2) the pickup and delivery locations are generated,

(3) the request service type is generated,

(4) the time windows are determined based on the time of request appearance and the service type:

(4.1) the start of the pickup time window a_{i^+} is the time of request appearance,

(4.2) the end of the delivery time window b_{i^-} is equal to the sum of a_{i^+} and the number of hours corresponding to the service type,

(4.3) the end of the pickup time window b_{i^+} and the start of the delivery time window a_{i^-} are calculated by formulas:

$$b_i^+ = b_i^- - t^*_{i^+,i^-}$$

$$a_i^- = a_i^+ + t^*_{i^+,i^-}$$

where $t^*_{i^+,i^-}$ is the minimal travel time between the pickup location i^+ and the delivery location i^-.

Request appearance is uniform throughout the problem service period. The positions of the stop locations are randomly chosen from the service area. The service types are: the one-hour service type, the two-hour service type, the four-hour service type, and the six-hour service type.

The fleet size is not bounded in all of our experiments. The initial fleet is either five vehicles for all problem sizes, or (20, 40, 60, 80) for instances with (100, 300, 500, 1000) requests, respectively. Regarding the depot, two sets of experiments were performed: no-depot experiments, and one-depot experiments with depot located at $(10km, 15km)$. In no-depot experiments, the starting positions of vehicles were taken from the set of pre-generated random positions.

7.2.1 Requests

7.2.1.1 First set of random instances

The first set of instances mirrors the distribution of real-life data collected in one medium-to-large courier company operating in Vancouver. There are 40 instances in the set. The service period lasts ten hours. There are three request service types and their distribution is: 20% of requests are one-hour requests, 30% are two-hour requests, 50% are four-hour requests. The request generator generates the one-hour requests within the first nine hours of the day, the two-hour requests within the first eight hours of the day, and the four-hour requests within the first six hours of the day. That is so because each request has to be served within the problem service period of ten hours. The distribution of request appearance by hour during one day is given in Table 7.1. The first percentage in columns three, four and five is the percentage with respect to all requests, and the second (in parenthesis) is the percentage of requests within one specific hour.

7.2.1.2 Second set of random instances

The second set of random instances contains also 40 instances. The service period lasts 800 minutes (more than 13 hours). There are three request service types, and their distribution

Request distribution.				
Hours	Problem sizes			
Real-life time	1h-req	2h-req	4h-req	Total
$7^{00} - 8^{00}$	1% (5%)	3% (25%)	10% (70%)	14%
$8^{00} - 9^{00}$	1% (10%)	3% (25%)	8% (65%)	12%
$9^{00} - 10^{00}$	1% (10%)	3% (25%)	8% (65%)	12%
$10^{00} - 11^{00}$	1% (10%)	3% (25%)	8% (65%)	12%
$11^{00} - 12^{00}$	2% (15%)	4% (25%)	8% (60%)	14%
$12^{00} - 13^{00}$	2% (15%)	4% (25%)	8% (60%)	14%
$13^{00} - 14^{00}$	3% (40%)	5% (60%)		8%
$14^{00} - 15^{00}$	3% (40%)	5% (60%)		8%
$15^{00} - 16^{00}$	6% (100%)			6%

Table 7.1: The first set of instances: distribution of request types during the day of service.

is: 30% of requests are two-hour requests, 40% are four-hour requests, and 30% are six-hour requests.

The majority of experiments are done on the first set of instances, while the second set is generated to test the waiting strategies in these settings.

7.2.2 Experiments with dynamic and with static problem instances

The first set of instances has also been tested as a set of instances of static PDPTW. The requests are the same, but it was assumed that all requests were known in advance. The purpose was to determine the value of information defined in Subsection 6.2.2, and to evaluate our heuristics for solving the dynamic PDPTW.

7.2.3 Speed of simulation

The speed of simulation is a parameter that indicates how much faster time passes in the computer (in our on-line system) than in the real-world. When the speed of simulation equals

Test instances and the speed of simulation.						
The on-line system without improvement procedure						
Service period	Speed of simulation	Service period simulated in	Number of requests			
			100	300	500	1000
600 min	60	$10min$	√	√		
600 min	30	$20min$			√	√
800 min	60	$\approx 13.3min$	√	√		
800 min	30	$\approx 26.6min$			√	√
The on-line system with tabu search improvement procedure						
Service period	Speed of simulation	Service period simulated in	Number of requests			
			100	300	500	1000
600 min	10	$1h$	√			
600 min	5	$2h$			√	
600 min	4	$2.5h$	√			
600 min	2	$5h$			√	

Table 7.2: Speed of simulation.

one, one real-life minute is simulated as one minute of computer time. When the speed of simulation equals 60, one minute is simulated as one second of computer time. This allows us to run more experiments in a shorter time. The speed of simulation is constrained to lower values by the time needed for running the solution procedure and the improvement procedures of the on-line system. The on-line insertion procedure described in Chapter 4, being a simple insertion procedure, allows a higher speed of simulation. Values 30 and 60 have been used. The on-line system with the tabu search improvement procedure must have the speed of simulation close to one, because tabu search is an open-ended heuristic (quality of solution depends on the running time of the tabu search procedure). Values from 2 to 10 have been employed, and each instance has been solved twice, in two different speeds of simulation, in order to compare the improvements in solution.

Table 7.2 shows all values of the speed of simulation used in our experiments. Note that the

The speed of simulation and the comparable instance sizes.						
The on-line system without improvement procedure						
Service period	Speed of simulation	Service period simulated in	Number of requests			
			100	300	500	1000
600 min	60	$10min$	600 req/h	1800 req/h		
600 min	30	$20min$			1500 req/h	3000 req/h
800 min	60	$\approx 13.3min$	450 req/h	1350 req/h		
800 min	30	$\approx 26.6min$			1125 req/h	2250 req/h
The on-line system with tabu search improvement procedure						
Service period	Speed of simulation	Service period simulated in	Number of requests			
			100	300	500	1000
600 min	10	$1h$	100 req/h			
600 min	5	$2h$			250 req/h	
600 min	4	$2.5h$	40 req/h			
600 min	2	$5h$			100 req/h	

Table 7.3: The increase in the speed of simulation is comparable to running the on-line system on larger problem instances. This table indicates the number of requests per hour in problem instances.

speed of simulation higher than one emulates experiments with higher number of requests per hour: when an experiment on a 300-request instance with a ten hour service period is performed with the speed of simulation set to 10, the service period lasts one hour of computer time. This experiment is similar to running an experiment on an instance that has 300 requests per hour. Consequently, the increase in the speed of simulation gives an indication of the sizes of problem instances that can be solved by one specific heuristic. Table 7.3 gives the sizes of instances, with respect to the speed of simulation, for the experiments that we have run.

7.3 The on-line insertion procedure and its variations

Many variations of the on-line insertion procedure described in Chapter 4 have been tested. The insertion heuristic is a sequential decision making heuristic (new requests are assigned to routes one at a time). A new request is assigned to the best active vehicle. There is an initial number of active vehicles at the start of the service period, and a new vehicle is activated only when a new request cannot be feasibly served by already active vehicles. The feasibility of the new request insertion into a route is tested gradually (see Section 4.3 on page 68). The objective is to minimize total route lengths, and it is enforced locally at each request insertion. Insertion of a request is done by the coupled insertion procedure described on page 86. The problem solutions are given in tables by the total route lengths and the number of vehicles used. The route lengths are shown in kilometers.

Details of the on-line insertion procedure used in experiments are given in Chapter 4 and in Chapter 5. There are three criteria defining a version of the heuristic:

Wait strategy:

- the drive-first waiting strategy,
- the wait-first strategy,
- the dynamic waiting strategy,
- the advanced dynamic waiting strategy.

Frequency of assignment, request sorting, request eligibility for assignment:

- a request is assigned immediately when it appears,
- requests are assigned every 15 minutes, requests are not sorted, and all new requests are assigned,
- requests are assigned every 15 minutes, requests are not sorted, and only requests with an impending end of the pickup time window are assigned,
- requests are assigned every 15 minutes, requests are sorted by the end of delivery time window, and all new requests are assigned,

- requests are assigned every 15 minutes, requests are sorted by the end of delivery time window, and only requests with an impending end of the pickup time window are assigned,
- requests are assigned every 15 minutes, requests are sorted by the difficulty degree defined in Equation 4.1, and all new requests are assigned,
- requests are assigned every 15 minutes, requests are sorted by the difficulty degree defined in Equation 4.1, and only requests with an impending end of the pickup time window are assigned,

Vehicle evaluation:

- evaluate all vehicles, unsorted,
- evaluate all vehicles, sorted,
- vehicles are examined in subsets, where one subset contains vehicles within the specified distance from the request,
- vehicles are examined in subsets, where the set of active vehicles is partitioned in the ten subsets, each containing 10% of the sorted vehicles.

In our experiments, we run many versions of the on-line insertion procedure. All combinations of the three criteria were tested: four different wait strategies; seven choices for requests sorting, eligibility and frequency of assignment; and four types of vehicle evaluations. The number of tested heuristics have gradually been decreased based on the solutions obtained.

7.3.1 Restricted vehicle evaluation

Restricted vehicle evaluation was aimed to allow quick decision making about the insertion of a new request. It restricts vehicle evaluation to a subset of active vehicles. For more details refer to Subsection 4.2.2. Tables in this subsection show the trade-off between the time to make decisions and the quality of the solutions. For 100-request and the 300-request instances, we observe the number of vehicles evaluated before a good vehicle is found to serve the new request. The percentage (over all active vehicles) representing this number is given in Table 7.4. The

Request is assigned to the vehicle found in the first k% of all vehicles.		
	Number of requests	
Evaluated vehicles	100	300
In sets, by distance	24.39%	26.93%
In sets, by percentage	7.79%	9.13%

Table 7.4: Best vehicles. Comparisons of two restricted vehicle evaluations.

The total route lengths, and the number of vehicles used.								
	Number of requests							
Evaluated vehicles	100		300		500		1000	
All, unsorted	2648.60	12.15	6197.80	23.7	9201.88	34.86	15139.52	54.56
In sets, by percentage	3488.82	13.83	8358.14	28.61	12275.79	39.9	20118.70	63.34
Deterioration	31.72%	13.83%	34.86%	20.72%	33.41%	14.46%	32.89%	16.09%

Table 7.5: The total route lengths and the number of vehicles show the trade-off between the time used for decision making for insertion of a new request, and the quality of the solution. In 'all, unsorted' the new request is assigned to the best vehicle, while in the 'in sets, by percentage' the new request is assigned to the best among the first 10% of sorted vehicles (see Table 7.6). The 'in sets, by percentage' represents a restricted vehicle evaluation policy.

table indicates that 'in sets, by percentage' strategy evaluates a smaller number of vehicles. Further experiments were done to evaluate deterioration in solution quality for the heuristic that evaluates all unsorted vehicles.

Restricted vehicle evaluation can shorten the decision making time because a vehicle to serve a new request is found after evaluating less than 10% of active vehicles (Table 7.6), but the solution is at least 30% worse (Table 7.5). Nevertheless, restricted evaluation can be used when quick decisions have to be made (like deciding whether a new request can be feasibly served by active vehicles). Further tuning of the measure used for distance evaluation between

New request is assigned to the vehicle found in the first k% of all vehicles.				
	Number of requests			
Evaluated vehicles	100	300	500	1000
All, unsorted	39.67%	44.30%	46.28%	46.67%
In sets, by percentage	7.79%	9.13%	9.26%	9.25%

Table 7.6: Best vehicles. Comparisons of two criteria for vehicle evaluation.

a vehicle and a request is recommended, because it can lessen the gap between the quality of solutions of the two criteria for vehicle evaluation.

In further experiments the assignment procedure evaluates all vehicles without sorting.

7.3.2 Waiting strategies

A waiting strategy is a way to derive a schedule in a dynamic PDPTW. Four waiting strategies are discussed in this thesis (Chapter 5), whose comparison is given in this subsection. Each waiting strategy has been tested seven times (over each problem instance) using seven heuristic versions that differ in request criteria (request sorting, eligibility and frequency of assignment).

Three sets of experiments were run. The first set was characterized with five initially active vehicles and with no depot. The second set of experiments has a larger number of initially active vehicles (20, 40, 60, 80 for 100-, 300-, 500-, 1000-request instances respectively). The third set of experiments introduces a depot. Each set of experiments is reported in a pair of tables. The first table of the pair reports on solution values (the total route lengths and the number of vehicles used), while the second table shows the improvement achieved by each waiting strategy compared to the drive-first strategy. (The drive-first waiting strategy has been the most commonly used strategy in a variety of vehicle routing studies.) Note that there is a row in the table that indicates the instance sizes. Each entry of this row shows two numbers: the total number of requests in the instance and the number of requests per hour of the corresponding fictive problem with the respect to the speed of simulation used.

The total route lengths, and the number of vehicles used. No-depot instances with five initially active vehicles.								
	Number of requests							
Wait strategies	100 (600/h)		300 (1800/h)		500 (1500/h)		1000 (3000/h)	
Drive-first	2822.96	11.43	6397.47	22.71	9208.96	34.1	15688.90	54.7
Wait-first	2563.57	13.38	6112.41	25.59	9246.90	37.26	14732.66	55.96
Dyn-wait	2587.45	12.35	6193.82	23.67	9325.38	35.06	15415.29	55.35
Adv-dyn-wait	2620.41	11.45	6087.52	22.82	9026.27	33.04	14721.21	52.25

Table 7.7: The total route lengths and the number of vehicles. Comparison of the four waiting strategies.

Improvements compared to the drive-first waiting strategy.								
	Number of requests							
Wait strategies	100 (600/h)		300 (1800/h)		500 (1500/h)		1000 (3000/h)	
Drive-first	2822.96	11.43	6397.47	22.71	9208.96	34.1	15688.90	54.7
Wait-first	2563.57	13.38	6112.41	25.59	9246.90	37.26	14732.66	55.96
Improvement	**9.19%**	-17.06%	**4.46%**	-12.68%	**-0.41%**	-9.27%	**6.10%**	-2.30%
Dyn-wait	2587.45	12.35	6193.82	23.67	9325.38	35.06	15415.29	55.35
Improvement	**8.34%**	-8.05%	**3.18%**	-4.23%	**-1.26%**	-2.82%	**1.74%**	-1.19%
Adv-dyn-wait	2620.41	11.45	6087.52	22.82	9026.27	33.04	14721.21	52.25
Improvement	**7.17%**	-0.17%	**4.84%**	-0.48%	**1.98%**	3.11%	**6.17%**	4.48%

Table 7.8: The total route length, and the number of vehicles. Improvement rows show improvements in the solution achieved by a waiting strategy compared to the drive-first waiting strategy.

The total route lengths, and the number of vehicles used. No-depot instances with 20, 40, 60, or 80 initially active vehicles.								
	Number of requests							
Wait strategies	100 (600/h)		300 (1800/h)		500 (1500/h)		1000 (3000/h)	
Drive-first	2825.66	11.21	6403.20	23.07	9410.63	34.46	15603.99	55.43
Wait-first	2571.03	13.06	6118.63	25.37	9176.99	37.63	14876.47	57.67
Adv-dyn-wait	2624.72	11.42	6178.74	22.73	9099.66	33.37	15417.30	54.13

Table 7.9: The total route lengths and the number of vehicles. Comparison of the three waiting strategies when the initial number of vehicles is increased.

The results of the first set of experiments are given in Table 7.7 and Table 7.8. Three proposed waiting strategies (the wait-first, the dynamic waiting, and the advanced dynamic waiting) have achieved better results on almost all instance sizes, when compared to solutions when the drive-first strategy was used. Out of the two dynamic waiting strategies, the advanced dynamic waiting generated shorter routes and used a smaller number of vehicles. Therefore, the dynamic waiting strategy was dropped from further testing. The drive-first and the wait-first strategies remained in experiments as two simple opposing strategies.

The next two tables (Table 7.9 and Table 7.10) give solutions when the initial number of active vehicles is increased to: 20 initially active vehicles for the 100-request instances, 40 initially active vehicles for the 300-request instances, 60 initially active vehicle for the 500-request instances, and 80 initially active vehicles for the 1000-request instances. The changes in the total route lengths and in the number of vehicles used were not significant.

Further analysis of log files and problem solutions raises the issue of switching from no-depot instances to one-depot instances. In no-depot experiments the starting vehicle positions are taken from pre-generated set of points, to render the solutions obtained by different heuristic versions comparable. Nevertheless, the situations were not rare in which different sets of vehicles (with different starting positions) were used by different heuristics. This can be seen by observing

Improvements compared to the drive-first waiting strategy.								
	Number of requests							
Wait strategies	100 (600/h)		300 (1800/h)		500 (1500/h)		1000 (3000/h)	
Drive-first	2825.66	11.21	6403.20	23.07	9410.63	34.46	15603.99	55.43
Wait-first	2571.03	13.06	6118.63	25.37	9176.99	37.63	14876.47	57.67
Improvement	**9.01%**	-16.50%	**4.44%**	-9.97%	**2.48%**	-9.20%	**4.66%**	-4.04%
Adv-dyn-wait	2624.72	11.42	6178.74	22.73	9099.66	33.37	15417.30	54.13
Improvement	**7.11%**	-1.87%	**3.51%**	1.47%	**3.30%**	3.16%	**1.20%**	2.35%

Table 7.10: The total route lengths and the number of vehicles. Comparison of waiting strategies when the initial number of vehicles is increased. Improvement rows show improvements in the solution achieved by a waiting strategy compared to the drive-first waiting strategy.

that when two different heuristics, $\mathcal{H}_1$ and $\mathcal{H}_2$, are used, a new request can be inserted into two different routes. This may render infeasible serving a new request by any of active vehicles in the case of $\mathcal{H}_1$, while the same new request can be served by an active vehicle in the case of $\mathcal{H}_2$. This makes the comparisons between strategies harder; therefore further experiments deal with one-depot instances. The requests remain the same, with only a few requests changed (service types) to make it feasible to serve them from the depot.

The depot is located at the coordinate $(10\,km, 15\,km)$ of $30 \times 30\,km^2$ service area. The initial number of active vehicles is: 20 for the 100-request instances, 40 for the 300-request instances, 60 for the 500-request instances, and 80 for the 1000-request instances. Solutions are given in Tables 7.11 and 7.12. The wait-first strategy and the advanced dynamic waiting achieved better solutions (shorter total route lengths) than the drive-first strategy. The heuristic with the wait-first strategy sometimes used more vehicles than the heuristic with the drive-first strategy. This can be explained by the decrease in slack times and the decrease in waiting periods along the route caused by increased waiting at the current vehicle position. Consequently, the number of feasible insertions of future locations will be smaller, which results in the need for more active

The total route lengths, and the number of vehicles used. One-depot instances with with 20, 40, 60, or 80 initially active vehicles.								
	Number of requests							
Wait strategies	100 (600/h)		300 (1800/h)		500 (1500/h)		1000 (3000/h)	
Drive-first	2641.11	14.23	6193.09	25.31	9596.96	43.24	14965.55	55.64
Wait-first	2453.61	13.76	5874.11	25.95	9034.82	41.31	14776.21	58.11
Adv-dyn-wait	2518.14	13.32	5882.26	24.92	8824.20	35.31	14539.68	53.69

Table 7.11: Total route lengths and the number of vehicles. Comparison of waiting strategies on the one-depot instances.

Improvements compared to the drive-first waiting strategy.								
	Number of requests							
Wait strategies	100 (600/h)		300 (1800/h)		500 (1500/h)		1000 (3000/h)	
Drive-first	2641.11	14.23	6193.09	25.31	9596.96	43.24	14965.55	55.64
Wait-first	2453.61	13.76	5874.11	25.95	9034.82	41.31	14776.21	58.11
Improvement	**7.10%**	3.30%	**5.15%**	-2.53%	**5.86%**	4.46%	**1.27%**	-4.44%
Adv-dyn-wait	2518.14	13.32	5882.26	24.92	8824.20	35.31	14539.68	53.69
Improvement	**4.66%**	6.39%	**5.02%**	1.54%	**8.05%**	18.34%	**2.85%**	3.50%

Table 7.12: The total route lengths and the number of vehicles. Comparison of three waiting strategies over one-depot instances. Improvement rows show improvements in the solution obtained by a waiting strategy compared to the drive-first waiting strategy.

The second set of random instances. Improvements compared to the drive-first waiting strategy.								
	Number of requests							
Wait strategies	100 (600/h)		300 (1800/h)		500 (1500/h)		1000 (3000/h)	
Drive-first	2999.38	12.20	7018.95	19.36	10315.30	23.68	16754.55	35.32
Wait-first	2827.16	11.42	6782.03	18.24	10443.81	25.76	17159.39	37.4
Improvement	**5.74%**	6.39%	**3.38%**	5.79%	**-1.25%**	-8.78%	**-2.42%**	-5.89%
Dyn-wait	2760.87	10.92	6733.88	17.92	10245.41	24.68	16883.68	36.28
Improvement	**7.95%**	10.49%	**4.06%**	7.44%	**0.68%**	-4.22%	**-0.77%**	-2.72%
Adv-dyn-wait	2796.51	11.20	6596.49	16.82	10037.76	23.07	16357.50	34.2
Improvement	**6.76%**	8.20%	**6.02%**	13.12%	**2.69%**	2.58%	**2.37%**	3.17%

Table 7.13: The total route length, and the number of vehicles. Improvement rows show improvements in the solution achieved by a waiting strategy compared to the drive-first waiting strategy.

vehicles. The number of vehicles in the advanced dynamic waiting is either close to the number of vehicles used in the drive-first strategy (Table 7.8) or better (Table 7.12).

It is worth observing that the wait-first strategy achieved total route lengths comparable to the advanced dynamic waiting. Therefore, when the number of vehicles is not important the wait-first strategy can be successfully used. Restricted to the one-depot experiments, which are the most fair settings for heuristic comparisons, the advanced dynamic waiting strategy showed the best quality solutions (considering the total route lengths and the number of vehicles used).

We performed experiments on the second set of random instances to check whether even wider time windows influence the comparative performance of the waiting strategies. The results given in Table 7.13, show that the comparable performance of the waiting strategies remains unchanged.

7.3.3 The two-strategy heuristic

The advanced dynamic waiting strategy happens to be a version of the two-strategy heuristic (Subsection 6.4.3) applied on solving the scheduling problem of the dynamic PDPTW. The short-term time horizon is equal to the time span of the first service zone of the route. The long-term time horizon covers the whole route. The short-term time horizon is embedded into the long-term time horizon. The short-term strategy is the drive-first strategy. The long-term strategy is to distribute the waiting times of the route along the route in several larger wait time intervals. The long-term strategy is resolved by advanced dynamic waiting. Consequently, the comparatively better performance of the advanced dynamic waiting strategy lends credibility to the two-strategy heuristic.

7.3.4 Request sorting, eligibility for assignment, and frequency of assignment

The criteria with regard to request sorting, eligibility and frequency of assignment are listed on page 156, and the overview is: a new request can be assigned to a vehicle as soon as it is known, or its assignment can be delayed. The heuristic, which delays the new request assignment, calls the assignment procedure every 15 minutes. In the meantime the on-line system collects new requests. During the assignment procedure all eligible requests are assigned to vehicles. Requests can also be sorted before they are considered for assignment. The following tables show the results for three sets of experiments:

- no-depot instances, five initially active vehicles, four waiting strategies,
- no-depot instances, more initially active vehicles, three waiting strategies,
- one-depot instances, more initially active vehicles, three waiting strategies.

Two tables are given for each set of experiments: the first table shows the total route lengths and the second shows the number of vehicles used. An entry of each table represents the average value

The total route lengths.					
Frequency of assignments, request sorting, request eligibility	Number of requests 100	300	500	1000	Average ranking
Immediate request assignment, do not sort	2637.57 (4)	6010.34 (1)	9256.37 (5)	12822.54 (1)	(2.75)
Delayed request assignment, do not sort, assign req. with close pck. deadline	2716.64 (7)	6386.80 (7)	9406.78 (7)	16091.81 (7)	(7)
Delayed request assignment, do not sort, assign all new requests	2635.52 (3)	6144.16 (4)	9119.81 (2)	15696.93 (6)	(3.75)
Delayed request assignment, sort requests by delivery deadline, assign req. with close pck. deadline	2663.74 (5)	6317.61 (6)	9387.54 (6)	15657.83 (5)	(5.5)
Delayed request assignment, sort requests by delivery deadline, assign all new requests	2614.24 (2)	6129.65 (3)	8878.20 (1)	15150.75 (2)	**(2)**
Delayed request assignment, sort requests by difficulty, assign req. with close pck. deadline	2671.15 (6)	6290.79 (5)	9226.17 (4)	15307.04 (4)	(4.75)
Delayed request assignment, sort requests by difficulty, assign all new requests	2601.31 (1)	6105.27 (2)	9138.24 (3)	15249.71 (3)	**(2.25)**

Table 7.14: The total route lengths. Comparisons of criteria regarding request selection, request sorting, and frequency of new request assignment. No-depot instances. Small number of initially active vehicles.

The number of vehicles used.					
	Number of requests				
Frequency of assignments, request sorting, request eligibility	100	300	500	1000	Average ranking
Immediate request assignment, do not sort	9.96 (1)	20.20 (1)	30.40 (1)	41.03 (1)	**(1)**
Delayed request assignment, do not sort, assign requests with close pickup deadline	13.11 (6)	24.98 (7)	37.02 (7)	59.32 (7)	(6.75)
Delayed request assignment, do not sort, assign all new requests	11.79 (3)	24.15 (4)	35.08 (4)	55.94 (4)	(3.75)
Delayed request assignment, sort requests by delivery deadline, assign requests with close pickup deadline	13.10 (5)	24.80 (6)	36.10 (6)	59.03 (6)	(5.75)
Delayed request assignment, sort requests by delivery deadline, assign all new requests	12.08 (4)	23.75 (3)	34.40 (2)	55.26 (3)	(3)
Delayed request assignment, sort requests by difficulty, assign requests with close pickup deadline	13.25 (7)	24.47 (5)	35.99 (5)	56.60 (5)	(5.5)
Delayed request assignment, sort requests by difficulty, assign all new requests	11.77 (2)	23.55 (2)	35.06 (3)	54.77 (2)	**(2.25)**

Table 7.15: The number of vehicles used. Comparisons of criteria regarding request selection, request sorting and frequency of request assignment. No-depot instances. Small number of initially active vehicles.

The total route lengths.					
Frequency of assignments, request sorting, request eligibility	Number of requests				Average ranking
	100	300	500	1000	
Immediate request assignment, do not sort	2652.14 (3)	6145.39 (2)	9218.91 (2)	13550.05 (1)	**(2)**
Delayed request assignment, do not sort, assign req. with close pck. deadline	2745.98 (7)	6397.96 (7)	9395.86 (7)	16306.44 (7)	(7)
Delayed request assignment, do not sort, assign all new requests	2665.20 (4)	6168.44 (3)	9330.84 (5)	15412.42 (4)	(4)
Delayed request assignment, sort requests by delivery deadline, assign req. with close pck. deadline	2693.24 (5)	6334.08 (6)	9370.78 (6)	15314.91 (3)	(5)
Delayed request assignment, sort requests by delivery deadline, assign all new requests	2642.04 (2)	6178.65 (4)	9250.19 (3)	15598.35 (5)	(3.5)
Delayed request assignment, sort requests by difficulty, assign req. with close pck. deadline	2719.90 (6)	6291.62 (5)	9272.61 (4)	15773.80 (6)	(5.25)
Delayed request assignment, sort requests by difficulty, assign all new requests	2598.10 (1)	6118.52 (1)	8764.46 (1)	15138.81 (2)	**(1.25)**

Table 7.16: The total route lengths. Comparisons of criteria regarding request selection, request sorting and frequency of request assignment. More initially active vehicles.

The number of vehicles used.					
	Number of requests				
Frequency of assignments, request sorting, request eligibility	100	300	500	1000	Average ranking
Immediate request assignment, do not sort	9.77 (1)	20.80 (1)	30.23 (1)	44.11 (1)	**(1)**
Delayed request assignment, do not sort, assign requests with close pickup deadline	12.55 (6)	24.77 (6)	36.57 (6)	60.49 (7)	(6.25)
Delayed request assignment, do not sort, assign all new requests	11.60 (2)	23.87 (3)	36.09 (4)	56.59 (3)	(3)
Delayed request assignment, sort requests by delivery deadline, assign requests with close pickup deadline	13.10 (7)	24.83 (7)	37.17 (7)	57.43 (5)	(6.5)
Delayed request assignment, sort requests by delivery deadline, assign all new requests	11.76 (3)	23.97 (4)	35.80 (3)	57.27 (4)	(3.5)
Delayed request assignment, sort requests by difficulty, assign requests with close pickup deadline	12.53 (5)	24.43 (5)	36.23 (5)	59.47 (6)	(5.25)
Delayed request assignment, sort requests by difficulty, assign all new requests	11.97 (4)	23.40 (2)	33.97 (2)	54.83 (2)	**(2.5)**

Table 7.17: The number of vehicles used. Comparisons of criteria regarding request selection, request sorting and frequency of request assignment. More initially active vehicles.

The total route lengths.					
Frequency of assignments, request sorting, request eligibility	Number of requests				Average ranking
	100	300	500	1000	
Immediate request assignment, do not sort	2494.73 (2)	5899.14 (2)	8954.05 (2)	13482.34 (1)	**(1.75)**
Delayed request assignment, do not sort, assign req. with close pck. deadline	2612.59 (7)	6143.10 (7)	9362.68 (7)	15578.08 (7)	(7)
Delayed request assignment, do not sort, assign all new requests	2462.80 (1)	5940.48 (4)	9132.09 (3)	15052.15 (5)	(3.25)
Delayed request assignment, sort requests by delivery deadline, assign req. with close pck. deadline	2578.26 (5)	6075.13 (6)	9199.73 (4)	14900.50 (4)	(4.75)
Delayed request assignment, sort requests by delivery deadline, assign all new requests	2511.83 (3)	5900.79 (3)	9251.57 (6)	14591.60 (3)	(3.75)
Delayed request assignment, sort requests by difficulty, assign req. with close pck. deadline	2582.74 (6)	6033.13 (5)	9245.28 (5)	15135.82 (6)	(5.5)
Delayed request assignment, sort requests by difficulty, assign all new requests	2520.40 (4)	5890.31 (1)	8918.54 (1)	14582.89 (2)	**(2)**

Table 7.18: The total route lengths. Comparisons of criteria regarding request selection, request sorting and frequency of request assignment. More initially active vehicles. One-depot instances.

The number of vehicles used.					
Frequency of assignments, request sorting, request eligibility	Number of requests				Average ranking
	100	300	500	1000	
Immediate request assignment, do not sort	12.64 (1)	21.30 (1)	36.07 (1)	48.13 (1)	**(1)**
Delayed request assignment, do not sort, assign requests with close pickup deadline	14.07 (5)	26.88 (7)	42.49 (6)	58.87 (7)	(6.25)
Delayed request assignment, do not sort, assign all new requests	13.92 (4)	26.43 (6)	40.45 (5)	57.55 (5)	(4)
Delayed request assignment, sort requests by delivery deadline, assign requests with close pickup deadline	14.27 (6)	25.43 (2)	40.02 (3)	56.90 (4)	(3.75)
Delayed request assignment, sort requests by delivery deadline, assign all new requests	13.84 (3)	25.79 (4)	42.57 (7)	55.18 (2)	(4)
Delayed request assignment, sort requests by difficulty, assign requests with close pickup deadline	14.33 (7)	26.31 (5)	40.13 (4)	58.57 (6)	(5.5)
Delayed request assignment, sort requests by difficulty, assign all new requests	13.33 (2)	25.62 (3)	37.95 (2)	55.50 (3)	**(2.5)**

Table 7.19: The number of vehicles used. Comparisons of criteria regarding request selection, request sorting and frequency of request assignment. More initially active vehicles. One-depot instances.

over 30 experiments (10 instances, 3 waiting strategies). (The three waiting strategies are: the drive-first strategy, the wait-first strategy, and the advanced dynamic waiting.) In each heuristic version, all (unsorted) active vehicles are evaluated during a request assignment. Compare the performance of seven heuristic versions which differ among themselves by the criterion regarding requests. Besides showing the total route lengths and the number of vehicles used, the tables rank the seven heuristics (given in parenthesis). The last column in each table represents the average ranking over all problem sizes. The top two ranks in each table are specified in boldface.

Intuitively, one expects a heuristic that postpones the assignment of easy requests, leaving more space and time for the assignment of new difficult requests, to perform better than the one that assigns all requests. But the experiments show otherwise: the better solutions were achieved when all new requests are assigned, without postponing assignment of easy requests. The two best criteria regarding request sorting, eligibility and frequency of assignment are:

- immediate request assignment at the time of appearance of a new request,
- delayed request assignment of all new requests collected since last assignment, in combination with sorting of request by degree of difficulty.

7.4 How good could gradual feasibility be?

The on-line insertion procedure determines the best coupled insertion of the two stop locations of a new request by examining all possible insertions in all the active routes. Prior to enlisting and evaluating feasible insertion slots, the feasibility of serving the new request by a route is examined. For details refer to the Section 4.3. The following tables and flow charts depicts how good a gradual feasibility testing could be. The analyzed experimental data cover twenty instances, ten for each of the two sizes: 100 requests, and 300 requests. For each test instance six experiments were run with six different heuristic versions (three waiting strategies, and two request criteria). Table 7.20 gives total number of evaluations of non-empty routes for serving a

Total number of evaluations of non-empty routes.				
Req	Frequency of assignments, request sorting, request eligibility	Wait strategy		
		Drive-first	Wait-first	Adv-dyn-wait
100	Immediate request assignment, do not sort	890.50	761.80	714.20
	Delayed request assignment, sort requests by difficulty, assign all new requests	869.50	788.89	749.20
300	Immediate request assignment, do not sort	4546.40	4442.60	4059.00
	Delayed request assignment, sort requests by difficulty, assign all new requests	4697.60	4687.60	4317.78

Table 7.20: Total number of evaluations of non-empty routes for serving a new request.

new request. The following three tables show failures of some of the feasibility tests: the direct-path test, lower bound on the number of vehicles, and separate location insertions. Each table entry represents percentage of failures, while the last column give average percentages. The percentages are relative to the total number of calls to the gradual feasibility testing routine. The routine is not called if the route is empty. The tables give the resulting number of test failures for each of six heuristic versions.

The failure of the direct-path test means that the direct path connecting the current vehicle position, the pickup location, and the delivery location is not feasible, *i.e.*, the request cannot be served by the vehicle. The average of failures is around 11%. Note that delayed request assignment causes a larger number of failures in the direct-path test.

The test for a lower bound on vehicles finds the lower bound on the number of vehicles needed to serve the new request and the unserved locations (of the route evaluated to serve the new request). A lower bound greater than one, means that the test failed, implying that the new

Direct path is non-feasible.					
Req	Frequency of assignments, request sorting, request eligibility	Wait strategy			Prcnt
		Drive-first	Wait-first	Adv-dyn-wait	
100	Immediate request assignment, do not sort	10.65%	9.07%	10.54%	10.09%
	Delayed request assignment, sort requests by difficulty, assign all new requests	15.02%	13.41%	13.12%	13.85%
	Average	12.83%	11.24%	11.83%	*11.97%*
300	Immediate request assignment, do not sort	11.10%	9.65%	10.47%	10.41%
	Delayed request assignment, sort requests by difficulty, assign all new requests	13.97%	12.62%	13.50%	13.36%
	Average	12.53%	11.14%	11.99%	*11.89%*
	Average				***11.93%***

Table 7.21: The failures of the direct-path test: the direct path (vehicle position, pickup location, delivery location) is not feasible. The percentages are calculated with respect to the number of evaluations of non-empty routes.

Lower bound on the number of vehicles is greater than one.					
Req	Frequency of assignments, request sorting, request eligibility	Wait strategy			Prcnt
		Drive-first	Wait-first	Adv-dyn-wait	
100	Immediate request assignment, do not sort	41.56%	47.66%	51.85%	47.02%
	Delayed request assignment, sort requests by difficulty, assign all new requests	36.70%	45.58%	42.82%	41.70%
	Average	39.13%	46.62%	47.33%	*44.36%*
300	Immediate request assignment, do not sort	47.34%	48.12%	54.04%	49.84%
	Delayed request assignment, sort requests by difficulty, assign all new requests	39.49%	39.89%	43.50%	40.96%
	Average	43.41%	44.01%	48.77%	*45.40%*
	Average				***44.88%***

Table 7.22: The vehicle-lower-bound test failures: the lower bound on the number of vehicles needed to serve the new request and the unserved locations of the route is greater than one. This failure is a proof that the new request cannot be served by this vehicle. The percentages are calculated with respect to the number of evaluations of non-empty routes.

Failure of the separate insertion of some of the two stop locations.					
Req	Frequency of assignments, request sorting, request eligibility	Wait strategy			Prcnt
		Drive-first	Wait-first	Adv-dyn-wait	
100	Immediate request assignment, do not sort	16.87%	29.78%	22.85%	23.17%
	Delayed request assignment, sort requests by difficulty, assign all new requests	15.79%	25.95%	20.65%	20.80%
	Average	16.33%	27.87%	21.75%	*21.98%*
300	Immediate request assignment, do not sort	30.40%	37.29%	34.03%	33.91%
	Delayed request assignment, sort requests by difficulty, assign all new requests	27.26%	34.50%	30.15%	30.64%
	Average	28.83%	35.90%	32.09%	*32.27%*
	Average				***27.13%***

Table 7.23: 'Separate insertion of the two stop locations': test failures.

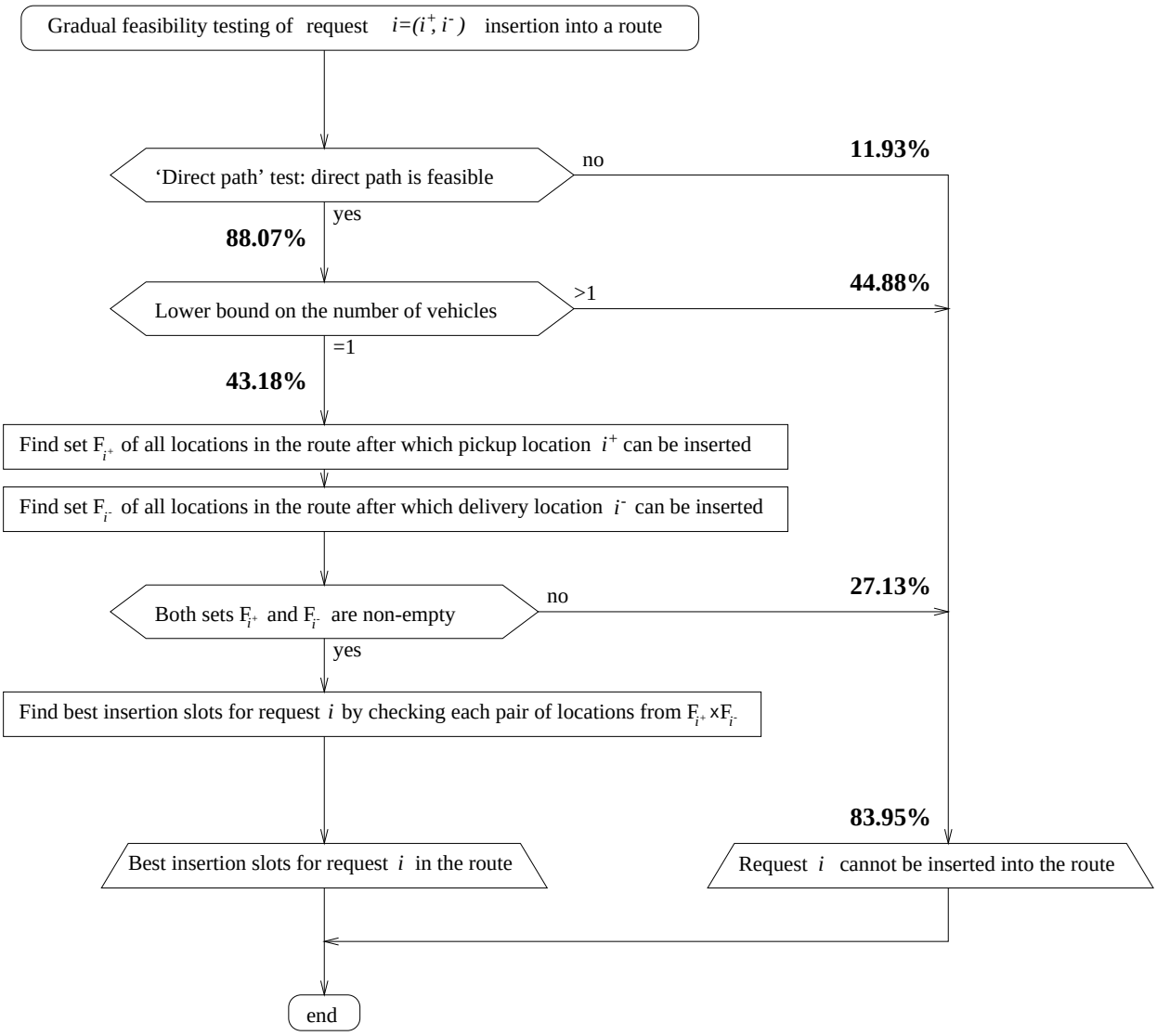


Figure 7.1: Percentages of flow through the algorithm of gradual feasibility testing of a new request being served by one route.

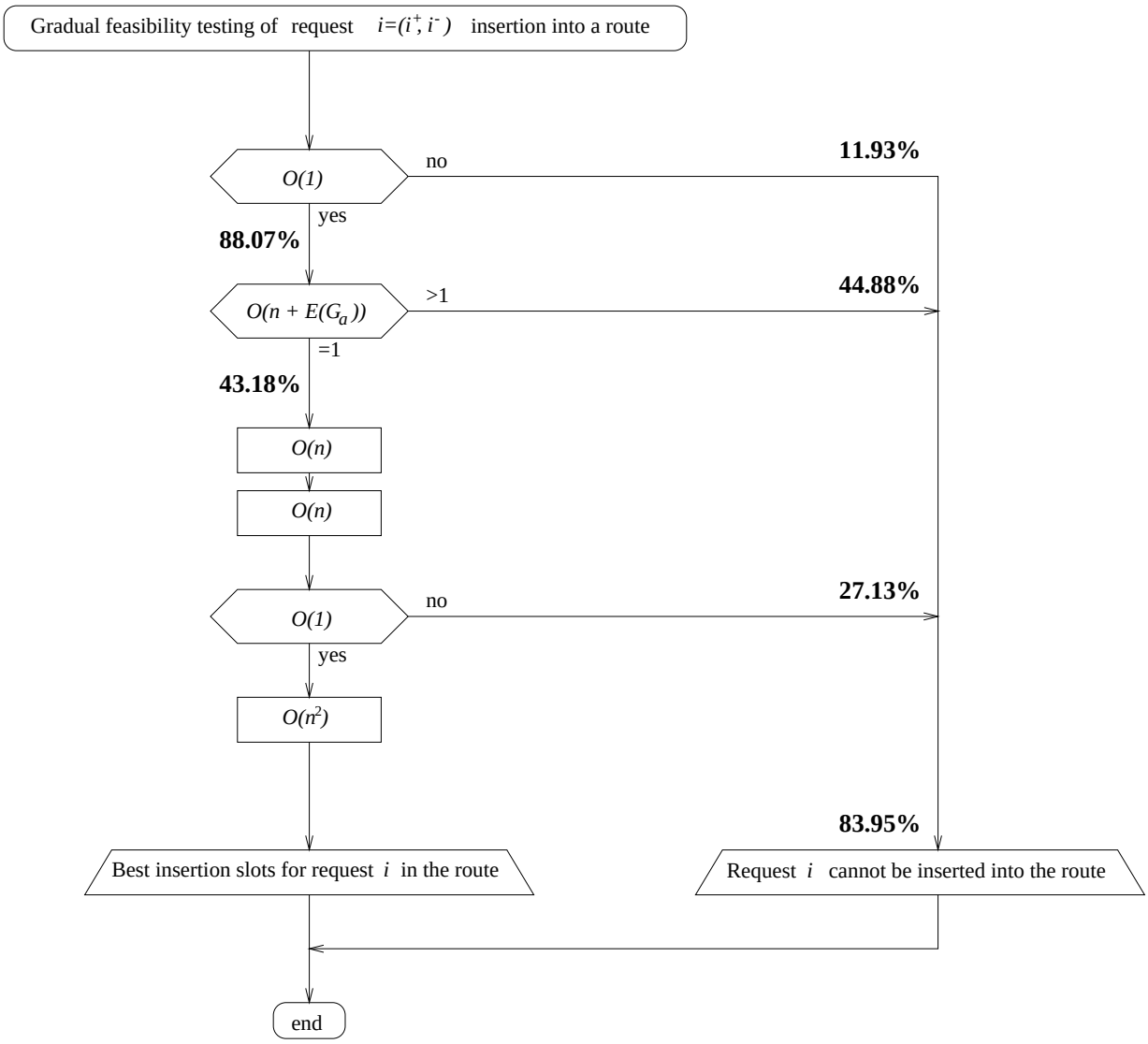


Figure 7.2: Percentages of flow through the algorithm of gradual feasibility testing of a new request being served by one route, and the complexity of each step of the algorithm.

request cannot be served by this vehicle. The number of the lower-bound test failures are almost equal for all heuristic versions. The number of failures are slightly larger for the heuristics with the immediate request assignment.

For the two tests examined so far, the differences are higher between different request criteria than between different waiting strategies.

The separate location insertion is a brute-force search (between pre-determined two route locations) of feasible insertion slots for each of the two stop locations of the request. It failed in 26.97% of the evaluations of non-empty routes, meaning that one of the stop locations of the request cannot be feasibly inserted into the route.

A gradual feasibility testing algorithm is given in Figure 7.1 with the percentages of flow through the algorithm. The percentages are based on test failure percentages from the given tables (of this subsection). Figure 7.2 shows the steps of the gradual feasibility testing algorithm by their complexity.

We believe that the gradual feasibility testing can be even more valuable when the time windows (of a problem) are narrow.

7.5 Tabu search improvement procedure

A tabu search improvement procedure is incorporated into the on-line heuristic with delayed request assignment: every 15 minutes all new requests are assigned, requests are sorted by difficulty (the most difficult requests are assigned first), all active vehicles are evaluated without sorting. The time interval (of 12 minutes) between two assignments is given to the tabu search procedure to improve the current solution. We solved the scheduling problem using two strategies: the drive-first waiting strategy, and the advanced dynamic waiting strategy. The problem instances are one-depot instances with 100 requests and with 500 requests, used in previous experiments.

The neighborhood implemented is based on ejection chains (see details in Subsubsection

On-line system with tabu search improvement procedure.						
		Results		Tabu search		
Req	**Wait strategy**	Route lengths	Veh	TS Calls	Succ	Iterat
100 (100/h)	Drive-first	2487.31	13.7	26.6	14.4	6118.5
	Adv-dyn-wait	2377.50	13.3	29.5	17.1	3412.67
500 (250/h)	Drive-first	8477.20	34.11	36.0	27.3	418.1
	Adv-dyn-wait	8263.62	34.2	36.0	27.0	297.33

Table 7.24: Results of the heuristic with the tabu search improvement procedure. Speed of simulation is ten (five) for 100-request (500-request) instances. Comparisons of two waiting strategies. One-depot instances.

On-line system without improvement procedure.			
		Results	
Req	**Wait strategy**	Route lengths	Veh
100	Drive-first	2619.54	13.78
	Adv-dyn-wait	2505.89	13.00
500	Drive-first	9266.08	40.70
	Adv-dyn-wait	8775.00	35.44

Table 7.25: Results of the heuristic without improvement procedure. One-depot instances. Delayed request assignment; requests sorted by deadline before all are assigned to vehicles. All unsorted vehicles are evaluated during a new request assignment.

Tabu search as improvement procedure. Without tabu search → With tabu search.				
		Route lengths		
Req	*Wait strategy*	Without TS	With TS	**Improvement**
100	Drive-first	2619.54	2487.31	**5.05%** shorter routes
	Adv-dyn-wait	2505.89	2377.50	**5.12%** shorter routes
	Improvement	***4.34%***	***4.41%***	
		Number of vehicles		
		Without TS	With TS	Improvement
	Drive-first	13.78	13.7	0.58% less vehicles
	Adv-dyn-wait	13.0	13.3	-2.31% less vehicles
	Improvement	*6%*	*2.92%*	
		Route lengths		
Req	*Wait strategy*	Without TS	With TS	**Improvement**
500	Drive-first	9266.08	8477.20	**8.51%** shorter routes
	Adv-dyn-wait	8775.00	8263.62	**5.83%** shorter routes
	Improvement	***5.30%***	***2.52%***	
		Number of vehicles		
		Without TS	With TS	Improvement
	Drive-first	40.7	34.11	16.19% less vehicles
	Adv-dyn-wait	35.44	34.2	3.50% less vehicles
	Improvement	*12.92%*	*-0.26%*	

Table 7.26: Improvements achieved by the usage of the tabu search improvement procedure.

On-line system with tabu search procedure applied longer.						
		Results		Tabu search		
Req	**Wait strategy**	Route lengths	Veh	TS Calls	Succ	Iterat
100 (40/h)	Drive-first	2515.32	13.77	24.9	12.3	25421.2
	Adv-dyn-wait	2365.95	12.7	30.0	17.2	14287.0
500 (100/h)	Drive-first	8445.77	36.3	36.0	28.4	733.3
	Adv-dyn-wait	8176.18	33.2	36.0	28.3	568.3

Table 7.27: Results of the heuristic with the tabu search improvement procedure. Speed of simulation is four (two) 100-request (500-request) instances. Comparisons of two waiting strategies. One-depot instances.

3.3.2.4). One neighborhood exploration starts with calculating distances between all (route, request) pairs. Distance between a route and each of its requests is set to infinity, in order to force moving requests to other routes. The closest k ($k = 10$) of all (route, request)-pairs represent seeds, each of which will be used for initializing a new ejection chain. A greedy heuristic is used to extend the current ejection chain. Forming new routes is not allowed.

The experiments were set as follows:

- The 100-request instances, with the speed of simulation set to ten: the service period is simulated by one hour of computer time. The tabu search has 1.2 minutes to improve the solution. The situation is similar to solving an instance with 100 requests per hour.
- The 100-request instances, with the speed of simulation set to four: the service period is simulated by 2.5 hours of computer time. The tabu search has three minutes to improve the solution. The situation is similar to solving an instance with 40 requests per hour.
- The 500-request instances, with the speed of simulation set to five: the service period is simulated by two hours of computer time. The tabu search has 2.4 minutes to improve the solution. The situation is similar to solving an instance with 250 requests per hour.

Tabu search improvement procedure. What if the tabu search procedure is applied 2.5 times longer?				
		Route lengths		
Req	*Wait strategy*	TS	TS longer	**Improvement**
100	Drive-first	2487.31	2515.32	**-1.13%** shorter routes
	Adv-dyn-wait	2377.50	2365.95	**0.49%** shorter routes
	Improvement	***4.41%***	***5.94%***	
		Number of vehicles		
		TS	TS longer	Improvement
	Drive-first	13.7	13.77	-0.51% less vehicles
	Adv-dyn-wait	13.3	12.7	4.51% less vehicles
	Improvement	*2.92%*	*7.77%*	
		Route lengths		
Req	*Wait strategy*	TS	TS longer	**Improvement**
500	Drive-first	8477.20	8445.77	**0.37%** shorter routes
	Adv-dyn-wait	8263.62	8176.18	**1.06%** shorter routes
	Improvement	***2.52%***	***3.19%***	
		Number of vehicles		
		TS	TS longer	Improvement
	Drive-first	34.11	36.3	-6.42% less vehicles
	Adv-dyn-wait	34.2	33.2	2.92% less vehicles
	Improvement	*-0.26%*	*8.54%*	

Table 7.28: Improvements achieved by the longer usage of the tabu search improvement procedure. The 'TS' column shows solution values when the tabu search is run 1.2 minutes (2.6 minutes) for 100-request (500-request) instances. The 'TS longer' column shows solution values when the tabu search is run 2.5 times longer, *i.e.*, three minutes (six minutes) for 100-request (500-request) instances.

- The 500-request instances, with the speed of simulation set to two: the service period is simulated by five hours of computer time. The tabu search has six minutes to improve the solution. The situation is similar to solving an instance with 100 requests per hour.

Table 7.24 describes the results of our on-line system with the tabu search improvement procedure. The experiments were done with the speed of simulation set to ten (five) for 100-request (500-request) instances. The left part of the table shows the total route lengths and the total number of vehicles, while the right part of the table gives some characteristics of the tabu search procedure. Column 'TS Calls' contains the number of calls to the tabu search procedure during the whole service period. The tabu search procedure is called only if there is more than one route with at least one request both of whose locations are unserved. Column 'Succ' shows the number of successful calls to the tabu search procedure, meaning the tabu search procedure has improved the current solution. Column 'Iterat' presents the average number of iterations of the tabu search procedure (per one call to the tabu search procedure). (One iteration is one neighborhood search.) Table 7.25 gives results of the experiments with the same heuristic but when the tabu search was not used. Table 7.26 gives comparisons between the on-line systems with and without the tabu search procedure. The results of the two waiting strategies are given: the drive-first strategy, and the advanced dynamic waiting. The last column depicts improvements achieved by the tabu search procedure, while the 'improvement' rows show the improvement obtained by the advanced dynamic waiting over the drive-first strategy. Improvements achieved by the tabu search range from 5.05% to 8.51% in the route lengths. Improvements achieved by the advanced dynamic waiting range from 2.52% to 4.41% in the route lengths, when the tabu search is used, and from 4.34% to 5.30% when the tabu search is not used.

Table 7.27 shows the results obtained when the tabu search procedure is given 2.5 times more time to improve a solution compared to the tabu search shown in Table 7.24. The speed of simulation equals four (two) for 100-request (500-request) instances. The same characteristics

of the tabu search procedure are given in the right part of the table. Table 7.28 displays the changes in the solutions if the tabu search improvement procedure is applied longer. The routes are slightly shorter, with better improvements reached when the advanced dynamic waiting is used rather than the drive-first strategy used. The number of vehicles used increases when the drive-first strategy is used in the combination with longer tabu search. The improvements due to the tabu search procedure would be even higher if the speed of simulation is one and the tabu search is applied for 12 minutes real time.

One can thus deduce that the difference between waiting strategies (for the benefit of the advanced dynamic waiting) might remain unaltered with the use of any other procedure for solving the routing problem.

Note that (check in Table 7.26 and in Table 7.28) the improvements achieved by the advanced dynamic waiting (compared to the first-drive waiting strategy) are similar whether the on-line system contains only the simple (greedy) insertion procedure, or the on-line system is extended by the tabu search improvement procedure (which is applied shorter or longer). We can deduce that the difference between waiting strategies (for the benefit of the advanced dynamic waiting) might remain unaltered with the use of any other procedure for solving the routing problem.

7.6 Value of information

The value of information defined in Subsection 6.2.2 represents a measure of how suitable one heuristic is for solving a dynamic optimization problem. To determine the value of information of some of heuristics implemented, experiments were performed on the static versions of the same instances (with 100 and 500 requests) in which all requests are known in advance.

These experiments are called off-line experiments. The performances of four on-line heuristics were examined:

- the heuristic with the tabu search improvement procedure, and with the drive-first waiting strategy,

Off-line heuristic results. All requests known in-advance. Heuristic with tabu search improvement procedure.

	Results		Tabu search	
Req	Route lengths	Veh	TS Calls	Iterat
100	2096.41	10.7	1.0	1018.3
500	7913.56	25.2	1.0	1288.1

Table 7.29: Results of the heuristic with the tabu search procedure. The problem is static with all requests known in advance. The heuristic has run for one (two) hour for 100-request (500-request) instances.

The value of information. Heuristic with the tabu search. Off-line → On-line.

Req	Route lengths			
100	Off-line	On-line		**Value of information × 100**
	2096.41	Drive-first	2487.31	**18.65%** longer routes
		Adv-dyn-wait	2377.50	**13.41%** longer routes
	Number of vehicles			
	Off-line	On-line		Value of information × 100
	10.7	Drive-first	13.7	28.4% more vehicles
		Adv-dyn-wait	13.3	24.30% more vehicles
Req	Route lengths			
500	Off-line	On-line		**Value of information × 100**
	7913.56	Drive-first	8477.20	**7.12%** longer routes
		Adv-dyn-wait	8263.62	**4.42%** longer routes
	Number of vehicles			
	Off-line	On-line		Value of information × 100
	25.2	Drive-first	34.11	35.36% more vehicles
		Adv-dyn-wait	34.2	35.71% more vehicles

Table 7.30: The value of information (multiplied by 100) for the heuristic with the tabu search improvement procedure. The two waiting strategies are used.

Off-line heuristic results. All requests known in-advance. Heuristic without improvement procedure.		
	Results	
Req	Route lengths	Veh
100	2156.25	10.7
500	7924.64	25.2

Table 7.31: Results of running off-line the heuristic without an improvement procedure. The problem is static.

- the heuristic with the tabu search improvement procedure, and with the advanced dynamic waiting strategy,
- the heuristic without an improvement procedure, and with the drive-first waiting strategy,
- the heuristic without an improvement procedure, and with the advanced dynamic waiting strategy.

To get the values of information for these heuristics off-line experiments were run with two heuristic: the heuristic with the tabu search improvement procedure and the heuristic without tabu search. (The waiting strategy does not play a role in solving the static problem instances.)

Table 7.29 gives results of the heuristic with the tabu search improvement procedure. The heuristic inserts all requests by the simple (greedy) insertion procedure, and then calls the tabu search procedure that improves the solution. The tabu search comprises around one thousand iterations.

Table 7.30 gives the value of information for the heuristic with the tabu search procedure. The value of information decreases with an increase in problem size. A similar situation arises (see Table 7.32) in the case of the heuristic without an improvement procedure, with the difference that the heuristic without tabu search procedure has higher values of information, no matter which waiting strategy is used. (The results of off-line experiments on the heuristic

The value of information. Heuristic without tabu search. Off-line → On-line.				
Req	Route lengths			
100	Off-line	On-line		**Value of information × 100**
	2156.2	Drive-first	2619.54	**21.49%** longer routes
		Adv-dyn-wait	2505.89	**16.22%** longer routes
	Number of vehicles			
	Off-line	On-line		Value of information × 100
	10.7	Drive-first	13.78	28.79% more vehicles
		Adv-dyn-wait	13.00	21.50% more vehicles
Req	Route lengths			
500	Off-line	On-line		**Value of information × 100**
	7924.64	Drive-first	9266.08	**16.93%** longer routes
		Adv-dyn-wait	8775.00	**10.73%** longer routes
	Number of vehicles			
	Off-line	On-line		Value of information × 100
	25.2	Drive-first	40.70	61.51% more vehicles
		Adv-dyn-wait	35.44	40.63% more vehicles

Table 7.32: The value of information (multiplied by 100) for the heuristic without an improvement procedure.

without the improvement procedure are given in Table 7.31. The data regarding the on-line heuristics are taken from Table 7.25.) Overall the most suitable heuristic for solving dynamic PDPTW problem is the heuristic with the tabu search improvement procedure in which the scheduling problem is solved by the advanced dynamic waiting. (The value of information given in all tables of this subsection is multiplied by 100 in order to show the percentages of route length increase.)

Chapter 8

Conclusions

In this thesis, we have investigated heuristic approaches to solving the dynamic pickup and delivery problem with time windows (PDPTW). The study makes two distinct contributions. It provides a solution framework, called two-strategy heuristic, explicitly developed for loosely time-constrained dynamic pickup and delivery problem. A second major contribution is development of waiting strategies for solving the scheduling problem for a fixed route of a dynamic PDPTW.

The two-strategy heuristic proposes solving dynamic problem over two time horizons: the short-term time horizon covers the first sections of routes, and the long-term time horizon covers the rest of routes (or the whole routes). Variations of the two-strategy heuristic are:

(1) The heuristic for solving the problem model with two different goals (the short-term goal, and the long-term goal) by using the same method for reaching both the goals. Each goal is to be strived for over one of the two time horizons: the short-term goal over the short-term time horizon, and the long-term goal over the long-term time horizon.

(2) The heuristic for solving the problem model with one goal, by using two different methods: one method, usually simple and fast, is applied over the short-term time horizon, and the other method over the long-term time horizon.

(3) The heuristics for solving the problem model with two different goals, each over one of the two time horizons, by using two different methods.

The two-strategy heuristic being a framework in which any known method can be embedded, represents an approach that can outperform any known method. It can be used for solving both the routing and scheduling of a dynamic vehicle routing problem. A version of the heuristic, the advanced dynamic waiting strategy that solves the scheduling problem, was tested on a series of dynamic PDPTW instances. Results obtained confirm efficiency of the proposed approach.

The heuristic is flexible enough that it can be used for solving many problem versions. It can accommodate problems with depot(s) or without a depot, it can be adapted to handle different goals (those of the company's, drivers', or customers'), and it can handle additional constraints: capacity constraints, different durations of drivers' working day, etc. Furthermore, this problem-solving concept can be implemented in parallel such that different strategies run on different processors. However, a fair amount of synchronization would be required.

From the methodological point of view, the two-strategy heuristic represents a hybrid method for solving dynamic problems that can combine the best features of fast algorithms, on the one hand, and complex (but slower) methods, on the other. As such the method can be used for solving many dynamic time-constrained vehicle routing problems including: dynamic VRPTW, dynamic m-TSPTW, dynamic traveling repairman problem, inter-modal services (Psaraftis, 1995), the combined pickup and delivery services (Psaraftis, 1995), and the prisoner transportation problem. Even already well-solved problems, such as the dial-a-ride problem (DARP) and the handicapped transportation problem (HTP), can benefit from the two-strategy heuristic applied in a dynamic environment: a fast greedy method can be used for assigning urgent requests, while the successful Dantzig-Wolfe decomposition/column generation method could handle all other requests. Another version of the two-strategy heuristic that relies on the concept of modeling a static goal by two distinct dynamic goals could be advantageous in solving many dynamic routing problems. Overall, the idea of integrating two strategies (two goals and/or two-methods)

is of wide applicability, especially on problems whose solutions spread over a significant part of the problem service period. Moreover, the two-strategy heuristic could be extended to a multi-strategy heuristic (over multiple time horizons) for solving complex practical dynamic problems.

The scheduling aspect of a routing problem proved itself to be more important in a dynamic environment than it is in a static environment. We have introduced several waiting strategies for solving the scheduling problem: two simple strategies (the drive-first strategy and the wait-first strategy), and two strategies that tend to distribute waiting times uniformly along the route (the dynamic waiting strategy and the advanced dynamic waiting strategy). Computational results clearly demonstrate the importance of scheduling, and show that applying more complex waiting strategies can be quite powerful. On two sets of instances, the advanced dynamic waiting strategy improves results by approximately 2-8%. (The routing problem was solved with either a greedy insertion algorithm or a tabu search procedure.) The advanced dynamic waiting strategy could perform even better if further refining was done on modelling of routes by service zones (shape of zone base, size of zone base, after-zone wait time).

We expect that the proposed waiting strategies can be applied to capacity-constrained problems as well. Still, the loosely time-constrained problems will gain more, because of the larger sets of feasible solutions. When applying this concept to solving problems which deal with transporting people, one has to be careful because vehicles are not allowed to wait when carrying passengers. Nevertheless, the strategies might be used in coordination with 'pieces of work', such that one service zone ends at the end of one 'piece of work'.

This thesis also describes other developments of our research among which are:

- The two-goal dynamic model for a dynamic routing problem consisting of the short-term goal (of minimizing route lengths) and the long-term goal (of maintaining the routes during the service period such that future requests can be effectively inserted so as to minimize the total route lengths).

- Dynamic utility function for the evaluation of one location insertion into a route.
- The value of information - a measure of the effectiveness of a method for solving dynamic problems. We have explored an approximation of the value of information which compares the solutions obtained by the same solution method in a static and in a dynamic environment. Other possible directions would be to research a lower and an upper bound of the value of information.
- The precedence graphs and their role in testing feasibility of a request insertion into a route in PDPTW. (The precedence graphs can also be used for finding bounds on the number of vehicles for m-TSPTW.)

Overall, research on dynamic time-constrained routing problems can be very challenging but also quite rewarding. We feel that there remain many unexplored avenues in pure dynamic environments and strongly recommend approaches that include mixing theoretical work with empirical studies. We also believe that the two-strategy class of heuristics points to one good direction for further research in solving dynamic time-constrained routing and scheduling problems.

The research can be extended by studying other applications of on-line algorithms. Prospective areas include: industrial manufacturing, practical computer systems, and robotics.

Production planning in industrial manufacturing is concerned with the determination of production, inventory, and work force levels to meet fluctuating demand requirements (Hax, Candea, 1984). A widely adopted strategy is planning with a rolling time horizon, within which a problem model is repeatedly solved as new information becomes available. In such an environment, the two-strategy heuristic can be used for extending the time horizon, if needed. Furthermore, in many cases managerial decisions require the consideration of several goals. If the goals can be classified as short-term goals and long-term goals, the two-strategy heuristic can be used to emphasize and utilize the situation. Also, the two-strategy heuristic can be useful

when the planning of the distant future depends on decisions made in the planning of the near future.

Operations scheduling in industrial manufacturing deals with the most detailed scheduling decisions, involving the assignment of the operations to specific machines and operators during a given time interval (Hax, Candea, 1984). In a large class of production systems, the operations scheduling corresponds to the job shop scheduling problem. The job shop is the set of all machines that are associated with a given set of jobs, and the problem is to order the operations to be performed on each machine subject to routing and shop constraints, such that some measurable function of the ordering is optimized. In a closed job shop, routine demand for finished products can be forecasted, while in an open job shop all products are made to order. In an open shop, dynamic aspect of the problem is usually present in the form of dynamic arrival of jobs. As customer orders come in, the scheduling function tries to accommodate them within the available capacity of the various work centers required for processing. Objectives are reduction of shop congestion and satisfaction of due dates. The emphasis is on the readiness of the work force and equipment to absorb fluctuating demands. In such a dynamic environment, the two-strategy heuristic can be useful.

Scheduling of computer jobs is a practical problem within a network of computer systems. The schedule determines for each job a set of machines on which the job will run and a set of time slots which the job will use. Each machine is assigned to a single job at any time, and the processing time of a job always takes at least its running time. The problem is dynamic. Additional complexity comes from jobs having different priorities. The two-strategy heuristic might be beneficial for constructing a schedule that covers a longer future period.

In robotics, dynamic moving of robots on the production floor can produce problems dealing with jobs that are mutually dependent, *i.e.*, serving one job requires serving another job within a prespecified time. The robots can also be non-homogeneous. A similar problem can be found in inventories in which a fleet of non-homogeneous robots serve requests comprising of a set of objects.

Further developments in telecommunications and in the computer industry will surely open new areas in which on-line algorithms will be needed and beneficial.

Bibliography

Aarts, E., J.K. Lenstra (eds.) (1997) *Local Search in Combinatorial Optimization*, Wiley, Chichester, UK.

Appelgren, L.H. (1969) "A column generation algorithm for a ship scheduling problem", *Transportation Science* 3, 53-68.

Assad, A.A. (1988) "Modeling and implementation issues in vehicle routing", in B.L. Golden, A.A. Assad (eds.) *Vehicle Routing: Methods and Studies*, Studies in Management Science and Systems, Volume 16, North-Holland, Amsterdam, 7-45.

Atkinson, JB (1998) "A greedy randomized search heuristic for time-constrained vehicle scheduling and the incorporation of a learning strategy", *Journal of the Operational Research Society* 49, 700-708.

Badeau, P., F. Guertin, M. Gendreau, J.-Y. Potvin, É.D. Taillard (1997) "A parallel tabu search heuristic for the vehicle routing problem with time windows", *Transportation Research C* 5, 109-122.

Baker, E.K. (1983) "An exact algorithm for the time constrained traveling salesman problem", *Operations Research* 31, 938-945.

Baker, E.K., J.R. Schaffer (1986) "Computational experience with branch exchange heuristics for vehicle routing problems with time window constraints", *American Journal of Mathematical and Management Sciences* 6, 261-300.

Ball, M.O., M. Magazine (1981) "The design and analysis of heuristics", *Networks* 11, 215-219.

Ball, M.O., T.L. Magnanti, C.L. Monma, G.L. Nemhauser (eds.) (1995) *Network Routing*, Handbooks in Operations Research and Management Science, Volume 8, North-Holland, Amsterdam.

Benyahia, I., J.-Y. Potvin (1998) "Decision support for vehicle dispatching using genetic programming", *IEEE Transactions on Systems, Man and Cybernetics - Part A: Systems and Humans* 28, 306-314.

Bodin, L., B.L. Golden (1981) "Classification in vehicle routing and scheduling", *Networks* 11, 97-108.

Bodin, L., B.L. Golden, A.A. Assad, M.O. Ball (1983) "Routing and scheduling of vehicles and crews: the state of the art", *Computers and Operations Research* 11, 63-211.

Bondy, J.A., U.S.R. Murty (1976) *Graph Theory with Applications*, American Elsevier, New York, and Macmillan, London.

Borodin, A., R. El-Yaniv (1998) *Online Computation and Competitive Analysis*, Cambridge University Press, Cambridge.

Borodin, A., S. Irani, P. Raghavan, B. Schieber (1992) "Competitive paging with locality of reference", in McGeoch, L.A., D.D. Sleator (eds.) *On-Line Algorithms: Proceeding of a DIMACS Workshop, February 11-13, 1991*, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Volume 7, American Mathematical Society, 167-168.

Brenner, W., R. Zarnekow, H. Wittig (1998) *Intelligent Software Agents*, Springer Verlag, New York.

Chakravarthy, A., D. Ghose (1998) "Obstacle avoidance in a dynamic environment: a collision cone approach", *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans* 28, 562-574.

Chartrand, G., L. Lesniak (1986) *Graphs and Digraphs*, 2nd ed., Wadsworth & Brooks/Cole, Mathematics Series, Monterey, California.

Chiang, W.-C., R.A. Russell (1996) "Simulated annealing metaheuristics for the vehicle routing problem with time windows", *Annals of Operations Research* 63, 3-27.

Christofides, N. (1985) "Vehicle routing", in E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, D.B. Shmoys (eds.) *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*, Wiley, Chichester, West Sussex, 431-448.

Christofides, N. (1985) "Vehicle scheduling and routing", Presented at the *12th International Symposium on Mathematical Programming*, Cambridge, MA.

Christofides, N., A. Mingozzi, P. Toth (1981) "State-space relaxation procedures for the computation of bounds to routing problems", *Networks* 11, 145-164.

Chvátal, V. (1983) *Linear Programming*, Freeman, New York.

Cordeau, J.-F., M. Gendreau, G. Laporte (1997) "A tabu search heuristic for periodic and multi-depot vehicle routing problems", *Networks* 30, 105-119.

Cordeau, J.-F., G. Laporte, A. Mercier (2000) "A unified tabu search heuristic for vehicle routing problems with time windows", to appear in *Journal of the Operational Research Society*.

Cormen, T.H., C.E. Leiserson, R.L. Rivest (1993) *Introduction to algorithms*, McGraw-Hill, New York.

Crainic, T.G., G. Laporte (eds.) (1998) *Fleet Management and Logistics*, Kluwer Academic Publishers, Boston.

Cullen, F.H., J.J. Jarvis, H.D. Ratliff (1981) "Set partitioning based heuristics for interactive routing", *Networks* 11, 125-143.

Desaulniers, G., J. Desrosiers, I. Ioachim, M.M. Solomon, F. Soumis, D. Villeneuve (1998) "A unified framework for deterministic time constrained vehicle routing and crew scheduling problems", in T.G. Crainic, G. Laporte (eds.) (1998) *Fleet Management and Logistics*, Kluwer Academic Publishers, Boston, 57-93.

Desrochers, M., J. Desrosiers, M.M. Solomon (1992) "A new optimization algorithm for the vehicle routing problem with time windows", *Operations Research* 40, 342-354.

Desrochers, M., J.K. Lenstra, M.W.P. Savelsbergh, F. Soumis (1988) "Vehicle routing with time windows: optimization and approximation" in B.L. Golden, A.A. Assad (eds.) *Vehicle Routing: Methods and Studies*, Studies in Management Science and Systems, Volume 16, North-Holland, Amsterdam, 65-84.

Desrosiers, J., Y. Dumas, M.M. Solomon, F. Soumis (1995) "Time constrained routing and scheduling", in M.O. Ball, T.L. Magnanti, C.L. Monma, G.L. Nemhauser, (eds.) *Network Routing*, Handbooks in Operations Research and Management Science, Volume 8, North-Holland, Amsterdam, 35-139.

Desrosiers, J., Y. Dumas, F. Soumis (1986) "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, 301-325.

Desrosiers, J., Y. Dumas, F. Soumis (1988) "The multiple vehicle dial-a-ride problem", in J.R. Daduna, A. Wren (eds.) *Computer Aided Transit Scheduling*, Proceedings of the Fourth International Workshop on Computer-Aided Scheduling of Public Transport, Lecture Notes in Economics and Mathematical Systems 308, Springer Verlag, 15-27.

Desrosiers, J., Y. Dumas, F. Soumis, S. Taillefer, D. Villeneuve (1991) "An algorithm for mini-clustering in handicapped transport", Les cahiers du GERAD G-91-02, École des Hautes Études Commerciales, Université de Montréal, Montréal.

Desrosiers, J., F. Soumis, M. Desrochers (1984) "Routing with time windows by column generation", *Networks* 14, 545-565.

Dror, M., G. Laporte, P. Trudeau (1989) "Vehicle routing with stochastic demands: Properties and solution frameworks", *Transportation Science* 23, 166-176.

Dror, M., W. Powell (1993) "Stochastic and dynamic models in transportation", *Operations Research* 41, 11-14.

Duhamel, C., J.-Y. Potvin, J.-M. Rousseau (1997) "A tabu search heuristic for the vehicle routing problem with backhauls and time windows", *Transportation Science* 31, 49-59.

Dumas, Y., J. Desrosiers, E. Gélinas, M.M. Solomon (1995) "An optimal algorithm for the traveling salesman problem with time windows", *Operations Research* 43, 367-371.

Dumas, Y., J. Desrosiers, F. Soumis (1989) "Large-scale multi-vehicle dial-a-ride problems", Les cahiers du GERAD G-89-30, École des Hautes Études Commerciales, Université de Montréal, Montréal.

Dumas, Y., J. Desrosiers, F. Soumis (1991) "The pickup and delivery problem with time windows", *European Journal of Operational Research* 54, 7-22.

Fiat, A., G.J. Woeginger (1998) *Online Algorithms: The State of the Art*, Springer, Berlin.

Fisher, M. (1995) "Vehicle routing", in M.O. Ball, T.L. Magnanti, C.L. Monma, G.L. Nemhauser (eds.) *Network Routing*, Handbooks in Operations Research and Management Science, Volume 8, North-Holland, Amsterdam, 1-33.

Ford, L.R., D.R. Fulkerson (1962) *Flows in Networks*, Princeton University Press, Princeton, NJ.

Garcia, B.-L., J.-Y. Potvin, J.-M. Rousseau (1994) "A parallel implementation of the tabu search heuristic for vehicle routing problems with time windows constraints", *Computers and Operations Research* 21, 1025-1033.

Garey, M.R., D.S. Johnson (1979) *Computers and Intractability: A Guide to the Theory of NP-Completeness*, Freeman, New York.

Gendreau, M., F. Guertin, J.-Y. Potvin, R. Séguin (1998) "Neighborhood search heuristics for a dynamic vehicle dispatching problem with pick-ups and deliveries", Report CRT-98-10, Centre de Recherche sur les Transports (CRT), Université de Montréal, Montréal.

Gendreau, M., F. Guertin, J.-Y. Potvin, É.D. Taillard (1999) "Parallel tabu search for real-time vehicle routing and dispatching", *Transportation Science* 33, 381-390.

Gendreau, M., A. Hertz, G. Laporte (1992) "New insertion and postoptimization procedures for the traveling salesman problem", *Operations Research* 40, 1086-1094.

Gendreau, M., A. Hertz, G. Laporte (1994) "A tabu search heuristic for the vehicle routing problem", *Management Science* 40, 1276-1290.

Gendreau, M., A. Hertz, G. Laporte, M. Stan (1998) "A generalized insertion heuristic for the traveling salesman problem with time windows", *Operations Research* 43, 330-335.

Gendreau, M., G. Laporte, J.-Y. Potvin (1997) "Vehicle routing: modern heuristics" in E. Aarts, J.K. Lenstra (eds.) *Local Search in Combinatorial Optimization*, Wiley, Chichester, UK, 311-336.

Gendreau, M., G. Laporte, F. Semet (2000) "A dynamic model and parallel tabu search heuristic for real-time ambulance relocation", Report CRT-2000-17, Centre de Recherche sur les Transports (CRT), Université de Montréal, Montréal.

Gendreau, M., G. Laporte, D. Vigo (1999) "Heuristics for the traveling salesman problem with pickup and delivery", *Computers and Operations Research* 26, 699-714.

Gendreau, M., J.-Y. Potvin (1998) "Dynamic vehicle routing and dispatching", in T.G. Crainic, G. Laporte (eds.) (1998) *Fleet Management and Logistics*, Kluwer Academic Publishers, Boston, 115-126.

Glover, F. (1989) "Tabu search - part I", *ORSA Journal on Computing* 1, 190-206.

Glover, F. (1990) "Tabu search - part II", *ORSA Journal on Computing* 2, 4-32.

Glover, F. (1996) "Ejection chains, reference structures and alternating path methods for traveling salesman problems", *Discrete Applied Mathematics* 65, 223-253.

Golden, B.L., A.A. Assad (eds.) (1988) *Vehicle Routing: Methods and Studies*, Studies in Management Science and Systems, Volume 16, North-Holland, Amsterdam.

Hax, C.A., D. Candea (1984) *Production and Inventory Management*, Prentice-Hall, Englewood Cliffs, NJ.

Homberger, J., H. Gehring (1999) "Two evolutionary metaheuristics for the vehicle routing problem with time windows", *INFOR* 37, 297-318.

Ichoua, S., M. Gendreau, J.-Y. Potvin (2000) "Diversion issues in real-time vehicle dispatching", *Transportation Science* 34, 426-438.

Ioachim, I., J. Desrosiers, Y. Dumas, M.M. Solomon, D. Villeneuve (1995) "A request clustering algorithm for door-to-door handicapped transportation", *Transportation Science* 29, 63-78.

Jaw, J.-J., A.R. Odoni, H.N. Psaraftis, N.H.M. Wilson (1986) "A heuristic algorithm for the multi-vehicle advance request dial-a-ride problem with time windows", *Transportation Research B* 20, 243-257.

Karp, R.M. (1992) "On-line algorithms versus off-line algorithms: how much it is worth to know the future?", in J. van Leeuwen (ed.) *Information Processing 92: Proceedings of the IFIP 12th World Computer Congress*, Madrid, Spain, 7-11 September 1992, North-Holland, 416-429.

Knox, E.L. (1995) "The Persian wars" in *History of Western Civilization*, Boise State University, 17 June 1995, <http://history.boisestate.edu/westciv/persian/10.htm>, 15 August 2001.

Kolen, A.W.J., A.H.G. Rinnooy Kan, H.W.J.M. Trienekens (1987) "Vehicle routing with time windows", *Operations Research* 35, 266-273.

Kontoravdis, G., J.F. Bard (1995) "A GRASP for the vehicle routing problem with time windows", *ORSA Journal on Computing* 7, 10-23.

Langevin, A., M. Desrochers, J. Desrosiers, S. Gélinas, F. Soumis (1993) "A two-commodity flow formulation for the traveling salesman and makespan problems with time windows", *Networks* 23, 631-640.

Laporte, G. (1992) "The vehicle routing problem: an overview of exact and approximate algorithms", *European Journal of Operational Research* 59, 345-358.

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

Laporte, G., H. Mercure, Y. Nobert (1986) "An exact algorithm for the asymmetrical capacitated vehicle routing problem", *Networks* 16, 33-46.

Laporte, G., Y. Nobert (1987) "Exact algorithms for the vehicle routing problem", *Annals of Discrete Mathematics* 31, 147-184.

Lawler, E.L. (1976) *Combinatorial Optimization: Networks and Matroids*, Holt, Rienhart and Winston, New York.

Lawler, E.L., J.K. Lenstra, A.H.G. Rinnooy Kan, D.B. Shmoys (eds.) (1985) *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*, Wiley, Chichester, West Sussex.

Lee, D.T., F.P. Preparata (1984) "Euclidean shortest paths in the presence of rectilinear barriers", *Networks* 14, 393-410.

Lenstra, J.K., A.H.G. Rinnooy Kan (1981) "Complexity of vehicle routing and scheduling problems", *Networks* 11, 221-227.

Lin, S., B.W. Kernighan (1973) "An effective heuristic algorithm for the traveling-salesman problem", *Operations Research* 21, 498-516.

Madsen, O.B.G., H.F. Ravn, J.M. Rygaard (1995) "A heuristic algorithm for a dial-a-ride problem with time windows, multiple capacities, and multiple objectives", *Annals of Operations Research* 60, 193-208.

Magnanti, T.L. (1981) "Combinatorial optimization and vehicle fleet planning: perspectives and prospects", *Networks* 11, 179-213.

McGeoch, L.A., D.D. Sleator (eds.) (1992) *On-Line Algorithms: Proceeding of a DIMACS Workshop, February 11-13, 1991*, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Volume 7, American Mathematical Society.

Mitchell, J.S.B., D.M. Mount, C.H. Papadimitriou (1987) "The discrete geodesic problem", *SIAM Journal on Computing* 16, 647-668.

Mitchell, J.S.B., C.H. Papadimitriou (1987) "The weighted region problem", *Proceedings of the Third Annual Symposium on Computational Geometry*, Waterloo, Ontario, Canada, June 8-10, 1987, Sponsored by ACM, ACM Press, 30-38.

Mitrović-Minić, S. (1998) "Pickup and delivery problem with time windows: A Survey", Report SFU-CMPT-TR-1998-12, School of Computing Science, Simon Fraser University, Burnaby, Canada.

Nanry, W.P., J.W. Barnes (2000) "Solving the pickup and delivery problem with time windows using reactive tabu search", *Transportation Research B* 34, 107-121.

Or, I. (1976) "Travelling salesman-type combinatorial problems and their relation to the logistics of blood banking", Ph.D. Dissertation, Department of Industrial Engineering and Management Sciences, Nortwestern University, Evanston, Ill.

Orloff, C.S. (1976) "Route constrained fleet scheduling" *Transportation Science* 10, 149-168.

Osman, I.H. (1993) "Metastrategy simulated annealing and tabu search algorithms for the vehicle routing problem" *Annals of Operations Research* 41, 421-451.

Osman, I.H., G. Laporte (1996) "Metaheuristics: a bibliography", in G. Laporte, I.H. Osman (eds.) *Metaheuristics in Combinatorial Optimization*, Annals of Operations Research 63, Baltzer, Basel, Switzerland, 513-623.

Papadimitrou, C.H., K. Stieglitz (1982) *Combinatorial Optimization: Algorithms and Complexity*, Prentice-Hall, Englewood Cliffs, NJ.

Pesant, G., M. Gendreau, J.-Y. Potvin, J.-M. Rousseau (1998) "An exact constraint logic programming algorithm for the traveling salesman problem with time windows", *Transportation Science* 32, 12-29.

Potvin, J.-Y., S. Bengio (1996) "The vehicle routing problem with time windows; part II: genetic search", *INFORMS Journal on Computing* 8, 165-172.

Potvin, J.-Y., T. Kervahut, B.-L. Garcia, J.-M. Rousseau (1996) "The vehicle routing problem with time windows; part I: tabu search", *INFORMS Journal on Computing* 8, 158-164.

Potvin, J.-Y., J.-M. Rousseau (1992) "Constraint-directed search for the advanced request dial-a-ride problem with service quality constraints", in O. Balci, R. Shrada, S.A. Zenios (eds.) *Computer Science and Operations Research: New Developments in Their Interfaces*, Pergamon Press, Oxford, England, 457-474.

Potvin, J.-Y., J.-M. Rousseau (1993) "A parallel route building algorithm for the vehicle routing and scheduling problem with time windows", *European Journal of Operational Research* 66, 331-340.

Potvin, J.-Y., J.-M. Rousseau (1995) "An exchange heuristic for routeing problems with time windows", *Journal of the Operational Research Society* 46, 1433-1446.

Powell, W.B. (1991) "Optimization models and algorithms: an emerging technology for the motor carrier industry", *IEEE Transactions on Vehicular Technology* 40, 68-80.

Powell, W.B. (1998) "On languages for dynamic resource scheduling problems", in T.G. Crainic, G. Laporte (eds.) (1998) *Fleet Management and Logistics*, Kluwer Academic Publishers, Boston, 127-157.

Powell, W.B. , P. Jaillet, A. Odoni (1995) "Stochastic and dynamic networks and routing", in M.O. Ball, Magnanti T.L., Monma C.L., Nemhauser G.L., (eds.) *Network Routing*, Handbooks in Operations Research and Management Science, Volume 8, North-Holland, Amsterdam, 141-295.

Psaraftis, H.N. (1980) "A dynamic programming solution to the single vehicle many-to-many immediate request dial-a-ride problem", *Transportation Science* 14, 130-154.

Psaraftis, H.N. (1983a) "An exact algorithm for the single vehicle many-to-many dial-a-ride problem with time windows", *Transportation Science* 17, 351-357.

Psaraftis, H.N. (1983b) "Analysis of an $O(n^2)$ heuristic for the single vehicle many-to-many Euclidean dial-a-ride problem", *Transportation Research B* 17, 133-145.

Psaraftis, H.N. (1986) "Scheduling large-scale advance-request dial-a-ride systems", *American Journal of Mathematical and Management Sciences* 6, 327-367.

Psaraftis, H.N. (1988) "Dynamic vehicle routing problems", in B.L. Golden, A.A. Assad (eds.) *Vehicle Routing: Methods and Studies*, Studies in Management Science and Systems, Volume 16, North-Holland, Amsterdam, 223-248.

Psaraftis, H.N. (1995) "Dynamic vehicle routing: status and prospects", *Annals of Operations Research* 61, 143-164.

Raghavan, P. (1992) "A statistical adversary for on-line algorithms", in McGeoch, L.A., D.D. Sleator (eds.) *On-Line Algorithms: Proceeding of a DIMACS Workshop, February 11-13, 1991*, DIMACS Series in Discrete Mathematics and Theoretical Computer Science, Volume 7, American Mathematical Society, 79-83.

Robinson, D.F., L.R. Foulds (1980) *Digraphs: Theory and Techniques*, Gordon and Breach Science Publishers, New York.

Rochat, Y., É.D. Taillard (1995) "Probabilistic diversification and intensification in local search for vehicle routing", *Journal of Heuristics* 1, 147-167.

Savelsbergh, M.W.P. (1985) "Local search in routing problems with time windows", *Annals of Operations Research* 4, 285-305.

Savelsbergh, M.W.P. (1990) "An efficient implementation of local search algorithms for constrained routing problems", *European Journal of Operational Research* 47, 75-85.

Savelsbergh, M.W.P. (1992) "The vehicle routing problem with time windows: minimizing route duration", *ORSA Journal on Computing* 4, 146-154.

Savelsbergh, M.W.P., M. Sol (1995) "The general pickup and delivery problem", *Transportation Science* 29, 17-29.

Savelsbergh, M.W.P., M. Sol (1998) "DRIVE: Dynamic routing of independent vehicles", *Operations Research* 46, 474-490.

Schwartz, J.T., M. Sharir (1988) "A survey of motion planning and related geometric algorithms", *Artificial Intelligence* 37, 157-169.

Schrage, L. (1981) "Formulation and structure of more complex/realistic routing and scheduling problems", *Networks* 11, 229-232.

Sexton, T.R., L.D. Bodin (1985a) "Optimizing single vehicle many-to-many operations with desired delivery times: I. scheduling", *Transportation Science* 19, 378-410.

Sexton, T.R., L.D. Bodin (1985b) "Optimizing single vehicle many-to-many operations with desired delivery times: II. routing", *Transportation Science* 19, 411-435.

Shen, Y., J.-Y. Potvin, J.-M. Rousseau, S. Roy (1995) "A computer assistant for vehicle dispatching with learning capabilities", *Annals of Operations Research* 61, 189-211.

Solomon, M.M. (1986) "On the worst-case performance of some heuristics for the vehicle routing and scheduling problem with time window constraints", *Networks* 16, 161-174.

Solomon, M.M. (1987) "Algorithms for the vehicle routing and scheduling problem with time window constraints", *Operations Research* 35, 254-265.

Solomon, M.M., E.K. Baker, J.R. Schaffer (1988) "Vehicle routing and scheduling problems with time window constraints: efficient implementations of solution improvement procedures" in B.L. Golden, A.A. Assad (eds.) *Vehicle Routing: Methods and Studies*, Studies in Management Science and Systems, Volume 16, North-Holland, Amsterdam, 85-105.

Solomon, M.M., J. Desrosiers (1988) "Time window constrained routing and scheduling problems", *Transportation Science* 22, 1-13.

Soumis, F., J. Desrosiers, M. Desrochers (1985) "Optimal urban bus routing with scheduling flexibilities", *Lecture Notes in Control and Information Sciences* 59, Springer, Berlin, 155-165.

Taillard, É.D. (1993) "Parallel iterative search methods for vehicle routing problems", *Networks* 23, 661-673.

Taillard, É.D., P. Badeau, M. Gendreau, F. Guertin, J.-Y. Potvin (1997) "A tabu search heuristic for the vehicle routing problem with soft time windows", *Transportation Science* 31, 170-186.

Thangiah, S.R. (1993) "Vehicle routing with time windows using genetic algorithms", Report SRU-CpSc-TR-93-23, Slippery Rock University, Slippery Rock, PA.

Thangiah, S.R., A.V. Gubbi (1992) "Effect of genetic sectoring on vehicle routing problems with time windows", Report SRU-CpSc-TR-92-13, Slippery Rock University, Slippery Rock, PA.

Thangiah, S.R., K.E. Nygard, P.L. Juell (1992) "GIDEON: a genetic algorithm system for vehicle routing with time windows" Report SRU-CpSc-TR-92-10, Slippery Rock University, Slippery Rock, PA.

Thangiah, S.R., I.H. Osman, T. Sun (1994) "Hybrid genetic algorithm, simulated annealing and tabu search methods for vehicle routing problems with time windows" Report SRU-CpSc-TR-94-27, Slippery Rock University, Slippery Rock, PA.

Thangiah, S.R., T. Sun, J.-Y. Potvin (1994) "Heuristic approaches to vehicle routing with backhauls and time windows", Report SRU-CpSc-TR-94-28, Computer Science Department, Slippery Rock University, Slippery Rock, PA.

Thompson, P.M., H.N. Psaraftis (1993) "Cyclic transfer algorithms for multivehicle routing and scheduling problems", *Operations Research*, 41, 935-946.

Toth, P., D. Vigo (1995) "Fast local search algorithms for the handicapped persons transportation problem", Report DEIS-OR-95-1(R).

Toth, P., D. Vigo (1997) "Heuristic algorithms for the handicapped persons transportation problem", *Transportation Science* 31, 60-71.

Trotter W.T. (1992) *Combinatorics and Partially Ordered Sets*, The Johns Hopkins University Press, Baltimore.

Van Der Bruggen, L.J.J., J.K. Lenstra , P.C. Schuur (1993) "Variable-depth search for the single-vehicle pickup and delivery problem with time windows", *Transportation Science* 27, 298-311.

Wilson, N.H.M., J.M. Sussman, H.-K. Wong, T. Higonnet (1970) "Scheduling algorithms for a dial-a-ride system", Report USL TR-70-13, Massachusetts Institute of Technology (MIT).

Wolfler Calvo, R. (2000) "A new heuristic for the traveling salesman problem with time windows", *Transportation Science* 34, 113-124.

Index