TU/e EINDHOVEN UNIVERSITY OF TECHNOLOGY

Time and multiple objectives in scheduling and routing problems

Citation for published version (APA):
Dabia, S. (2012). *Time and multiple objectives in scheduling and routing problems*. [Phd Thesis 1 (Research TU/e / Graduation TU/e), Industrial Engineering and Innovation Sciences]. Technische Universiteit Eindhoven. https://doi.org/10.6100/IR724568

DOI:
10.6100/IR724568

Document status and date:
Published: 01/01/2012

Document Version:
Publisher's PDF, also known as Version of Record (includes final page, issue and volume numbers)

Please check the document version of this publication:

• A submitted manuscript is the version of the article upon submission and before peer-review. There can be important differences between the submitted version and the official published version of record. People interested in the research are advised to contact the author for the final version of the publication, or visit the DOI to the publisher's website.
• The final author version and the galley proof are versions of the publication after peer review.
• The final published version features the final layout of the paper including the volume, issue and page numbers.

Link to publication

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal.

If the publication is distributed under the terms of Article 25fa of the Dutch Copyright Act, indicated by the "Taverne" license above, please follow below link for the End User Agreement:

www.tue.nl/taverne

Take down policy
If you believe that this document breaches copyright please contact us at:

openaccess@tue.nl

providing details and we will investigate your claim.

Download date: 17. Nov. 2023

Time and Multiple Objectives in Scheduling and Routing Problems

This thesis is number D149 of the thesis series of the Beta Research School for Operations Management and Logistics. The Beta Research School is a joint effort of the School of Industrial Engineering and the department of Mathematics and Computer Science at Eindhoven University of Technology, and the Center for Production, Logistics and Operations Management at the University of Twente.

A catalogue record is available from the Eindhoven University of Technology Library.

ISBN: 978-90-8891-358-7

Printed by proefschriftmaken.nl

This research has been funded by TRANSUMO, project number 10004927.

Time and Multiple Objectives in Scheduling and Routing Problems

PROEFSCHRIFT

ter verkrijging van de graad van doctor aan de
Technische Universiteit Eindhoven, op gezag van de
rector magnificus, prof.dr.ir. C.J. van Duijn, voor een
commissie aangewezen door het College voor
Promoties in het openbaar te verdedigen
op maandag 9 januari 2012 om 16.00 uur

door

Said Dabia

geboren te Oujda, Marokko

Dit proefschrift is goedgekeurd door de promotoren:

prof.dr. A.G. de Kok
en
prof.dr. T. van Woensel

To...
Souhaila,
Aya,
Naïl,
my parents,
my family,
and my friends

Acknowledgements

The completion of this thesis learned me two main things. First, how to answer "yes" to the question my father asks me every time I call him, "Son, when are you done with your studies?". Secondly, doing a PhD is a four years process of writing and debugging programs. Anyhow, I am now typing down the last sentences of this thesis, and to be honest, I am a bit confused on how to feel exactly. Am I happy? I think I am, as I am earning my doctoral degree after an intense period of hardworking and many frustrating moments behind my computer (debugging programs!). On the other hand, I think I am feeling a bit sad as well, because this is the end of a very pleasant period of great freedom, traveling around to present my work, and meeting and collaborating with many outstanding minds. One thing I am sure of!, the completion of this dissertation would not have been possible without the support of many people. I would like to use this opportunity to thank these people who supported me and helped me during the course of my PhD project.

First and foremost, I would like to thank my promotor Ton de Kok. His contribution was certainly critical for the success of this project. The first time I met Ton was during his course SCOP (Supply Chain Operations Planning). I still remember his very first question he asked during the first lecture of SCOP: "who likes mathematics?". As an answer to his question, all students (including me) raised their hands. Consequently, Ton was very enthusiastic and exited as he started writing a lot of equations on the board. Next lecture, almost half students did not come back for the course. Actually, they never came back. I came back for all lectures that followed, and which I liked very much. Ton's enthusiasm scared half students, but was very motivating and inspiring to me. It fueled my admiration for the field of supply chain management, logistics and operations research. Thank you Ton for your enthusiasm, for your trust and for continuously encouraging me.

I would like to express my gratitude to my daily supervisor and second promotor Tom van Woensel for his excellent coaching and professional guidance. This Dissertation would not exist without his support and valuable feedbacks. Tom gave me all freedom to take charge of my own project, while providing the necessary guidance and supervision. His many ideas helped me move on again when I was stuck and confused. Tom, thank you for being a great supervisor and a supporting friend, for

believing in our work and your trust in my abilities. I am indebted to the energy, time and care you invested in our joint research. I will never forget your famous expression "..., the rest is less important". The empty space is often filled in with something like "enjoy what you are doing"; sometimes, it is filled in with "write a paper". In both cases, I really enjoyed working under your supervision.

My research visit to the Technical University of Denmark (DTU) during the period April-July 2010 was a wonderful experience. I had the great pleasure to work with Stefan Ropke. The collaboration with Stefan resulted in Chapter 5 of this dissertation. I know Stefan does not like the use of many colorful words. My words about him here are facts. Working with Stefan means literally: everyday learning something new, discussions you wish they never end, and a lot of fun. He was a good teacher for me as I learned a lot from him about exact methods for routing problems. Probably more importantly, Stefan learned me how to code in C++ and how to debug my codes. As my PhD project was all about debugging codes and Stefan is an excellent "debugger!", he was exactly the right man at the right moment. Stefan, thank you for hosting me, for the many enjoyable discussions, for the energy and the time you reserved for our joint work, and for willing to be in my inner doctoral committee. I almost forgot! I want to thank you and all staff members of DTU Transport for the exiting soccer games, I still remember you telling me to "keep fighting" during the game.

I would like to thank Gerhard Woeginger and Maria Grazia Speranza for their willingness to take part of my inner doctoral committee, and heir valuable comments and feedback on the previous version of this dissertation. Thanks to their help, the quality of the final version of this dissertation improved tremendously. I would like to separately thank Gerhad Woeginger for helping with the $\mathcal{NP}$-hardness proof in Chapter 4. I would also like to express my gratitude to René de Koster and Peter de Langen, who kindly accepted to be an external dissertation committee member. Many other people contributed to the improvement of this dissertation in one way or another, I would like to thank all of them, especially Nico Dellaert for reading and commenting on Chapter 5, and my ex-roommate Frank Karsten for his feedback on different parts of the dissertation. I would also like to thank El-Ghazali Talbi for his contribution to Chapter 2 of this dissertation, and Selen Kökten for her contribution to Chapter 3.

Doing my PhD at Eindhoven University of Technology and being part of OPAC has been a blessing for me. I would like thank all the current and former colleagues for the unforgettable pleasant memories and for facilitating such an enjoyable working environment. Thank you for the nice discussions during the coffee breaks, and the interesting talks during the PhD seminars. My special thanks to Youssef Boulaksil for being a good friend and companion in our travels to many exotic destinations (e.g., Tokyo, San Diego, Las Vegas, Los Angeles etc), and to Ola Jabali for being such a nice and pleasantly messy roommate.

Finally, I want to express my sincere appreciation for my proud family for their

unconditional love, support and prayers throughout the years. Last but definitely not least, I would like to ... my lovely wife Souhaila, my daughter Aya and my son Naïl. I really can not find the right words, simply because there are no words in any kind of language that can be enough to thank you. Nothing would have made sense without you. You brought light into my life in dark times, and made life easy in difficult times. Thank you for all that was and all that is to come.

The last four years were a milestone in my life. So, I have to admit that I started wrong when I said that a PhD is all about writing and debugging programs. It is certainly far more than only that. However, I chose to keep the first paragraph as it is. A PhD is also about sometimes starting "wrong" and then get it right.

Said Dabia,
November 2011

Contents

Chapter 1

Introduction

1.1. Optimization Problems

Yearly, over 1,721 billion ton kilometers of goods are transported on the European road networks. In the Netherlands, the turnover of the road transport sector was about 23 billion euro in 2008. Furthermore, transportation costs are responsible for up to 10% of a product selling price (Coyle et al., 1996). Thus, transportation is a key logistics activity that has a huge impact on national economies. The efficient utilization of transportation resources leads to substantial cost savings for all the parties involved in the transportation process (e.g., shippers, transporters etc). The efficient utilization of transportation entails different decision levels; strategic decisions (e.g., the location of delivery centers), tactical decisions (e.g., the type of fleets) and operational decisions (e.g, the routing of fleets and cargo loading). This thesis provides decision support tools for operational decisions faced by companies in the transportation sector as well as in other sectors.

Nowadays, organizations are operating in an increasingly complex and competitive environment. Consequently, they are faced with more challenging planning problems that need the appropriate knowledge and resources to deal with, and for which more detailed and accurate plans are required. Finding these plans is exactly what we call an *Optimization Problem*. In this line of thought, decision support tools are becoming more and more indispensable. Operations research forms the basis for such tools as it provides the necessary instruments such as mathematical modeling and mathematical programming. While mathematical modeling concerns the explicit formulation of problems, mathematical programming covers solution methods for the mathematical formulations. Their wide applicability led to several commercial

software packages that became available during the last decades, and which were fostered by technological developments (e.g., computer power) and algorithmic improvements.

The mathematical formulation of an optimization problem is represented by a set of decision variables that reflect the actions taken by decision makers (e.g., accept or reject an order), and a set of constraints that reflect the boundaries for these actions (e.g., transportation capacity). An optimization problem calls for the minimization (or the maximization) of an objective function. The objective function is a function of the decision variables; it guides the optimization process to a solution in the feasible space. The optimization process ends when a minimum (or maximum) is reached. When the objective function and the constraints are linear functions of the decision variables, and the variables are continuous, the optimization problem is called a linear program (LP). When integrality is imposed on the variables, it is called an integer program (IP), or a mixed integer program (MIP) if both integer and continuous variables are involved. Many optimization problems can be formulated as an MIP. In general, MIPs can be solved using three approaches:

- *Exact algorithms* lead to a solution that is proven to be optimal, i.e., there does not exist another feasible solution with a better objective value.
- *Heuristics* lead to a solution with no guarantee on its quality. Heuristics are used when the optimal solution is hard to find and when computation time is an issue.
- *Approximation algorithms* are basically heuristics that have a guarantee on the quality of the obtained solution. Moreover, in general, we have more understanding of the mechanism on which an approximation algorithm is based.

Although exact algorithms are less attractive in practice, as they usually require long computation times, their analysis often provides insights into the problem structure and reveals its complexity. Nowadays, many good heuristics and approximation algorithms are based on exact approaches. Moreover, many exact solution procedures incorporate heuristics to speed up computation times. In this thesis, we investigate both an approximation algorithm and an exact approach for two well-known optimization problems, namely the *Vehicle Routing Problem* (VRP) and the *Knapsack Problem* (KP). Furthermore, heuristics are used in part of the exact approach.

The rest of this chapter is organized as follows. Sections 1.2 and 1.3 describe the Vehicle Routing Problem (VRP) and the Knapsack Problem (KP) respectively. In Section 1.4, the applications that motivate the research conducted in this thesis are presented. Section 1.5 is devoted to the solution methods used in the thesis. Finally, an overview of the thesis is provided in Section 1.6.

1.2. The Vehicle Routing Problem

In the transportation sector, decision makers are, on a daily basis, faced with solving the Vehicle Routing Problem (VRP). The VRP was first introduced by Dantzig and Ramser (1959). It concerns the determination of a set of routes starting and ending at a depot, in which the demand of a set of geographically scattered customers is fulfilled. Each route is traversed by a vehicle with a fixed and finite capacity, and each customer must be visited exactly once by exactly one vehicle. The total demand delivered in each route should not exceed the vehicle's capacity. Due to its practical relevance and theoretical importance, a considerable amount of research has been devoted to solve the different variants of the VRP (see Laporte (1992), Toth and Vigo (2002), and Laporte (2007) for some reviews). In this thesis, we mainly treat the Vehicle Routing Problem with Time Windows (VRPTW). For good reviews on the VRPTW, the reader is referred to Bräysy and Gendreau (2005a,b); Kallehauge (2008) and Gendreau and Tarantilis (2010).

The VRPTW is defined on a graph $G = (V, A)$ where $V = \{0, 1, ..., N, N+1\}$ is the set of nodes. $V_c = V \setminus \{0, N+1\}$ represents the set of customers while nodes 0 and $N+1$ represent the depot. Nodes 0 and $N+1$ will be the start and end, respectively, of any route. $A = \{(i, j) : i \neq j \text{ and } i, j \in V\}$ is the set of all arcs between the nodes. Let K be the set of homogeneous vehicles each with a finite capacity Q. Service of node $i \in V$ can only start within its time window $[a_i, b_i]$. Furthermore, let q_i be the demand and s_i be the service time of node i. We assume $s_0 = s_{N+1} = q_0 = q_{N+1} = 0$. To each arc we associate the cost c_{ij}. Moreover, we let τ_{ij} denote the travel time from node i to node j.

Several mathematical formulations are proposed for the VRPTW. We present an MIP arc flow formulation based on the flow variables $x_{ijk}, (i, j) \in A, k \in K$, that take the value 1 if and only if the arc (i, j) is traversed by the vehicle k, and the time variables $\omega_{ik}, i \in V, k \in K$, representing the start time of service at node i. Furthermore, we denote, for every subset $A' \subseteq A$ and vehicle $k \in K$, $x^k(A') = \sum_{(i,j)\in A'} x_{ijk}$, and we let $\gamma^+(j)$ and $\gamma^-(j)$ be the set of arcs originating from j and the set of arcs ending in j respectively. The three-index formulation of the VRPTW is as follows

$$\min z = \sum_{k\in K} \sum_{(i,j)\in A} c_{ij} x_{ijk} \tag{1.1}$$

subject to

$$\sum_{k\in K} x^k(\gamma^+(i)) = 1 \qquad \forall i \in V_c \tag{1.2}$$

$$x^k(\gamma^+(0)) = 1 \qquad \forall k \in K \tag{1.3}$$

$$x^k(\gamma^+(j)) = x^k(\gamma^-(j)) \qquad \forall k \in K, \forall j \in V_c \tag{1.4}$$

$$x^k(\gamma^-(N+1)) = 1 \qquad \forall k \in K \tag{1.5}$$

$$\omega_{ik} + s_i + \tau_{ij} \leq \omega_{jk} + (1 - x_{ijk})M \qquad \forall k \in K, \forall (i,j) \in A \tag{1.6}$$

$$a_i \leq \omega_{ik} \leq b_i \qquad \forall k \in K, \forall i \in V \tag{1.7}$$

$$\sum_{i \in V} q_i x^k(\gamma^+(i)) \leq Q \qquad \forall k \in K \tag{1.8}$$

$$w_{ik} \geq 0 \qquad \forall k \in K, \forall i \in V \tag{1.9}$$

$$x_{ijk} \in \{0,1\} \qquad \forall k \in K, \forall (i,j) \in A \tag{1.10}$$

The objective function (1.1) expresses the total cost to be minimized. Constraints (1.2) ensure that every customer is assigned to exactly one vehicle. Constraints (1.2) are sometimes referred to as the *coupling constraints*, as they are the only constraints that link all the vehicles in K. Constraints (1.3)-(1.5) are related to the flow of arcs on the path traversed by vehicle $k \in K$. Furthermore, constraints (1.6) and (1.7) guarantee feasibility with respect to time consideration. M is a very large number. Constraints (1.8) make sure that the vehicles' capacity is respected. Finally, constraints (1.9) ensure that the time variables are non-negative, and constraints (1.10) impose binary conditions on the flow variables.

The exact optimization, and even finding a feasible solution (Savelsbergh, 1985), of the VRPTW is $\mathcal{NP}-$complete. In fact, the VRPTW is a generalization of the VRP, which is a generalization of the TSP. The TSP is proven to be $\mathcal{NP}-$complete by Karp (1972).

1.3. The Knapsack Problem

Due to its several applications in industry and finance, the Knapsack Problem has been extensively studied since the work of Dantzig and Ramser (1959). Moreover, the interest in the knapsack problem is also due to its theoretical importance as it frequently occurs as a subproblem in more complicated combinatorial optimization problems. These problems will benefit from any improvement in solving the knapsack problem. For instance, a knapsack problem has to be solved to derive a bounding function for the vehicle routing problem (Righini and Salani, 2006).

The knapsack problem calls for the selection of a subset of some given set of items, each with a profit and a weight, such that the corresponding sum of profits is maximized and the sum of weights does not exceed the knapsack capacity. The knapsack problem has many practical applications. Consider an investor who has to choose among N projects the ones to invest in given a limited budget of β euros. Each

selected project j contributes with a profit $p_j, j = 1, ..., N$, and incurs an investment cost w_j. The optimal investment portfolio can be found by solving a knapsack problem. Furthermore, many problems that appear in cargo loading, cutting stock, and financial management may be formulated as a knapsack problem (Kellerer et al., 2005). Beside its practical relevance, the knapsack problem is an interesting problem from a theoretical point of view. In fact, many general problems can be translated to a knapsack problem. Moreover, the knapsack problem appears as a subproblem in many more complex problems. For instance, the knapsack problem appears as subproblem when solving the Generalized Assignment Problem, which is frequently used in solving the vehicle routing problem (Pisinger, 1995; Laporte, 1992).

In the literature, many variants of the knapsack problem are formulated and solved. The variants of the knapsack problem include the bounded knapsack problem (Pisinger, 2000), the unbounded knapsack problem (Kellerer et al., 2005), the multi-dimensional knapsack problem (Freville, 2004), the multiple-choice knapsack problem (Pisinger, 1997), and the quadratic knapsack problem (Pisinger, 2007). The reader is refereed to Martello and Toth (1990) for a good overview on the family of knapsack problems. If we further define x_j as the binary variable that takes the value 1 if and only if item j is selected in the solution, the 0-1 Knapsack Problem can be formulated as follows

$$\max z = \sum_{j=1}^{N} p_j x_j \tag{1.11}$$

subject to

$$\sum_{j=1}^{N} w_j x_j \leq \beta \tag{1.12}$$

$$x_j \in \{0, 1\}, \quad \forall \ \ j = 1, ..., N \tag{1.13}$$

The objective function (1.11) expresses the total profit to be maximized. Constraint (1.12) guarantees that the capacity of the knapsack is respected. Constraints (1.13) are needed to impose binary conditions on the variables $x_j, j = 1, ..., N$.

1.4. Time and Multiple Objectives

The research presented in this thesis approaches scheduling and routing problems from a time and a multiple objective perspective. The time perspective is reflected by capturing the dynamic nature of real-life scheduling and routing problems. In

Chapters 4 and 5, the time perspective relates to the vehicles' travel time in routing problems. In Chapter 4, in addition to travel time, customers' demand is also time-dependent. In case of perishable products, quality deteriorates as time elapses. Consequently, customers' demand decreases with time. In Chapter 3, costs (or profits) involved in the knapsack problem are time-dependent. The multiple objective perspective is reflected by capturing the complexity of the decision making process. In Chapters 3 and 4, multi-criteria objective functions are considered. In Chapter 2, a generic multi-objective scheduling problem is considered with cost parameters depending upon the state of the system. Time-dependent cost parameters are a special case when the state of the system is described as a function of time.

Section 1.4.1 highlights the time perspectives considered in Chapters 3, 4 and 5. Section 1.4.2, describes the multiple objective perspective treated in Chapters 2, 3 and 4. Table 1.1 illustrates the perspectives considered in each chapter.

Chapter	Multiple objectives	Time
2	X	
3	X	X
4	X	X
5		X

Table 1.1 Main features per chapter.

1.4.1 The time perspective

Nowadays, organizations are operating in a stochastic and dynamic environment. Consequently, the planning and the scheduling of their operations became harder. At the operational level, Logistic Service Providers are, on a daily basis, faced with the problem of deciding on the routing and the scheduling of their fleets. This decision is equivalent to solving a VRP where the total operational cost is minimized. The majority of research and commercial software packages consider this operational cost in terms of total travel time or total distance traveled. Moreover, travel time is considered to be equivalent to distance as it is implicitly assumed that vehicles travel with constant speed throughout their operating period. In reality, speed is unlikely to be constant throughout the day. Malandraki and Daskin (1992) define two main causes of speed variability. The first is associated with random events such as accidents. The second is due to more or less predictable events as road congestion caused by traffic density. In this thesis, we mainly deal with the latter.

Traffic congestion has a large impact on the quality and the feasibility of vehicle route plans. Clearly, traveling during harsh hours results in a much longer travel time due to congestion. Many surveys have shown that traffic congestion is a fast growing phenomenon (Jorritsma et al., 2008; Schrank and Lomax, 2007). Due to traffic congestion, travel times between customers depend on the time of the departure, and ignoring congestion may result in high travel times. Furthermore, even feasibility of the routes is not guaranteed as time windows, especially when they are tight, may not be satisfied due to the delay caused by traffic congestion. Therefore, to improve delivery reliability, models dealing with the vehicle routing problem should account for the time-dependency of travel times.

Similarly to Ichoua et al. (2003), we divide the planning horizon into time zones (e.g., morning, afternoon etc) where a different speed is associated with each of these zones. The resulting stepwise speed function is translated into travel time functions that do not allow overtaking. Such travel time functions satisfy the First-In First-Out (FIFO) principle. Taking time-dependency of travel times into account results in the Time-Dependent Vehicle Routing Problem (TDVRP). In the TDVRP, the operational cost is expressed in terms of total time traveled resulting in a harder optimization problem as vehicles' dispatch times at the depot are crucial. Consequently, a scheduling component is added to the routing decision. In fact, a later dispatch time at the depot may result in a reduced travel time as congestion might be avoided. Moreover, when delivery time windows are considered, it might be more beneficial to wait at the depot rather than at customers. A detailed review on the TDVRP is provided in Chapters 4 and 5.

Not only in the context of distribution and transportation planning, but in many other real-world problems, costs and profits vary over time. Considering the investor example described in Section 1.3, profits and investments incurred by selecting a project depend on the project's execution time (e.g., in the retail environment profits depend on whether it is Christmas or not). The optimal investment portfolio for the investor problem with time-dependent profits can be found by solving the time-dependent knapsack problem (TDKP). Note that, contrary to the time-independent knapsack problem, the sequence in which the projects are executed has an impact on the solution value. This implies the addition of a kind of routing component to the scheduling decision. In Chapter 3, a review on the knapsack problems including the TDKP is provided.

1.4.2 The multiple objective perspective

The word *optimum* originates from the Latin word "optimus" which means "the best one". This suggests that human beings, when adopting an optimizing behavior,

strive to perform a given task in the best possible way. In other words, they want to achieve the optimum. This optimum is defined in a very remarkable way as it is unique and proven. Its superiority is a fact that leaves no room for doubts and possibilities. This explains why decision makers are obsessed by such a dominant concept when confronted with an optimization problem. However, the optimum is only defined when an optimization problem is based on a single criterion, which is unlikely to be the case in most real-life situations. It is difficult, if not impossible, to reflect in a single objective the complexity of organizations (e.g., production processes, distribution networks, humans etc). It is therefore realistic to assume that a decision involves several objectives that should simultaneously be achieved. In fact, optimization problems should be solved according to multiple conflicting objectives that together thwart the uniqueness of a best solution. In other words, the "optimus" is no more sufficient to represent the ideal solution. Consequently, one needs to find a set of solutions that capture the trade-offs between the objectives which led to the development of multi-objective optimization.

The roots of multi-objective optimization go back to the nineteenth century in the work of Edgeworth (1881) and Pareto (1896). At that time, concepts from multi-objective optimization were applied in the field of economics to gain insights into income and wealth distribution. During the last three decades, the discipline of multi-objective optimization has prospered and grown to include many application domains ranging from operations research (Ehrgott, 2005; Zeleny, 1982) to biology (Ecker et al., 2002; Greenberg et al., 2004). Roughly speaking, multi-objective optimization is the selection of compromise solutions from a set of alternatives with respect to multiple criteria or objective functions. Formally, a multi-objective optimization problem can be formulated as follows:

$$\begin{cases} \text{vopt } G(S) = [g_1(S), g_2(S), \ldots, g_n(S)] \\ s.t.\ S \in \mathbf{S} \end{cases}$$

where n ($n \geq 2$) is the number of objectives, $S = [s_1, ..., s_\theta]$ is a vector representing a state from the feasible state space $\mathbf{S}$. The state S is the results of a series of decisions. $G(S) = [g_1(S), g_2(S), ..., g_n(S)]$ is an objective vector representing the image of the state S in the objective space. The operator vopt stands for *vector optimization*.

Research conducted on the multi-objective vehicle routing problem and the multi-objective knapsack problem is scarce compared to the single-objective variants. For an extensive literature review on multi-objective VRP models, we refer to Jozefowiez et al. (2008). Other references on the multi-objective VRP are provided in Chapter 4. For a review on the multi-objective knapsack problem, we refer to the work of Erlebach et al. (2002). More on the multi-objective knapsack problem is provided in Chapter 3.

1.5. Solution Methods

This section covers the methodology used in this thesis. It is not intended to be an in-depth survey, but primarily a brief description to facilitate the understanding of the main concepts presented in this thesis. When needed, references are provided that supply more details on the subjects. In Chapters 2-4, an approximation based on the *trimming* technique for *Dynamic Programming* is presented. The approximation is presented in Chapter 2 in a generic fashion for multi-objective scheduling problems, and is validated in Chapter 3 on the time-dependent multi-objective knapsack problem. In Chapter 4, the approximation is applied to a specific time-dependent multi-objective *Single* Vehicle Routing Problem with Time Windows (SVRPTW). Yet, the trimming action (i.e., cleaning up the state space) is slightly different as, contrary to Chapters 2 and 3, the trimming action is dependent on the current iteration of the dynamic programming. In Chapter 5, a *Branch-and-Cut-and-Price* algorithm is developed to solve the time-dependent vehicle routing problem with time windows. The *Dantzig-Wolfe Decomposition* is applied on an arc flow based formulation resulting in a *master problem* and a *pricing subproblem*. While the master problem is solved by *Column Generation*, the pricing subproblem is solved by dynamic programming (also called *labeling algorithm*).

1.5.1 Dynamic programming

Since the work of Bellman (1956), Dynamic Programming has evolved as a powerful framework for solving optimization problems. In contrast to the branch-and-bound paradigm, where a large problem is solved at the beginning and smaller subproblems are solved as we precede with the branching, in dynamic programming the problem is first broken down into smaller subproblems that are easier to solve. The information obtained from solving these smaller subproblems is used to solve larger problems in later iterations of the dynamic programming. For the VRPTW, the first dynamic programming approach is presented in Kolen et al. (1987) which was inspired by the work of Christofides et al. (1976) on dynamic programming for the VRP. Dynamic programming for the VRP is based on the dynamic programming algorithm for the Traveling Salesman Problem (TSP) proposed by Held and Karp (1962) and Bellman (1962). In the dynamic programming for the TSP, a state $(S, j), j \in S, S \subseteq V_c$, is defined, which represents a minimum-length tour with cost $C(S, j)$ starting at the depot 0, visiting all nodes in S and ending in node j. V_c represents the entire set of nodes to be visited. If c_{ij} is the cost of traveling from node i to node j, the dynamic programming recursion is written as: $C(S, j) = \min_{i \in S \setminus \{j\}} \{C(S \setminus j, i) + c_{ij}\}$.

Mostly, dynamic programming algorithms are computationally very expensive.

Therefore, dynamic programming is usually used as a heuristic or an approximation. In Malandraki and Dial (1996), a restricted dynamic programming heuristic is proposed to solve the time-dependent TSP. In each iteration of the restricted dynamic programming, only a subset (hence "restricted") of the best solutions is kept and used to compute solutions in the next iteration. Sahni (1976) is probably the first that transformed a dynamic programming algorithm into a "polynomial" approximation by rounding the data of the instance. Later, Garey and Johnson (1978) introduced the term *Fully Polynomial Approximation Scheme* (FPTAS) to distinguish this type of approximation schemes. Rounding the data of the instance results in an easier problem to solve. In Ibarra and Kim (1975), a different technique is adopted (i.e., the trimming technique). The main idea is to clean up the state space after each iteration of the dynamic programming, and therefore keep only a polynomial amount of data. In Woeginger (2000), the trimming technique is applied to derive an FPTAS for different optimization problems. An FPTAS is an algorithm that, for any $\epsilon > 0$, runs in time polynomial in the size of the instance and in $1/\epsilon$. Dynamic programming also proved to be powerful in solving knapsack problems. For an overview see, e.g., Toth (1980) and Pisinger (1997).

1.5.2 Dantzig-Wolfe decomposition

By closely looking at the arc flow based formulation of the VRPTW presented in Section 1.2, we can observe that only constraints (1.2) are coupling all the vehicles, while the other constraints are dealing with each vehicle separately. Such a structure suggests adopting a Dantzig-Wolfe decomposition (DWD) to break up the overall problem into a master problem and a subproblem, also called the pricing subproblem, for each vehicle. DWD for linear programs was introduced by Dantzig and Wolfe (1960). For DWD for integer programs see for example Wolsey (1998). In case of the VRPTW, the master problem is a difficult integer programming problem, therefore it is usually relaxed and solved. The pricing subproblem is translated into a constrained shortest path problem. Often, the master problem is stated as a set partitioning problem without describing the underlying DWD on which it is based. In Appendix A, we shortly describe the steps of the DWD. For more details, see, e.g., Barnhard et al. (1998) and Lübbecke and Desrosiers (2005).

1.5.3 Column generation

The solution of the relaxed master problem is often fractional. As we are looking for an integer solution, solving the linear programming relaxation of the master problem (the *LP master problem*) provides a lower bound on the value of the set partitioning

problem. To overcome the huge number of columns (i.e., paths) in the LP master problem, the formulation (A.8)-(A.10), in Appendix A, is solved using only a small subset $\Omega' \subseteq \Omega$ of columns resulting in a *restricted* LP master problem. Usually we start with columns visiting only one customer, meaning paths with the form *depot-i-depot*, where i is a customer. Generating new columns is done by solving a pricing subproblem by using the information available from the current solution of the restricted LP master problem, more specifically, the vector of dual variables π corresponding to constraints (A.9) (see Appendix A). In case of the VRPTW, the pricing problem is an Elementary Shortest Path Problem with Resource Constraints (ESPPRC) where the constrained resources are vehicles' capacity and time windows. By modifying the objective function of the pricing problem, we can identify the columns with negative reduced cost which, when added to the restricted LP master problem, improve its objective function. The column generation terminates when no columns with negative reduced cost exist.

In Appendix B, we provide a short description of column generation. For a detailed overview of column generation algorithms, the reader is referred to Lübbecke and Desrosiers (2005).

1.5.4 Branch-and-bound in column generation

Usually, solving the LP master problem mostly provides a fractional lower bound on the value of the integer master problem. Therefore, column generation is embedded in a *branch-and-bound* framework to ensure integrality. The gap between the obtained lower bound and the integer optimal value has an important impact on the size of the branching tree (more branching is needed when the gap is larger). For traditional integer programs, branching is usually performed by choosing a fractional variable and create two branches, one where the value of the chosen variable is less than its rounded down value, and another where the value of the variable is greater than its rounded up value.

For the VRP, branching is done by setting a fractional variable to 0 for the one branch and to 1 for the other branch. Branch-and-bound frameworks has been used extensively to solve the VRP. The reader is referred to Laporte and Nobert (1987) for a review on the branch-and-bound algorithms proposed in the literature.

1.5.5 Cutting planes

For general integer programs, cutting planes are valid inequalities that cut off a fractional solution of their LP relaxation without losing any of the feasible integer

solutions. In case of the VRPTW, adding cuts to the master problem can significantly improve the lower bound obtained by solving the LP master problem resulting in smaller branching trees (i.e., the gap between the obtained lower bound and the optimal solution can easily be closed). Valid inequalities of the original problem (1.1)-(1.10) can be easily reformulated into the master problem. In other words, valid inequalities of the original formulation are also valid inequalities for the integer master problem. In Appendix C, we show how dual variables corresponding to valid inequalities written in the original variables are dealt with in the pricing problem. Valid inequalities can also be added in the set partitioning formulation (A.8)-(A.10) presented in Appendix A. However, adding valid inequalities in the set partitioning results in a much more complicated pricing problem as the corresponding dual variables can not be expressed in the variables of the original formulation (i.e., the x variables). Consequently, additional resources are needed to handle the additional cost component in the objective function of the pricing problem. For more detail, see for instance Jespen et al. (2008).

1.6. Overview of the Thesis

In Chapter 2, we propose a generic approach to deal with Multi-Objective Scheduling Problems (MOSPs). The aim is to determine the set of Pareto solutions that capture the trade offs between the different objectives. Due to the complexity of MOSPs, an efficient approximation based on dynamic programming is developed. The approximation has a provable worst case performance guarantee. Even though the approximate Pareto set consists of fewer solutions, it represents a good coverage of the true set of Pareto solutions. We consider generic cost parameters that depend upon the state of the system. Chapter 3 presents a validation of the methodology described in Chapter 2. We consider the time-dependent multi-objective knapsack problem (TDMOKP). Numerical results are presented for a multi-objective function with four objectives, showing the value of the approximation in the special case when the state of the system is expressed in terms of time.

In Chapter 4, a single vehicle performs several tours to serve a set of geographically dispersed customers according to a predefined sequence. The vehicle has a finite capacity and is only available for a limited amount of time. Moreover, the tours' duration is restricted, and customers need to be delivered in their specific time windows. Travel times are time-dependent because of road congestion. Furthermore, customers' demand is non-increasing in time. We consider a multi-objective cost function in which we simultaneously minimize the total time traveled including waiting times at customers, and maximize the total demand fulfilled. An efficient

approximation based on dynamic programming is developed to approximate the set of Pareto solutions. To investigate the value of the approximation, numerical experiments are conducted on sets with 100 customers.

In Chapter 5, we present a Branch-and-Cut-and-Price algorithm for the Time-Dependent Vehicle Routing Problem with Time Windows (TDVRPTW). We capture road congestion by considering time-dependent travel times. That is, depending on the departure time at a customer, a different travel time is incurred. Because of time-dependency, vehicles' dispatch times at the depot are crucial as road congestion might be avoided. Due to its complexity, all known solution methods to the TDVRPTW are based on (meta-)heuristics. The decomposition of an arc flow based formulation leads to a set partitioning problem as the master problem, and a time-dependent shortest path problem with resource constraints as the pricing problem. The master problem is solved by means of column generation, and a modified labeling algorithm is used to solve the pricing problem. We introduce new dominance criteria that allow the domination of more labels. For our numerical results, we modified Solomon's data sets by adding time-dependency. Our algorithm is able to optimally solve about 70% of the instances with 25 customers, 47% of the instances with 50 customers and 18% of the instances with 100 customers.

The chapters of the thesis are based on the following working papers:

Chapters 2 and 3: Dabia, S, E-G. Talbi, T. van Woensel, A.G. de Kok. *Approximating Multi-Objective Time-Dependent Optimization Problems.* Beta working paper number wp 362, School of Industrial Engineering, Eindhoven University of Technology.

Chapters 4: Dabia, S, T. van Woensel, A.G. de Kok. *A Dynamic Programming Approach to Multi-Objective Time-Dependent Capacitated Single Vehicle Routing Problems with Time Windows.* Beta working paper number wp 313, School of Industrial Engineering, Eindhoven University of Technology.

Chapters 5: Dabia, S, S. Ropke, T. van Woensel, A.G. de Kok. *Branch and Cut and Price for the Time-Dependent Vehicle Routing Problem with Time Windows.* Beta working paper number wp 361, School of Industrial Engineering, Eindhoven University of Technology.

Chapter 2

Approximating Multi-Objective Scheduling Problems

2.1. Introduction

Many optimization problems encountered in practice are multi-objective in nature, i.e., different objectives are conflicting and equally important. Many times, it is not desirable to drop some of them or to optimize them in a hierarchical manner. For instance, while designing a product, many criteria are taken into account: the product's reliability should be maximized, while the cost and the environmental impact should be minimized. Obviously, the amount of examples is infinite.

Contrary to single-objective optimization problems where the optimal value is unique, the aim of Multi-Objective optimization Problems (in short, MOPs) is to determine the set of solutions representing the trade-offs between the different conflicting objectives. This set of solutions is denoted as the set of Pareto solutions or the Pareto front. In this line of thought, decision makers are presented with the entire Pareto front (rather than a single solution) to select a solution (or a region of solutions) depending on their preferences. Although the roots of multi-objective optimization go back to the nineteenth century in the work of Edgeworth (1881) and Pareto (1896), most optimization problems dealt with are single-objective. In fact, objective functions are usually reduced to a composite single objective function by using a (weighted) sum of the various objectives. It is argued that solutions obtained as such might represent only a subset of the entire set of Pareto solutions, and therefore could lead to suboptimal managerial decisions (Ehrgott, 2005; Miettinen, 1999; Talbi, 2009).

In multi-objective decision making, the number of Pareto solutions increases with the size of the problem, mainly with the number of objectives. Therefore, multi-objective decision making is very challenging. In fact, most multi-objective problems are $\mathcal{NP}-$hard. Hence, it is computationally very expensive to compute the complete Pareto front. Furthermore, multi-objective decision making does not end when the Pareto front is found. In practice, only a single solution (or a region of solutions), taking decision makers preferences into account, needs to be implemented. There exist several methods allowing the selection of a solution from the Pareto front (Ferreira et al., 2007). These methods might not converge easily when the size of the Pareto front is very large. Consequently, many researchers direct their efforts on approximating the Pareto front to reduce the complexity of the applied algorithms and produce good approximations (i.e., approximate Pareto fronts) of the Pareto front. Approximate Pareto fronts contain fewer solutions, which facilitate the selection of a final solution. However, a good approximate Pareto front should properly represent the real Pareto front.

In this chapter, an approximation algorithm based on the trimming method for dynamic programming is proposed for *Multi-Objective Scheduling Problems* (MOSPs). The multi-dimensional state space is partitioned into intervals with exponentially increasing size. Each interval defines a cluster of states considered to be *very close* to each other. From each cluster, only one state is kept and the dynamic programming is adapted to the partitioned state space. In this way, in each iteration of the dynamic programming, only a polynomial number of states is processed. The approximation has a provable performance guarantee. Even with fewer solutions, the resulting approximate Pareto front still properly covers the real Pareto front as each Pareto solution is represented by at least one approximate Pareto solution. The proposed approximation can be applied to the multi-objective version of a variety of well-known optimization problems for which a dynamic programming formulation is possible (e.g., knapsack problems, shortest path problems, variants of vehicle routing problems, job scheduling problems etc). Furthermore, we consider a generic cost structure where costs depend on the state of the system. In many practical situations, cost parameters are not constant. In the transport sector, for instance, carriers work with tariff sheets where costs are computed depending on the utilization of their fleets. In fact, the tariff depends on the truck load or on the total kilometers traveled. In Chapter 3, we show the value of the approximation by applying it to the multi-objective knapsack problem (with 4 objectives) where the state of the system is expressed as a function of time.

The contributions of this chapter are summarized as follows. A generic approximation is proposed which can be applied to multi-objective scheduling problems for which a dynamic programming formulation is possible. The approximation generates

an approximate Pareto front with fewer solutions. The approximate Pareto front represents a very good coverage of the real Pareto front. Additionally, the approximation's worst case performance guarantee is provable. The approximation is flexible in the sense that the decision maker can choose different precision levels for the different objectives. In fact, the decision maker might be willing to tolerate more error for less sensitive objectives (i.e., with a "flat" cost structure). Finally, we are dealing with a class of realistic MOSPs for which costs are state-dependent. For instance, in a traffic network, travel costs are a function of travel times which change depending on the state of the traffic network (e.g., due to congestion).

This chapter is organized as follows. Section 2.2 reviews the literature relevant to MOPs. Section 2.3 is devoted to the introduction of the main concepts related to MOPs. Section 2.4 describes a generic MOSP, the input structure and the dynamic programming formulation for the MOSP. In Section 2.6, an approximation of the Pareto front is developed and the main results of the chapter are derived. Finally, Section 2.7 concludes with a summary of the main results.

2.2. Literature Review

As in single-objective optimization, MOPs can be divided into two categories: those whose solutions are encoded with *real-valued* variables, also known as *continuous MOPs*, and those where the solutions are encoded using *discrete* variables, known as multi-objective combinatorial optimization problems *(MOCO)*. In the class of continuous MOP, usually an infinite number of Pareto solutions composes the Pareto front whereas in combinatorial MOPs, the Pareto front is finite. Most heuristics for solving MOPs are designed to deal with continuous MOPs using, for instance, multi-objective simplex (Zeleny, 1982; Steuer, 1986). In the last decade, there is a growing interest in solving combinatorial MOPs. However, in most cases, they are bi-objective optimization problems. Furthermore, there is a lack of test instances for real-life combinatorial MOPs, especially problems with many objectives (Ishibuchi et al., 2008; Liefooghe et al., 2007) and dynamicity (Farina et al., 2004).

The study of computational complexity classes for MOCO started with the work of Serafini (1986), and Papadimitriou and Yannakakis (2002), where a connection is made between the results obtained in single-objective combinatorial optimization and the multi-objective field for several optimization problems. Serafini (1986) depictes nine possible definitions for MOCO problems and establishes reductions between them in order to facilitate obtaining complexity results. He shows that the following definition (denoted as V8 in his article) can be considered as a standard reference version to measure the computational complexity of MOCO problems. The definition

can also be seen as the decision problem associated with a MOCO problem.

Definition 2.1 (Generic definition of MOCO by Serafini (1986)) Given $z \in Z^n$, does there exist a solution x to MOCO such that $g_i(x) \leq z_i, 1 \leq i \leq n$?

where the functions g_i reflect some measures of interest, and $g_i(x)$ is computable in polynomial time. A $\mathcal{NP}-$hard single-objective problem implies a $\mathcal{NP}-$hard character to its multi-objective extensions. In the multi-objective case, the $\mathcal{NP}-$hardness appears for the majority of problems. For example, $\mathcal{NP}-$completeness is proved for shortest path problems, assignment problems and minimum maximal matching by Serafini (1986); for the minimum weight spanning tree by Camerini and Vercellis (1984); and for the max-linear spanning tree by Hamacher and Ruhe (1994).

Similarly to single-objective optimization problems, MOPs can be solved by means of *exact* and *approximate* algorithms. In the literature, more attention has been devoted to bi-criteria optimization problems by using exact methods such as branch-and-bound algorithms (Sen et al., 1988; Ulungu and Teghem, 1995; Visee et al., 1998; Sayin and Karabati, 1999; Lemesre et al., 2007a), branch-and-cut (Jozefowiez et al., 2007), *A* algorithm* (Stewart and White, 1991; Mandow and Millan, 1996), and dynamic programming (White, 1982; Carraway et al., 1990). Because of the complexity of MOPs, exact methods are only effective for problems with small instances and with no more than two criteria. There exist some new advances in this area, with several exact approaches proposed in the literature for bi-objective (Lemesre et al., 2006; Laumanns et al., 2004; Lemesre et al., 2007b) and multi-objective problems (Lemesre et al., 2006). Approximate methods are mainly used to solve large-scale problems and when multiple criteria are involved. They can be divided into two classes: on the one hand algorithms that are only applicable to a specific problem. Such algorithms are developed based on some knowledge on the structure of the problem at hand. On the other hand, meta-heuristics which are of general purpose, in the sense that they can be applicable to a large variety of MOPs. A unifying view for analyzing, designing and implementing multi-objective meta-heuristics is provided in the book by Talbi (2009). The main drawback of meta-heuristics is the lack of guarantee with regard to the quality of the approximate Pareto front. Moreover, the resulting approximate Pareto fronts might not properly cover the real Pareto front as they might contain very few solutions.

In the context of single-objective optimization problems, an ϵ-approximation scheme is an algorithm that, for every instance of the problem, finds an approximate solution that is guaranteed to be within a constant factor from optimal. Two classes of approximation schemes are mainly considered: Polynomial Time Approximation Scheme (PTAS) and Fully Polynomial Time Approximation Scheme (FPTAS). For any $\epsilon > 0$, a PTAS runs in time polynomial in the size of the instance, while an FPTAS

runs in time polynomial in the size of the instance and $1/\epsilon$. From a computational complexity point of view, FPTASs are the strongest approximation schemes with performance guarantee that can be obtained for $\mathcal{NP}-$hard optimization problems. The notion of approximation schemes can be generalized to the case of multi-objective optimization problems by considering, for each solution on the approximate Pareto front, worst case performance guarantees with regard to all criteria (Erlebach et al., 2002).

2.3. Definitions, Variables and Background

This section aims to give the relevant definitions used in the remainder of the chapter.

Definition 2.2 (Multi-objective optimization problem) A multi-objective optimization problem (MOP) is defined as

$$(MOP) = \begin{cases} \text{vmin } \{G(S) = [g_1(S), g_2(S), \ldots, g_n(S)]\} \\ s.t.\ S \in \mathbf{S} \end{cases} \tag{2.1}$$

where n ($n \geq 2$) is the number of objectives, $S = [s_1, \ldots, s_\theta]$ is the vector representing the state of MOP that encodes a solution resulting from a series of decisions, and $\mathbf{S}$ represents the feasible state space associated with equality and inequality constraints. $G(S) = [g_1(S), g_2(S), \ldots, g_n(S)]$ is the vector of costs corresponding to the state S.

The space to which the cost vector belongs is called the objective space. G is a mapping from the state space to the objective space which evaluates the quality of each state $[s_1, \ldots, s_\theta]$ by assigning an objective vector $[y_1, \ldots, y_n]$ (Figure 2.1). The decision maker is usually interested in the value of a state on each criterion. Therefore, the analysis of MOPs is usually done in the objective space. The set $G(\mathbf{S})$ represents the feasible points in the objective space, and $G(S) = [y_1, y_2, ..., y_n]$, where $y_i = g_i(S)$, is a point in the objective space. The operator vmin in Equation 2.1 stands for vector minimization. For a set of vectors, vmin generates only the non-dominated ones.

Definition 2.3 (Pareto dominance) A vector $u = [u_1, \ldots, u_n] \in \mathbb{R}^n$ is said to dominate a vector $v = [v_1, \ldots, v_n] \in \mathbb{R}^n$ (we write $u \preceq v$) if and only if no component of v is smaller than the corresponding component of u, i.e.,

$$\forall i \in \{1, \ldots, n\} : \ u_i \leq v_i.$$

Definition 2.4 (Pareto Optimality) A state $S^* \in \mathbf{S}$ is Pareto optimal if for every $S \in \mathbf{S}$, $G(S)$ does not dominate $G(S^*)$.

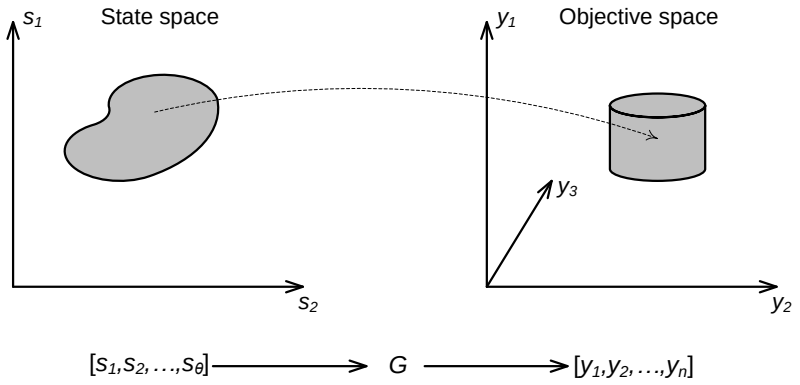

Figure 2.1 State space and objective space in a MOP.

A MOP involves the determination of a set of solutions known as the *Pareto optimal set*. The image of this set in the objective space is denoted as the *Pareto front*. We define the Pareto optimal set and the Pareto front as follows:

Definition 2.5 (Pareto optimal set) For a given MOP $(G, \mathbf{S})$, the Pareto optimal set is defined as $\mathcal{P}^* = \{S \in \mathbf{S} | \nexists S' \in \mathbf{S} : G(S') \preceq G(S)\}$.

Definition 2.6 (Pareto front) For a given MOP $(G, \mathbf{S})$ and its Pareto optimal set $\mathcal{P}^*$, the Pareto front is defined as $\mathcal{PF}^* = \{G(S) : S \in \mathcal{P}^*\}$.

The generation of the Pareto optimal set often turns out to be practically impossible or computationally too expensive. Therefore, good approximations of $\mathcal{PF}^*$ are desirable. We state the following definitions:

Definition 2.7 (ϵ-dominance) Let $\epsilon > 1$ be a real number. A vector $u = [u_1, \ldots, u_n] \in \mathbb{R}^n$ is said to ϵ-dominate $v = [v_1, \ldots, v_n]$ (we write $u \preceq_\epsilon v$) if and only if

$$\forall i \in \{1, \ldots, n\} : \; u_i \leq \epsilon v_i.$$

Definition 2.8 (Quasi ϵ-dominance) Let $\epsilon > 1$ be a real number. A vector $u = [u_1, \ldots, u_n] \in \mathbb{R}^n$ is said to quasi ϵ-dominate $v = [v_1, \ldots, v_n]$ on a subset $I_c \subseteq \{1, \ldots, n\}$ (we write $u \preceq_\epsilon^{I_c} v$) if and only if

$$\forall i \in I_c : \; u_i \leq v_i \; \wedge \; \forall i \in \{1, \ldots, n\} \setminus I_c : \; u_i \leq \epsilon v_i.$$

Obviously, quasi ϵ-dominance implies ϵ-dominance.

Definition 2.9 (ϵ-Pareto optimality) A state $S^* \in \mathbf{S}$ is ϵ-Pareto optimal if for every $S \in \mathbf{S}$, $G(S)$ does not ϵ-dominate $G(S^*)$.

Definition 2.10 (ϵ-Pareto front) For a given MOP $(G, \mathbf{S})$, the ϵ-Pareto front is defined as $\mathcal{PF}^\epsilon = \{G(S) | \nexists S' \in \mathbf{S} : G(S') \preceq_\epsilon G(S)\ \}$(Figure 2.2).

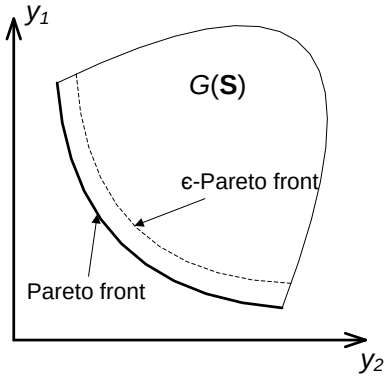

Figure 2.2 ϵ-Pareto concept. Sets of Pareto and ϵ-Pareto solutions

Furthermore, we define the following

$\lvert X \rvert$	: Size of the set X
$\lceil x \rceil$	: Nearest integer larger or equal to the real number x
$[\![a, b]\!]$	: The interval of integer numbers between the integers a and b (a and b included)
$\mathbb{N}$	: The set of natural numbers
$\mathbb{R}$	: The set of real numbers

2.4. Dynamic Programming for the MOSP

In this section, we consider a generic Multi-Objective Scheduling Problem (MOSP) that jointly minimizes n objectives. The derived results can easily be converted to the case of joint maximization, or when some objectives are minimized and some are maximized. In Section 2.4.1, we describe the structure of the input for any instance of MOSP. Section 2.4.2 specifies the dynamic programming formulation of MOSP. For illustration, two examples are provided in Section 2.5.

2.4.1 Structure of the input

We assume that for any instance of MOSP, the input decomposes into N vectors $Y_1, Y_2, ..., Y_N \in \mathcal{C}^m$. $\mathcal{C}$ is a set of mappings from $\mathbb{R}^\theta$ to $\mathbb{R}$, and $m \in \mathbb{N}$ is a positive number that may depend on the input. $\theta \in \mathbb{N}$ is a positive number that depends on MOSP but not on the numerical value of the input. For every $k \in [\![1, N]\!]$, the vector Y_k consists of m mappings $[y_{1,k}, \ldots, y_{m,k}]$ in $\mathcal{C}$. All input is encoded in binary.

2.4.2 Structure of the dynamic programming

The execution of the dynamic programming DP for MOSP consists of β iterations such that, in the k^{th} iteration, $k \in [\![1, \beta]\!]$, the input vector Y_{i_k}, $i_k \in [\![1, N]\!]$, is processed and a set of states $\mathbf{S_k}$ is generated. $\beta \in \mathbb{N}$ is a positive number that may depend on the numerical value of the input, and $\mathbf{S_k}$ is a subset of $\mathbb{R}^\theta$. Any state $S \in \mathbf{S_k}$ consists of the components $[s_1, \ldots, s_\theta]$, and describes a solution defined by the partial input $Y_{i_1}, Y_{i_2}, ..., Y_{i_k}$. Intuitively, the positive number θ specifies the amount of information needed to describe the state of MOSP. DP may process the same vector at several iterations. Moreover, not all vectors need to be processed by DP, and the sequence in which vectors are processed is important as a different sequence of the same vectors leads to a different state. Some of the state components may be constrained (e.g., due to due dates in a job scheduling problem, knapsack capacity in a knapsack problem, delivery time windows in a routing problem etc). We let $I_c \subseteq \{1, \ldots, \theta\}$ be the subset of indexes corresponding to the constrained state components.

Let $\mathcal{T}$ be a finite set of mappings from $\mathcal{C}^m \times \mathbb{R}^\theta$ to $\mathbb{R}^\theta$, and $\mathcal{F}$ a finite set of mappings from $\mathcal{C}^m \times \mathbb{R}^\theta$ to $\{0, 1\}$. We call $\mathcal{T}$ the set of transition mappings, and $\mathcal{F}$ the set of feasibility mappings. For every transition mapping $T \in \mathcal{T}$ there exists a corresponding feasibility mapping $F_T \in \mathcal{F}$. In fact, in the k^{th} iteration of DP, a transition from the set of states $\mathbf{S_{k-1}}$ to the set of states $\mathbf{S_k}$ is achieved through the mappings in $\mathcal{T}$, and the feasibility of the new generated states are checked by the mappings in $\mathcal{F}$ (i.e., infeasible states are discarded from the state space $\mathbf{S_k}$). That is, for every $S \in \mathbf{S_{k-1}}$ and $T \in \mathcal{T}$, the new generated state $T(Y_{i_k}, S)$ is feasible, and hence included in $\mathbf{S_k}$, if and only if $F_T(Y_{i_k}, S) = 1$.

The cost function G is a mapping from $\mathbb{R}^\theta$ to $\mathbb{R}^n$. For every state $S \in \mathbb{R}^\theta$, G calculates the cost vector $G(S) = [g_1(S), \ldots, g_n(S)]$ where $g_r(S) \in \mathbb{R}, r \in [\![1, n]\!]$, is the cost component corresponding to the r^{th} objective. The output of DP is the Pareto front $\mathcal{PF}^*$.

In the following, we state the assumptions regarding the relation of the cost function G, the transition mappings in $\mathcal{T}$ and the feasibility mappings in $\mathcal{F}$ to the concept of

dominance.

Assumption 2.1 For any two states $S, S' \in \mathbb{R}^{\theta}$, for any $T \in \mathcal{T}$, for any $Y \in \mathcal{C}^m$, and for any $F \in \mathcal{F}$, if $S \preceq S'$, it holds that:

1. $T(Y, S) \preceq T(Y, S')$.
2. $F(Y, S') \leq F(Y, S)$.
3. $G(S) \preceq G(S')$.

Conditions 1 and 3 of Assumption 2.1 imply that dominance is preserved by the mappings in $\mathcal{T}$ and the mapping G. Condition 2 guarantees that any feasible extension of the partial solution encoded by the state S' is also a feasible extension of the partial solution encoded by the state S. Given Assumption 2.1, the following lemma shows that only non-dominated states need to be stored in each iteration of DP.

Lemma 2.1 Let $k \in [\![1, \beta]\!]$. For any non-dominated state S in the set of states $\mathbf{S_k}$, there exists a non-dominated state S' in the set of states $\mathbf{S_{k\text{-}1}}$, such that $S = T(Y_{i_k}, S')$ for some $T \in \mathcal{T}$ and $Y_{i_k} \in \mathcal{C}^m, 1 \leq i_k \leq N$.

Proof: Let $S \in \mathbf{S_k}$ be a non-dominated state in $\mathbf{S_k}$. Let $\overline{\mathbf{S}}_{\mathbf{k\text{-}1}} \subseteq \mathbf{S_{k\text{-}1}}$ be the subset of states in $\mathbf{S_{k\text{-}1}}$ that lead to the state S. Let us assume that there is a state $S" \in \mathbf{S_{k\text{-}1}} \setminus \overline{\mathbf{S}}_{\mathbf{k\text{-}1}}$ that dominates all the states in $\overline{\mathbf{S}}_{\mathbf{k\text{-}1}}$. Due to Condition 1 of Assumption 2.1, for any $T \in \mathcal{T}$ and $Y_{i_k}, 1 \leq i_k \leq N$, it holds that for all $\overline{S} \in \overline{\mathbf{S}}_{\mathbf{k\text{-}1}}$, $T(Y_{i_k}, S") \preceq T(Y_{i_k}, \overline{S})$. Condition 2 of assumption 2.1 ensures that $T(Y_{i_k}, S")$ is feasible, and hence is an element of the state space $\mathbf{S_k}$. Consequently, $T(Y_{i_k}, S")$ dominates S in $\mathbf{S_k}$, which contradicts the fact that S is non-dominated in $\mathbf{S_k}$. This completes the proof of Lemma 2.1. □

The iterative structure of DP is summarized in Algorithm 1. In the next section, we provide two illustrative examples for the concepts introduced in this section.

2.5. Examples

In the previous section, the structure of the input in MOSP and the dynamic programming are presented in a rather abstract fashion. In this section, we present two illustrative examples. First, we present a multi-objective job scheduling on identical machines. Then, restriction on the availability of some machines is added in the second example.

Algorithm 1 The dynamic programming DP

```
1. Initialize S_0
2. for all 1 ≤ k ≤ β do
3.     S_k := ∅
4. end for
5. for k = 1 to β do
6.     for all S ∈ S_{k-1} do
7.         for all T ∈ 𝒯 do
8.             for i = 1 to N do
9.                 if F_T(Y_i, S) = 1 then
10.                    S_k := S_k ∪ T(Y_i, S)
11.                end if
12.            end for
13.        end for
14.    end for
15.    S_k := vmin{S_k}
16. end for
17. 𝒫ℱ* = vmin{G(S) : S ∈ S_k, 1 ≤ k ≤ β}
```

2.5.1 Multi-objective job scheduling on identical machines

The input consists of N jobs $J_1, J_2, \ldots, J_N$. All jobs are available at time 0 for scheduling without preemption on l parallel and identical machines $M_1, \ldots, M_l$. Each job $J_k, 1 \leq k \leq N$, is processed only once and incurs a processing time. The objective is to minimize the total processing times on all machines.
We let $n = \theta = |\mathcal{T}| = l$, $\beta = N$ and $m = |\mathcal{F}| = 1$. A state $S = [s_1, \ldots, s_l]$ in $\mathbf{S_k}$ describes the partial schedule consisting of jobs $J_{i_1}, \ldots, J_{i_k}$, where $J_{i_p}, 1 \leq i_p \leq N$, is the job added in the p^{th} iteration, and s_j represents the total processing time on machine M_j. We set $\mathbf{S_0}$ to $\{[1, \ldots, 1]\}$. For each $k \in [\![1, N]\!]$, the input vectors are:

$$Y_k = [y_{1,k}] \quad \text{for all} \quad k \in [\![1, N]\!] \tag{2.2}$$

where $y_{1,k}$ is a mapping from $\mathbb{R}^l$ to $\mathbb{R}$ that for every state $S \in \mathbb{R}^l$, computes job's J_k processing time as

$$p_k(S) = \ln\left(1 + \sum_{j=1}^{l} x_{kj} s_j\right) \tag{2.3}$$

x_{kj} is a binary variable that takes the value 1 if and only if job J_k is scheduled on machine M_j. The finite set of transition mappings $\mathcal{T}$ consists of the mappings

$T_1, \ldots, T_l$ such that

$$T_j(Y_k, S) = [s_1, \ldots, s_j + \ln(1 + s_j), \ldots, s_l] \tag{2.4}$$

In other words, the transition mapping T_j plans jobs on machine M_j. The set of feasibility mappings $\mathcal{F}$ consists of only one mapping F_1 that maps any element in $\mathcal{C} \times \mathbb{R}^l$ to 1. In other words, the new generated states are always feasible. Finally, we set

$$G(S) = [s_1, \ldots, s_l] \tag{2.5}$$

2.5.2 Multi-objective job scheduling on identical machines with limited availability

We consider the same problem as described in the previous section with the only difference that some of the machines have limited availability (e.g., due to maintenance activities). Let us assume that machines M_1 and M_2 are only available in the time interval $[0, a]$, and that there are no restrictions on the availability of the other machines. In this case, we have feasibility issues as jobs can not be processed on machines M_1 and M_2 outside the interval $[0, a]$. Therefore, the finite set of feasibility mappings consists of the mappings F_1, F_2 and F_3. F_1 is the same as in the previous example, and for $i = 2, 3$

$$F_i(Y_k, S) = \begin{cases} 1 & \text{if } \; s_i + \ln(1 + s_i) \leq a \\ 0 & \text{otherwise} \end{cases} \tag{2.6}$$

The mapping F_2 (respectively F_3) checks the feasibility of the new states generated by the transition mapping T_1 (respectively T_2). The mapping F_1 confirms the feasibility of the new states generated by the mappings $T_3, \ldots, T_l$. Furthermore, the set of indexes corresponding to the constrained state components is $I_c = \{1, 2\}$.

2.6. A Dynamic Programming Approximation

Usually, the size of the set of states $\mathbf{S_k}$ is very large. Consequently, the running time of DP is large too. Therefore, the determination of the Pareto front for most multi-objective optimization problems is a very difficult task. In the this section, an approximate dynamic programming (DP^ϵ) is presented.

Additional structure is added to the execution of DP by trimming the state space (i.e., deleting states very close to each other), which results in an approximate dynamic

programming denoted by DP^{ϵ}, where ϵ is the precision of DP^{ϵ}. In the k^{th} iteration of DP^{ϵ}, the new (untrimmed) state space $\widetilde{\mathbf{S}}_{\mathbf{k}}$ is obtained from the old state space generated in the $(k-1)^{th}$ iteration. Trimming the state space $\widetilde{\mathbf{S}}_{\mathbf{k}}$ results in the (trimmed) state space $\widetilde{\mathbf{S}}^{\epsilon}_{\mathbf{k}}$. The idea of adding this type of structure to the execution of algorithms was first introduced by Ibarra and Kim (1975). Sahni (1976) and Woeginger (2000, 2005) applied it to a variety of single-objective scheduling problems. We state the assumptions regarding the relation of the cost function G, the transition mappings in $\mathcal{T}$ and the feasibility mappings in $\mathcal{F}$ to the concept of $\epsilon-$ dominance and quasi $\epsilon-$ dominance.

Assumption 2.2 For any real number $\Delta > 1$, for any two states $S, S' \in \mathbb{R}^{\theta}$, for any $T \in \mathcal{T}$, for any $Y \in \mathcal{C}^m$, and for any $F \in \mathcal{F}$, if $S \preceq_{\Delta} S'$, it holds that:

1. $T(Y, S) \preceq_{\Delta} T(Y, S')$.
2. $G(S) \preceq_{\Delta} G(S')$.

Assumption 2.2 implies that $\epsilon-$dominace is preserved by the mappings in $\mathcal{T}$ and the mapping G. Intuitively speaking, if the state S' deviates by at most a factor Δ from the state S, the costs as well as the new generated state do not explode by more than the factor Δ. In other words, sudden increases in costs due to small deviations in the state are not allowed, which is reasonable from a real-life point of view (e.g., road congestion due to traffic density increases smoothly). However, a feasible extension of the partial solution encoded by the state S' is not necessarily a feasible extension of the partial solution encoded by the state S. To be able to claim the opposite, we need to have $s_i \leq s'_i$ for all $i \in I_c$. Therefore, we formulate the following assumption:

Assumption 2.3 Let $I_c \subseteq \{1, \ldots, \theta\}$ be the subset of indexes corresponding to the constrained state components. For any real number $\Delta > 1$, for any two states $S, S' \in \mathbb{R}^{\theta}$, for any $T \in \mathcal{T}$, for any $Y \in \mathcal{C}^m$, and for any $F \in \mathcal{F}$, if $S \preceq^{I_c}_{\Delta} S'$, it holds that:

1. $F(Y, S') \leq F(Y, S)$

Assumption 2.3 implies that any feasible extension of the partial solution encoded by the state S' is still a feasible extension of the partial solution encoded by a state S that quasi $\epsilon-$dominates S'. It is easy to check that Assumptions 2.2 and 2.3 are satisfied in the examples of section 2.5. In the sequel, we assume that MOSP has at most 1 constrained state component. For simplicity and without loss of generality, we assume that if $I_c \neq \emptyset$ then $I_c = \{1\}$, i.e., the first state component is constrained.

2.6.1 Setting up DP^{ϵ}

Formally, in each iteration k of DP^{ϵ}, the state space $\widetilde{\mathbf{S}}_{\mathbf{k}}$ can be represented by geometric points in the polyhedron $[0,b]^{\theta}$, where $b \in \mathbb{R}$ is an upper bound on the state components (i.e., $s_i \leq b$ for all $S \in \mathbf{S_k}$, $k \in [\![0,\beta]\!]$ and $i \in [\![1,\theta]\!]$). b may depend on the numerical value of the input. The polyhedron is cut into multiple boxes of exponentially increasing size. States contained by the same box are considered to be very close to each other. In each box, only the state with the smallest constrained state component is retained.

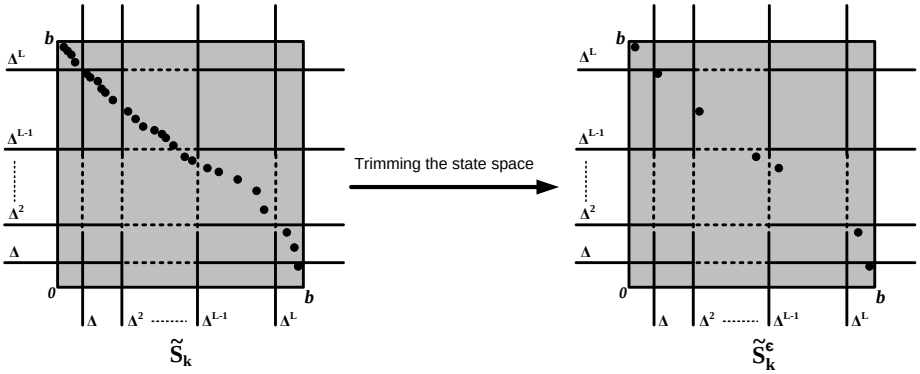

Figure 2.3 Partitioning and trimming the state space $\widetilde{\mathbf{S}}_{\mathbf{k}}$ in case $\theta = 2$.

The cuts are executed at the coordinates $\Delta^r, r = 1, ..., L$. The real number Δ is chosen as:

$$\Delta = 1 + \frac{\epsilon}{2\beta} \tag{2.7}$$

and L is chosen such that $\Delta^L \leq b$. We set:

$$L = \left\lceil \frac{\ln b}{\ln \Delta} \right\rceil \leq \left\lceil \left(1 + \frac{2\beta}{\epsilon}\right) \ln b \right\rceil$$

where ϵ is a real number between 0 and 1 representing the approximation's precision. Figure 2.3 illustrates the partitioning and the trimming of the state space in case $\theta = 2$. Boxes with exponentially increasing size are crucial as they result in a sort of logarithmic scale with a polynomial number of boxes. If the boxes' size increases linearly, the number of states kept after trimming will still be exponential. After trimming, only one state is retained in each box, thus a polynomial number of states is kept, and the amount of data to be processed is polynomially bounded. Furthermore, it is also important to have the small boxes close to the origin. In fact, in the early iterations of the dynamic programming, the generated states will usually be contained

in boxes close to the origin. Therefore, having small boxes avoids deleting, at the early stage of the dynamic programming, partial solutions that might lead to good final solutions. Obviously, the size of boxes depends on the precision ϵ. Smaller precisions result in smaller boxes. Moreover, different precisions may be assigned to the different state components. In fact, states that are less sensitive to errors (e.g., with a "flat" structure) could be assigned larger precision values. Furthermore, the form of the real number Δ given by Equations (2.7) is justified by two reasons. First, its value is very close to 1. Hence, two states in the same box are indeed very close to each other. Second, we know how the sequence $\left(1+\frac{x}{a}\right)^a$ behaves when a approaches infinity.

The following lemma states the relation of the state space partition to the concepts of $\epsilon-$dominance and quasi $\epsilon-$dominance.

Lemma 2.2 Let $k \in [\![1, \beta]\!]$. For any two states $S, S' \in \widetilde{\mathbf{S}}_\mathbf{k}$. If S and S' are in the same box, and S is the state with the smallest constrained state component, it holds that:

$$S \preceq_\Delta^{I_c} S' \quad \text{and} \quad S' \preceq_\Delta S$$

Proof: There exists some $r \in [\![1, L]\!]$ such that for all $i \in [\![1, \theta]\!]$, we have:

$$\Delta^{r-1} \leq s'_i \leq \Delta^r$$

We can also write that for all $i \in [\![1, \theta]\!]$, we have:

$$\frac{\Delta^r}{\Delta} \leq s'_i \leq \Delta^{r-1}\Delta$$

We know that S is in the same box as S'. Hence for all $i \in [\![1, \theta]\!]$, we have:

$$\Delta^{r-1} \leq s_i \leq \Delta^r$$

Consequently, for all $i \in [\![1, \theta]\!]$, we have;

$$\frac{s_i}{\Delta} \leq s'_i \leq \Delta s_i$$

Furthermore, we know that S is the state with smallest constrained state component. Hence $s_1 \leq s'_1$. This completes the proof of Lemma 2.2. □

The structure of DP^ϵ is summarized in Algorithm 2.

Algorithm 2 The dynamic programming DP^ϵ

1. Initialize $\widetilde{\mathbf{S}}_\mathbf{0}^\epsilon := \widetilde{\mathbf{S}}_\mathbf{0} := \mathbf{S_0}$
2. **for** all $1 \leq k \leq \beta$ **do**
3. $\quad \widetilde{\mathbf{S}}_\mathbf{k} := \emptyset$
4. **end for**
5. **for** $k = 1$ to β **do**
6. $\quad$ **for** all $\widetilde{\mathbf{S}}_\mathbf{k\text{-}1}^\epsilon$ **do**
7. $\quad\quad$ **for** all $T \in \mathcal{T}$ **do**
8. $\quad\quad\quad$ **for** $i = 1$ to N **do**
9. $\quad\quad\quad\quad$ **if** $F_T(Y_i, S) = 1$ **then**
10. $\quad\quad\quad\quad\quad \widetilde{\mathbf{S}}_\mathbf{k} := \widetilde{\mathbf{S}}_\mathbf{k} \cup T(Y_i, S)$
11. $\quad\quad\quad\quad$ **end if**
12. $\quad\quad\quad$ **end for**
13. $\quad\quad$ **end for**
14. $\quad$ **end for**
15. $\quad \widetilde{\mathbf{S}}_\mathbf{k} := \text{vmin}\left\{\widetilde{\mathbf{S}}_\mathbf{k}\right\}$
16. $\quad$ Compute $\widetilde{\mathbf{S}}_\mathbf{k}^\epsilon$ by trimming $\widetilde{\mathbf{S}}_\mathbf{k}$
17. **end for**
18. $\mathcal{PF}^\epsilon = \text{vmin}\left\{G(S) : S \in \widetilde{\mathbf{S}}_\mathbf{k}^\epsilon, 1 \leq k \leq \beta\right\}$

The trimmed set of states contains at most $(L+1)^\theta$ solutions. We compute the complexity of DP^ϵ as being proportional to

$$\begin{aligned} \sum_{k=1}^{\beta} \sum_{j=1}^{|\mathcal{T}|} \sum_{i=1}^{N} \left|\widetilde{\mathbf{S}}_\mathbf{k}^\epsilon\right| &= O\left(\beta|\mathcal{T}|N(L+1)^\theta\right) \\ &= O\left(\beta^{\theta+1}|\mathcal{T}|N\left(\frac{\ln b}{\epsilon}\right)^\theta\right) \end{aligned} \tag{2.8}$$

All input is encoded in binary, and θ and $|\mathcal{T}|$ depend only on MOSP and DP, and not on any specific instance of MOSP. Equation (2.8) shows that the running time of DP^ϵ is polynomial in $N, \beta, \ln b$ and $\frac{1}{\epsilon}$. Obviously, there is a trade-off between the value of ϵ and the running time of DP^ϵ. In fact, for small values of ϵ more solutions are kept during the execution of DP^ϵ as the boxes illustrated in Figure 2.3 are smaller. Therefore, more data is processed which results in an increase of the running time. Intuitively, the quality of the approximation depends on ϵ. In fact, because of the trimming action, DP^ϵ generates "incorrect data" as the new set of generated states deviates from the set of states that the MOSP would have reached through DP.

However, less data is trimmed for small values of ϵ which limits the error caused by the trimming action.

2.6.2 Worst case performance of DP^ϵ

In this section, we show that the worst case performance guarantee is such that every state generated by DP is at most a constant factor from a state generated by DP^ϵ. Now, we state the following important lemma:

Lemma 2.3 Let $k \in [\![0, \beta]\!]$. For all $S \in \mathbf{S_k}$, there exists $\widetilde{S} \in \widetilde{\mathbf{S}}^\epsilon_\mathbf{k}$ such that:

$$\widetilde{S} \preceq^{I_c}_{\Delta^k} S$$

Proof: To prove Lemma 2.3, we use induction on $k \in [\![0, \beta]\!]$.
$\widetilde{\mathbf{S}}^\epsilon_\mathbf{0} = \mathbf{S_0}$, hence Lemma 2.3 is true for $k = 0$. Let us assume Lemma 2.3 is true for $k-1$ and prove it for k.
Let $S \in \mathbf{S_k}$. Per definition of the set $\mathbf{S_k}$, S is feasible. Hence, there exists a feasible state $S' \in \mathbf{S_{k-1}}$, some vector Y_{i_k}, $i_k \in [\![1, N]\!]$, and some transition mapping $T \in \mathcal{T}$ such that $S = T(Y_{i_k}, S')$. On the other hand, because of the induction assumption, there exists $\widetilde{S}' \in \widetilde{\mathbf{S}}^\epsilon_\mathbf{k-1}$, such that:

$$\widetilde{S}' \preceq^{I_c}_{\Delta^{k-1}} S' \tag{2.9}$$

Furthermore, in the k^{th} iteration, DP^ϵ generates the state $T(Y_{i_k}, \widetilde{S}')$. The feasibility of the state $T(Y_{i_k}, \widetilde{S}')$ is guaranteed thanks to Assumption 2.3 and Equation 2.9. $T(Y_{i_k}, \widetilde{S}')$ might be removed after trimming. However, some state $\widetilde{S}$, located in the same box, should be left.
From Lemma 2.2, we know that:

$$\widetilde{S} \preceq^{I_c}_{\Delta} T(Y_{i_k}, \widetilde{S}') \tag{2.10}$$

Due to Assumption 2.2 and Equations 2.10, we have:

$$\widetilde{S} \preceq^{I_c}_{\Delta^k} S$$

This completes the proof of Lemma 2.3. □

Lemma 2.3 implies that a guarantee on the quality of the approximation is proved. In the following theorem, we further demonstrate that the performance guarantee depends on the precision ϵ and is preserved by the approximate Pareto fronts.

Theorem 2.1 For every $X \in \mathcal{PF}^*$, there exists $\widetilde{X} \in \mathcal{PF}^\epsilon$ such that

$$\widetilde{X} \preceq_{1+\epsilon} X$$

Proof: Let $X \in \mathcal{PF}^*$ be a Pareto solution. For some $k \in [\![1, \beta]\!]$ there exists a state $S \in \mathbf{S_k}$ such that $X = G(S)$. According to Lemma 2.3, there exists $S' \in \widetilde{\mathbf{S}}^\epsilon_{\mathbf{k}}$ such that:

$$\widetilde{S} \preceq^{I_c}_{\Delta^k} S$$

Hence, the following also holds:

$$\widetilde{S} \preceq^{I_c}_{\Delta^\beta} S$$

The sequence $\left(1 + \frac{\epsilon}{2\beta}\right)^\beta$ is increasing in β and converges to $e^{\frac{\epsilon}{2}}$. Hence:

$$\left(1 + \frac{\epsilon}{2\beta}\right)^\beta \leq e^{\frac{\epsilon}{2}}$$

Furthermore, for every real number $0 < x < 1$ we have:

$$e^{\frac{x}{2}} \leq 1 + x$$

Therefore,

$$\widetilde{S} \preceq^{I_c}_{1+\epsilon} S \tag{2.11}$$

Hence, due to Lemma 2.1, we have:

$$G(\widetilde{S}) \preceq_{1+\epsilon} G(S) \tag{2.12}$$

As $X = G(S)$, we have:

$$G(\widetilde{S}) \preceq_{1+\epsilon} X \tag{2.13}$$

If $G(\widetilde{S}) \in \mathcal{PF}^\epsilon$, then we are done. We just take $\widetilde{X} = G(\widetilde{S})$. If $G(\widetilde{S})$ is not in $\mathcal{PF}^\epsilon$, then $G(\widetilde{S})$ is dominated by some vector in $\mathcal{PF}^\epsilon$. We take $\widetilde{X}$ to be that vector.
This completes the proof of Theorem 2.1. □

Theorem 2.1 also implies that the approximate Pareto front covers well the real Pareto front. In fact, every Pareto solution is closely approximated by at least one solution from the approximate Pareto front. Furthermore, when β does not depend on the numerical value of the input, DP^ϵ belongs to the family of FPTAS algorithms.

2.7. Conclusions

Multi-objective optimization problems are very challenging. They are at least as complex as their single-objective version. Most existing algorithms fail to perform well in terms of both computation times and solutions quality. While exact algorithms only deal with small problems, heuristics produce weak Pareto fronts that badly cover the real Pareto fronts. In this chapter, we propose a generic and flexible framework to deal with multi-objective scheduling problems. An efficient approximation based on dynamic programming is developed that generates good quality approximate Pareto fronts. Although they contain fewer solutions, the approximate Pareto fronts cover well the real Pareto fronts. The quality of the solutions can be decided on as the precision on each objective is an input to the algorithm and can be tuned by the decision maker. However, small precisions require more computation time. Moreover, for each objective a different precision can be set. Therefore, larger errors might be allowed for less sensitive objectives (e.g., with a flat cost structure). In Chapter 3, the approximation is tested on the multi-objective knapsack problem with time-dependent cost parameters.

Chapter 3

The Time-Dependent Multi-Objective Knapsack Problem

3.1. Introduction

The knapsack problem (KP) is a classical $\mathcal{NP}-$hard problem in combinatorial optimization. The KP is well studied in the literature due to its wide range of applications such as cargo loading, cutting stock, project selection and budget control (Kellerer et al., 2005). Furthermore, the KP is an interesting problem from a theoretical point of view. In fact, in many cases, the KP appears as a subproblem for more complicated combinatorial optimization problems such as the vehicle routing problem (Righini and Salani, 2006). In the KP, the input consists of a knapsack with a finite capacity and a set of items, each with a certain weight and a profit. A feasible solution to the KP is a selection of items such that their total weight does not exceed the knapsack capacity. In most of the KP literature, the aim is to maximize (or minimize) the single objective function consisting of the total profit (or cost) of the selected items. Profits (or costs) are assumed to be constant over time.

Many optimization problems encountered in practice are multi-objective in nature, i.e., different objectives are conflicting and equally important. Furthermore, in many practical settings cost/profit parameters change over time. For instance, in the retail environment, profits are dependent on the time of the year (e.g., whether it is Christmas or not). Dock planning is a very important warehousing activity.

Usually, the number of persons available on a dock to load/unload trucks varies during the day resulting in a time-dependent loading/unloading speed. Obviously, in practical situations costs are hardly constant over time. A particular multi-objective optimization problem is the Multi-Objective Knapsack Problem (MOKP). The MOKP is a generalization of the KP in which items have multiple costs. The MOKP is less studied in the literature compared to the KP, and is solved by means of exact methods, approximation algorithms and heuristic approaches. Considering time-dependent costs results in the *Time-Dependent Multi-Objective Knapsack Problem* (TDMOKP). Research dealing with the TDMOKP is limited. In fact, only small instances of this problem are solved using dynamic programming, and no approximations or heuristics are known for it. In this chapter, we propose an efficient approximation algorithm for the TDMOKP based om the trimming method described in Chapter 2.

The main contributions of this chapter are summarized as follows. First, we validate the methodology derived in Chapter 2. An approximation algorithm based on dynamic programming is proposed for the TDMOKP. The approximation generates approximate Pareto fronts with fewer solutions. Nonetheless, the approximate Pareto fronts provide a very good coverage of the true Pareto fronts. Additionally, the worst case performance of the approximation is guaranteed. The decision maker has the flexibility to tune the precision level for the different objectives as he might be willing to tolerate more error for less sensitive objectives. In our numerical experiments, we consider a multi-objective function with four objectives. Secondly, we capture the dynamic nature of real-life optimization problems by assuming time-dependent costs. Adding time-dependency makes the KP harder as, opposed to the time-independent version, the sequence in which items are put in the knapsack has an impact on the value of the objective function.

This chapter is organized as follows. Section 3.2 reviews the literature relevant to the knapsack problem. Section 3.3 is devoted to the formal description of the problem at hand and its dynamic programming formulation. In Section 3.4, an approximation of the Pareto front is developed and the main results of the chapter are derived. In Section 3.5, a numerical study is conducted. Finally, Section 3.6 concludes with a summary of the main results.

3.2. Literature Review

Since the pioneer work of Dantzig (1957), the KP has been widely studied in the literature due to its theoretical importance and practical relevance. The many variants of the KP such as the bounded KP (Pisinger, 2000), the unbounded KP (Kellerer et al.,

2005), the multi-dimensional KP (Freville, 2004), the multiple-choice KP (Pisinger, 1997) and the quadratic KP (Pisinger, 2007) are mostly addressed as single-objective and time-independent optimization problems (see also Martello and Toth (1990)). In the literature, several efficient exact methods, approximation algorithms and heuristics have been successfully developed to solve large scale KPs. Exact methods include branch-and-bound (Horowitz and Sahni, 1974; Fayard and Plateau, 1975; Nauss, 1976; Martello and Toth, 1988), dynamic programming (Toth, 1980; Pisinger, 1997), core concept (Balas and Zemel, 1980), and hybrid approaches (Martello et al., 1999). Approximation algorithms consist mainly of polynomial time approximation schemes (PTAS) (Sahni, 1975), and fully polynomial time approximation schemes (FPTAS) (Ibarra and Kim, 1975; Magazine and Oguz, 1981).

The MOKP is less studied in the literature compared to the single-objective KP, and most of its solution methods are inspired from these proposed for the single-objective KP. In the literature, the MOKP is solved by means of exact methods including branch-and-bound procedures (Visee et al., 1998), dynamic programming (Bazgan et al., 2009b; Klamroth and Wiecek, 2000a), core concept (Da Silva et al., 2008), and labeling algorithms (Captivo et al., 2003). However, to solve reasonably large instances, approximation algorithms and heuristics are proposed. In Erlebach et al. (2002), an FPTAS and a PTAS are proposed and their existence is proven for the one-dimensional and the multi-dimensional MOKP respectively; yet the algorithms are not implemented and there performance is not evaluated. In Bazgan et al. (2009a), an efficient FPTAS is designed and its performance is evaluated on a bi- and tri-objective KP. Furthermore, meta-heuristic methods are developed to deal with large scale MOKPs. These include evolutionary algorithms (Zitzler and Thiele, 1999), tabu search (Hansen, 1997), simulated annealing (Ulungu et al., 1999), hybrid meta-heuristics (Ben Abdelaziz et al., 1999) and scatter search (Erlebach et al., 2002). In the literature, almost all experimental results are derived based on a bi-criteria objective function (Captivo et al., 2003; Da Silva et al., 2008; Visee et al., 1998). In Bazgan et al. (2009b), numerical experiments are also performed for a tri-criteria objective function. In this chapter, the numerical experiments are conducted on a four-criteria objective function; yet the methodology is valid, as proved in Chapter 2, for any number of objectives.

An extension of the MOKP, is the time-dependent multi-objective knapsack problem in which items have time-dependent costs (i.e., costs depend on the time items are put in the knapsack). Research on the TDMOKP is scarce. The only two papers we are aware of are by Klamroth and Wiecek (2000b), and Klamroth and Wiecek (2001). In Klamroth and Wiecek (2000b), the capital budgeting problem is considered where a subset of projects needs to be selected from a possible set of projects according to multiple objectives, and when a budget restriction is imposed. In Klamroth and

Wiecek (2001), the single-machine scheduling problem is dealt with where items production times are time-dependent. In both papers, dynamic programming is used to solve very small instances. As a matter of fact, no approximation algorithms exist in the literature for the TDMOKP.

3.3. Problem Description

Throughout this chapter, we address the Time-Dependent Multi-Objective Knapsack Problem (TDMOKP). Without loss of generality, we consider the case when the TDMOKP involves the joint minimization of a fixed number $n \geq 2$ of objectives.

3.3.1 The input

An instance of the TDMOKP consists of N items $I_1, \ldots, I_N$, and a knapsack with a finite capacity $\beta \in \mathbb{N}$. Each item I_k is associated with a cost vector, a weight and a duration. In line with the multi-objective nature of the TDMOKP, a cost vector contains n components (i.e., each cost component corresponds to a specific objective). The goal is to determine all possible subsets of items that when put in the knapsack, the corresponding total weight does not exceed the knapsack capacity, and the corresponding total cost vector is non-dominated (i.e., determine the Pareto front $\mathcal{PF}^*$). The actual cost vector incurred by an item and its duration depend on the time, $s \in \mathbb{R}$, the item is added to the knapsack. Therefore, the sequence in which items are put in the knapsack has an impact on the value of the solution. An item can be put zero or several times in the knapsack, and all items are available for planning at time 0.
Let $\mathcal{C}$ be a set of mappings from $\mathbb{R}$ to $\mathbb{R}$. To each item I_k, we associate the input vector $Y_k = [w_k, p_k(s), c_{1,k}(s), \ldots, c_{n,k}(s)] \in \mathcal{C}^{n+2}$. $p_k(s)$ is a mapping from $\mathbb{R}$ to $\mathbb{R}$ that, given the time s, returns the duration of item I_k. For all $r \in [\![1, n]\!]$, $c_{r,k}(s)$ is a mapping from $\mathbb{R}$ to $\mathbb{R}$ that computes the time-dependent cost component corresponding to the r^{th} objective. $w_k \in \mathbb{N}$ is a positive number representing the weight of item I_k (i.e., we restrict ourself to the case of time-independent integer weights). All input is encoded in binary.
Items can, for example, be projects to be executed. Each project incurs an investment (i.e., its weight) and generates a profit that depends on its execution time. Moreover, a project has a start and end time (i.e., a duration). A project can be executed zero or several times, depending on its contribution to the value of the objective function. In Klamroth and Wiecek (2001), exactly the same TDMOKP is studied.

3.3.2 Dynamic programming for the TDMOKP

The dynamic programming for the TDMOKP goes through β iterations. In iteration k, the item $I_{i_k}, 1 \leq i_k \leq N$, (not necessarily I_k) is added to the knapsack. In other words, the input piece Y_{i_k} is processed and a set of states $\mathbf{S_k}$ is generated. The state space $\mathbf{S_k}$ is a subset of $\mathbb{R}^{n+2}$, such that any state $S \in \mathbf{S_k}$ consists of the components $[s_1, s_2, \ldots, s_{n+1}, s_{n+2}]$. The first state component represents the total weight of the partial solution encoded by the state S. The second component stands for the total duration. The remaining n state components represent the total selected cost with regard to the n objectives. Every state S encodes a partial solution defined by the sequence $I_{i_1}, \ldots, I_{i_k}$ of items. Note that the sequence is crucial as a different permutation of the same set of items may result in a different state.

Let T be a mapping from $\mathcal{C}^{n+2} \times \mathbb{R}^{n+2}$ to $\mathbb{R}^{n+2}$. To T, we associate a mapping F from $\mathcal{C}^{n+2} \times \mathbb{R}^{n+2}$ to $\{0, 1\}$. we call T the transition mapping and F the feasibility mapping. In fact, in iteration k of the dynamic programming, for any state S in the old state space $\mathbf{S_{k-1}}$ and any input vector Y_{i_k}, the new generated state $T(Y_{i_k}, S)$ is feasible if and only if $F(Y_{i_k}, S) = 1$. The new state $T(Y_{i_k}, S)$ is added to the state space $\mathbf{S_k}$ only if it is feasible. The new state $T(Y_{i_k}, S)$ is determined as:

$$T(Y_{i_k}, S) = [s_1 + w_{i_k}, s_2 + p_{i_k}(s_2), s_3 + c_{1,i_k}(s_2), \ldots, s_{n+2} + c_{n,i_k}(s_2)]$$

the feasibility mapping is defined as follows:

$$F(Y_{i_k}, S) = \begin{cases} 1 & \text{if } \; s_1 + w_{i_k} \leq \beta \\ 0 & \text{otherwise} \end{cases}$$

The value of a state is evaluated by means of the mapping G from $\mathbb{R}^{n+2}$ to $\mathbb{R}^n$ such that for any state $S \in \mathbb{R}^{n+2}$, the image of S in the objective space is the $n-$dimensional vector

$$G(S) = [s_3, \ldots, s_{n+2}]$$

Similarly to Klamroth and Wiecek (2001), we assume the following structure on the vector of cost functions and the duration function:

Assumption 3.1 For any item I_i, and for any times $s, s' \in \mathbb{R}$ such that $s \leq s'$ it holds that:

1. $s + p_i(s) \leq s' + p_i(s')$

2. $c_{r,i}(s) \leq c_{r,i}(s')$, for all $r \in [\![1, n]\!]$

Condition 1 of Assumption 3.1 implies that the duration of any item adheres to the well-known FIFO principle, meaning that overtaking is not allowed. The FIFO assumption is realistic from a practical point of view. In fact, in many production and transportation processes, cost parameters satisfy the FIFO principle (e.g., a job's processing time on a machine, vehicle's travel time etc). Condition 2 implies that cost components are non-decreasing in time. Although such cost structure is not always true, it captures many practical situations. For example, the selling price of perishable products decreases in time as their quality deteriorates when time elapses. It is easy to check, that given Assumption 3.1, Assumption 2.1 (Chapter 2, Page 23) is satisfied. Similarly to Klamroth and Wiecek (2001), we state the non-dominance principle for the TDMOKP in the following lemma.

Lemma 3.1 (principle of non-dominance) Let S be a non-dominated state in $\mathbf{S_k}$. There exists a non-dominated state $S' \in \mathbf{S_{k-1}}$ and an input vector Y_{i_k} such that $S = T(Y_{i_k}, S')$

Proof: Follows directly from Lemma 2.1 of Chapter 2. □

We let $\underline{0}$ be the zero vector of $\mathbb{R}^{n+2}$. The iterative structure of DP is summarized in Algorithm 3.

Algorithm 3 Dynamic programming for the TDMOKP

1. Initialize $\mathbf{S_0} := \{\underline{0}\}$
2. **for** all $1 \leq k \leq \beta$ **do**
3. $\quad \mathbf{S_k} := \emptyset$
4. **end for**
5. **for** $k = 1$ to β **do**
6. $\quad$ **for** all $S \in \mathbf{S_{k-1}}$ **do**
7. $\quad\quad$ **for** $i = 1$ to N **do**
8. $\quad\quad\quad$ **if** $s_1 + w_i \leq \beta$ **then**
9. $\quad\quad\quad\quad \mathbf{S_k} := \mathbf{S_k} \cup T(Y_i, S)$
10. $\quad\quad\quad$ **end if**
11. $\quad\quad$ **end for**
12. $\quad$ **end for**
13. $\quad \mathbf{S_k} := \text{vmin}\{\mathbf{S_k}\}$
14. **end for**
15. $\mathcal{PF}^* = \text{vmin}\{G(S) : S \in \mathbf{S_k}, 1 \leq k \leq \beta\}$

The size of the state space $\mathbf{S_k}$ is usually very large. Consequently, the running time of DP is large too. In the next section, DP is approximated in order to reduce its complexity.

3.4. Approximating the TDMOKP

In this section, we approximate the dynamic programming presented in the previous section by applying the trimming method introduced in Chapter 2. We impose extra structure on the execution of the DP algorithm. In the k^{th} iteration, we calculate the state space $\widetilde{\mathbf{S}}_\mathbf{k}$ from the state space $\widetilde{\mathbf{S}}^\epsilon_{\mathbf{k-1}}$ (the trimmed copy of the state space $\widetilde{\mathbf{S}}_{\mathbf{k-1}}$). Trimming the state space in DP results in an approximate dynamic programming DP^ϵ with the precision ϵ. The goal is to generate an approximate Pareto front denoted as $\mathcal{PF}^\epsilon$. The iterative recursion of DP^ϵ recursion is formulated as follows:

Algorithm 4 Approximate dynamic programming for the TDMOKP

1. Initialize $\widetilde{\mathbf{S}}^\epsilon_\mathbf{0} := \widetilde{\mathbf{S}}_\mathbf{0} := \mathbf{S_0}$
2. **for** all $1 \leq k \leq \beta$ **do**
3. $\quad \widetilde{\mathbf{S}}_\mathbf{k} := \emptyset$
4. **end for**
5. **for** $k = 1$ to β **do**
6. $\quad$ **for** all $\widetilde{\mathbf{S}}^\epsilon_{\mathbf{k-1}}$ **do**
7. $\quad\quad$ **for** $i = 1$ to N **do**
8. $\quad\quad\quad$ **if** $s_1 + w_i \leq \beta$ **then**
9. $\quad\quad\quad\quad \widetilde{\mathbf{S}}_\mathbf{k} := \widetilde{\mathbf{S}}_\mathbf{k} \cup T(Y_i, S)$
10. $\quad\quad\quad$ **end if**
11. $\quad\quad$ **end for**
12. $\quad$ **end for**
13. $\quad \widetilde{\mathbf{S}}_\mathbf{k} := \text{vmin}\left\{\widetilde{\mathbf{S}}_\mathbf{k}\right\}$
14. $\quad$ Compute $\widetilde{\mathbf{S}}^\epsilon_\mathbf{k}$ by trimming $\widetilde{\mathbf{S}}_\mathbf{k}$
15. **end for**
16. $\mathcal{PF}^\epsilon = \text{vmin}\left\{G(S) : S \in \widetilde{\mathbf{S}}^\epsilon_\mathbf{k}, 1 \leq k \leq \beta\right\}$

Let $b \in \mathbb{R}$ be an upper bound on the numerical values of all the state components. We, for example, set:

$$b = \max\left\{\beta, \left\lceil \frac{\beta}{\min\limits_{i\in[\![1,N]\!]} w_i} \right\rceil \max_{i\in[\![1,N]\!], s\in T_p} p_i(s), \max_{r\in[\![1,n]\!]} \left\{ \left\lceil \frac{\beta}{\min\limits_{i\in[\![1,N]\!]} w_i} \right\rceil \max_{i\in[\![1,N]\!], s\in T_r} c_i^r(s) \right\} \right\}$$

where T_p is the domain definition of the duration function $p_i(s)$, and T_r is the domain definition of the cost function $c_{r,i}(s)$. $\left\lceil \frac{\beta}{\min\limits_{i \in [\![1,N]\!]} w_i} \right\rceil$ is an upper bound on the number of items that can be put in the knapsack.

Formally, the set $\widetilde{\mathbf{S}}_\mathbf{k}$ is represented by geometric points in the polyhedron $[0,b]^{n+2}$. The polyhedron is cut into multiple boxes of exponentially increasing size. States contained by the same box define a cluster of states that are very close to each other. In each box, only the state with the smallest first state component (i.e., the state component representing the total weight) is retained. The cuts are executed at the coordinates $\Delta^m, m \in [\![1, L]\!]$, where

$$\Delta = 1 + \frac{\epsilon}{2\beta} \tag{3.1}$$

ϵ is a real number between 0 and 1 representing the precision, and the value of L is chosen such that $\Delta^L \leq b$. We set

$$L = \left\lceil \frac{\ln b}{\ln \Delta} \right\rceil \leq \left\lceil \left(1 + \frac{2\beta}{\epsilon}\right) \ln b \right\rceil \tag{3.2}$$

The trimmed state space contains at most $(L+1)^{n+2}$ states. We can compute the complexity of DP^ϵ as being proportional to:

$$\begin{aligned} \sum_{k=1}^{\beta} \sum_{i=1}^{N} \left| \widetilde{\mathbf{S}}_\mathbf{k}^\epsilon \right| &= O\left(\beta N (L+1)^{n+2}\right) \\ &= O\left(\beta^{n+3} N \left(\frac{\ln b}{\epsilon}\right)^{n+2}\right) \end{aligned} \tag{3.3}$$

From Equation 3.3, we conclude that the running time of DP^ϵ is polynomial in $N, \beta, \ln b$ and in $\frac{1}{\epsilon}$. β (the knapsack capacity) depends on the numerical value of the input. Hence, DP^ϵ runs in time polynomial in the input size (i.e., pseudo-polynomial time). The number of objectives n is fixed and does not depend on any specific instance of the TDMOKP.

3.4.1 Worst case performance guarantee of DP^ϵ:

We show that the worst case performance guarantee is such that every solution generated by DP is at most, in all objectives, a constant factor from that of a DP^ϵ

solution. Henceforth, we assume the following structure regarding the cost vector and the duration:

Assumption 3.2 For every item I_i, for any real number $\alpha \geq 1$, and for any two times $s, s' \in \mathbb{R}$ such that $s \leq \alpha s'$, it holds that for all $r \in [\![1, n]\!]$:

$$p_i(s) \leq \alpha p_i(s') \quad \text{and} \quad c_{r,i}(s) \leq \alpha c_{r,i}(s')$$

Assumption 3.2 implies that if an item is added to the knapsack at a later time $\alpha s'$ instead of time s, its cost components as well as its duration do not explode by more than the coefficient α. In other words, sudden increases in the costs and in the duration due to deviations in the start times of items are not allowed, which is reasonable in many real-life situations. For instance, the costs functions $c_{r,i}(s) = s$, $c_{r,i}(s) = \sqrt{s}$ and $c_{r,i}(s) = \ln s$ satisfy Assumption 3.2. Given Assumption 3.2, it is easy to check that Assumption 2.2 (Chapter 2, Page 26) is satisfied. Furthermore, for any two states $S, S' \in \mathbb{R}^{n+2}$ such that $s_1 \leq s'_1$, and for any input vector $Y_i \in \mathcal{C}^{n+2}$, it holds that $s_1 + w_i \leq s'_1 + w_i$. Hence, Assumption 2.3 (Chapter 2, Page 26) is satisfied. The following theorem follows directly from Lemma 2.3 and Theorem 2.1 of Chapter 2.

Theorem 3.1 For the TDMOKP, for any $y \in \mathcal{PF}^*$ there exists $\tilde{y} \in \mathcal{PF}^\epsilon$ such that for all $r \in [\![1, n]\!]$:

$$\tilde{y}_r \leq (1 + \epsilon) y_r$$

In the next section, numerical experiments are run for the TDMOKP with four objectives.

3.5. Computational Results

We consider a multi-objective cost function in which the joint optimization of four objectives is carried out. The first objective represents the total duration of the selected items, and must be minimized. The other objectives (2,3 and 4) are arbitrary, and need to be maximized. For an item I_i, the time-dependent cost parameters are such that:

$$c_i(s) = \begin{bmatrix} c_{1,i}(s) = p_i(s) \\ c_{2,i}(s) = U_2 - V_2\sqrt{s} - W_2 s \\ c_{3,i}(s) = U_3 - V_3\sqrt{s} - W_3 s \\ c_{4,i}(s) = U_4 - V_4\sqrt{s} - W_4 s \end{bmatrix} \tag{3.4}$$

where s is the start time for processing item I_i, and $p_i(s)$ is its duration. $p_i(s)$ is obtained by dividing the planning horizon into 5 zones such that in each zone an item is processed using a different pace. Furthermore, we consider three type of items: fast (with a short duration), normal (with a moderate duration) and slow (with a long duration). The items type is chosen randomly and remains the same for all instances. For $r \in \{2, 3, 4\}$, the constants U_r, V_r and W_r are randomly generated in the intervals [500, 1000], [0, 0.2] and [0, 0.05] respectively. In total, we generated 20 instances for which we use the notation $ITm.\beta$, where T is the instance type which is either 1 (weight of the items is randomly generated in the range [1,10]) or 0 (weight of the items is randomly generated in the range [10,20]); m is the instance number and β is the knapsack capacity. In our experiments we choose $\beta \in \{50, 150\}$. Instances of type 1 with $\beta = 150$ are the most difficult instances because the number of items that can be put in the knapsack is large. The algorithms DP and DP^{ϵ} are implemented on an Intel(R) Core(TM)2 CPU, 2.13 GHz, 3 GB of RAM computer, in a Matlab R2010b environment. Furthermore, we set 24 hours as the limit for computation times.

	Exact		ϵ=0.05		ϵ=0.1		ϵ=0.3	
Instances	Time(s)	Size	Time(s)	Size	Time(s)	Size	Time(s)	Size
I01.50	-	-	1279.54	2868	152.29	1441	13.31	400
I02.50	-	-	157.67	1903	42.04	1075	5.77	368
I03.50	-	-	351.36	1137	74.09	584	7.46	213
I04.50	-	-	94.61	778	26.88	496	4.50	226
I05.50	-	-	66.58	1335	20.89	728	4.43	376
I01.150	-	-	-	-	7122.28	8470	199.73	2084
I02.150	-	-	4005.17	11028	638.12	3984	62.47	1225
I03.150	-	-	10624.29	4664	1596.27	2433	106.47	843
I04.150	-	-	3411.90	7854	470.84	2607	46.17	713
I05.150	-	-	1781.76	4534	305.79	2367	41	997
I11.50	3.38	562	2.23	132	1.37	95	0.41	48
I12.50	1.93	384	1.12	88	0.74	67	0.35	35
I13.50	20.39	1342	6.80	256	3.78	231	1.16	92
I14.50	7.41	730	2.39	173	1.31	131	0.40	62
I15.50	5.68	809	1.93	138	1.07	118	0.42	64
I11.150	-	-	1955.10	1331	88.36	256	7.76	87
I12.150	-	-	1776.20	2200	37.47	313	6.33	155
I13.150	-	-	2214,6	2147	358.31	823	23	298
I14.150	-	-	407	1341	89.59	652	9.48	258
I15.150	-	-	676.84	1384	142.27	723	12.75	253

Table 3.1 Instances with 25 items in case of 4 objectives.

	Exact		ϵ=0.05		ϵ=0.1		ϵ=0.3	
Instances	Time(s)	Size	Time(s)	Size	Time(s)	Size	Time(s)	Size
I01.50	-	-	1896.63	2398	333.23	1311	28.01	411
I02.50	-	-	635.19	3432	131.71	1545	15.24	422
I03.50	-	-	3174.23	2070	617.30	1054	48.93	301
I04.50	-	-	1392.35	1784	229.77	883	20.86	296
I05.50	-	-	497.85	1715	109.23	718	12.12	333
I01.150	-	-	61693.73	15768	7532.06	6044	405.31	1565
I02.150	-	-	14154.03	15219	2408.69	7092	180.62	1917
I03.150	-	-	-	-	11943.75	3416	720.24	1052
I04.150	-	-	44689.53	12242	5303	3920	278.42	1051
I05.150	-	-	17431.76	8101	2081.78	3667	147.61	936
I11.50	86.64	2337	17.88	282	6.44	171	2.06	59
I12.50	9.98	634	4.29	146	2.64	101	1.07	51
I13.50	261.72	4972	44.68	486	15	324	2.22	74
I14.50	43.76	1538	12.27	215	5.01	131	1.20	58
I15.50	25.36	1138	7.48	154	3.66	120	1.12	62
I11.150	-	-	2127.23	1331	351.16	573	26.81	208
I12.150	-	-	275.47	839	92.45	514	15.59	186
I13.150	-	-	9854.94	1563	1209.24	409	56.57	172
I14.150	-	-	755.27	635	180.07	389	21.19	173
I15.150	-	-	730.24	900	178.51	558	21.42	240

Table 3.2 Instances with 50 items in case of 4 objectives.

3.5.1 Pareto front vs. approximate Pareto fronts

As an illustration, Figure 3.1 depicts the different (approximate) Pareto fronts corresponding to the instance $I02.150$ with 50 items. For the sake of a clear presentation, the comparison is carried out in case of only two objectives (the total duration and the second objective). As expected, smaller values of the worst case precision ϵ result in an approximate Pareto front with both a better quality and coverage of the Pareto front. Furthermore, the deviation of an approximate Pareto front from the Pareto front increases in later iterations of the dynamic programming (remember that the size of boxes in Figure 2.3 (Chapter 2, Page 27) increases as we move farther from the origin). However, the deviation stays within the worst case precision ϵ.

3.5.2 Impact of the precision ϵ for DP^{ϵ}

Tables 3.1, 3.2 and 3.3 show the impact of the precision ϵ on the computation time of DP^{ϵ} (columns denoted by "Time(s)"), and on the number of points of the approximate Pareto fronts (columns denoted by "Size"). Clearly, considerable savings in computation times can be obtained by using DP^{ϵ}. Furthermore, the number of

	Exact		ϵ=0.05		ϵ=0.1		ϵ=0.3	
Instances	Time(s)	Size	Time(s)	Size	Time(s)	Size	Time(s)	Size
I01.50	-	-	3162.46	3362	622.11	1663	54.58	505
I02.50	-	-	2683.23	1567	442.60	850	43.40	320
I03.50	-	-	16301.60	3133	1882.03	1454	126.68	517
I04.50	-	-	10740.19	7338	1940.79	3399	129.46	968
I05.50	-	-	7437.86	4391	1456.14	2084	101.91	639
I01.150	-	-	-	-	13864.67	7033	760.64	1858
I02.150	-	-	73063.91	6248	9047.92	3045	486.26	999
I03.150	-	-	-	-	61554.45	5208	1830.81	1731
I04.150	-	-	-	-	42457.98	13345	1970.85	3950
I05.150	-	-	-	-	25335.80	10237	1453.76	2451
I11.50	857.84	7244	87.46	566	27.89	330	4.14	94
I12.50	87	2123	24.52	374	11.18	225	2.74	76
I13.50	1999.08	10835	187.25	655	50.48	292	5.38	93
I14.50	354.30	4651	48.02	420	15.69	234	2.69	89
I15.50	159.78	2231	20.94	217	8.13	164	2.25	74
I11.150	-	-	17843.35	3679	2121.10	1593	102.40	449
I12.150	-	-	2136.82	2200	446.74	1131	48.60	332
I13.150	-	-	38451.28	5065	4775.61	1974	161.20	443
I14.150	-	-	3961.95	1633	755.41	850	58.08	287
I15.150	-	-	1314.48	1713	331.12	1076	40.89	367

Table 3.3 Instances with 100 items in case of 4 objectives.

solutions in the approximate Pareto fronts decreases considerably compared to the true Pareto front. The savings in computation times are more significant for small precisions. The exact DP can only solve easy instances (i.e., instances with a small knapsack capacity and large item weights). $DP^{0.05}$ solve most of the instances, except some of the difficult instances (i.e., instances with a large knapsack capacity and small item weights). Finally, $DP^{0.1}$ is able to solve all instances.

3.5.3 Impact of the number of items N and the knapsack capacity β

Computation time increases with the number of items and the knapsack capacity. However, the impact of the knapsack capacity is more significant, which complies with Equation (3.3). Figure 3.2 illustrates the total computation time (CPU) over all instances as a function of the number of items, for both $\beta = 50$ and $\beta = 150$, and in case $\epsilon = 0.1$.

3.6. Conclusions

In this chapter, we validate the methodology presented in Chapter 2 by considering the time-dependent multi-objective knapsack problem where the state of the system is expressed as a function of time. In fact, to reflect the dynamic nature of real-life situations, cost parameters are considered to be time-dependent. Reasonably large instances with 100 items and a knapsack capacity of 150 are solved using the approximation in case of four criteria. Considerable gains in computation time are achieved by applying the approximation. Furthermore, the size of the generated approximate Pareto fronts can be considerably reduced. Although they consist of fewer solutions, the approximate Pareto fronts cover well the true Pareto front.

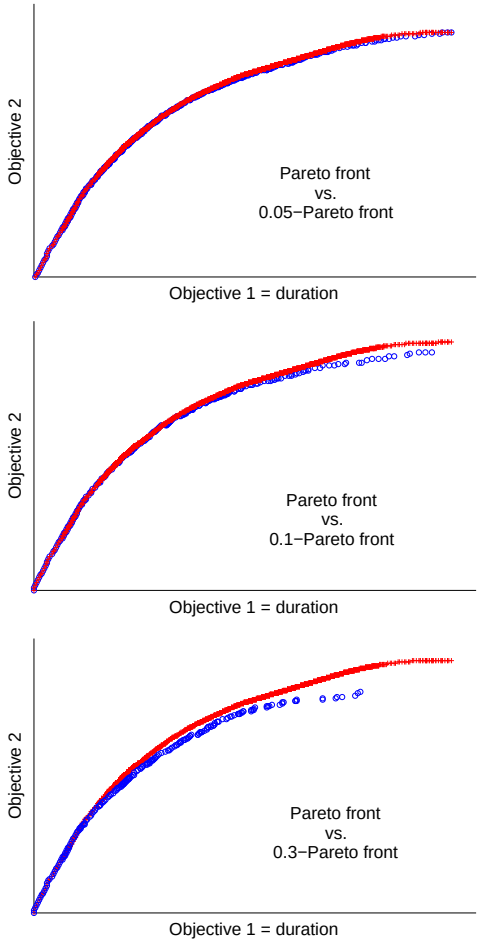

Figure 3.1 Pareto front vs. ϵ-Pareto fronts in case of 2 objectives.

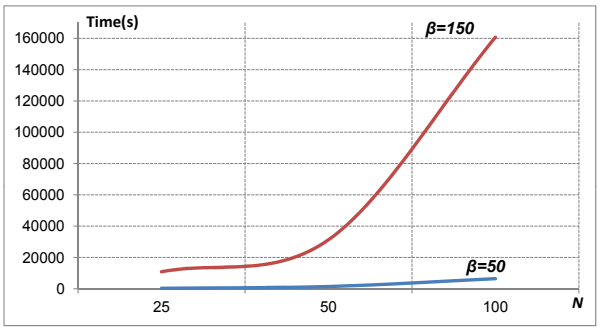

Figure 3.2 CPU as a function of β and N.

Chapter 4

The Time-Dependent Multi-Objective SVRPTW

4.1. Introduction

Consider the situation in which a vehicle is required to fill up ATMs located at different places from a central bank. For security reasons, it is not allowed to carry a large amount of money. Consequently, the vehicle is forced to make several short tours during its operating period (e.g., a working day) going back and forth to the central bank. Similarly, in the case of food home delivery, tours are relatively short as products are perishable (e.g., should remain warm) and need to be delivered as soon as possible to their final destinations (Azi et al., 2007). Clearly, a vehicle can make several tours during its designated operating time, respecting the vehicle's capacity for each tour. A 3PL company typically aims at scheduling its fleet such that a vehicle's total travel time is minimized, its capacity utilization is maximized and all customers are delivered in their specific time windows. This illustrates the importance to consider multiple dimensions in the objective function.

Because of road congestion, vehicles travel time in a traffic network is dependent on their departure time. In order to capture road congestion, we divide the planning horizon into time zones (e.g., morning, afternoon etc) where a different speed is associated with each time zone. The resulting stepwise speed function is translated into travel time functions that satisfy the FIFO principle (see also Ichoua et al. (2003)).

Formally, we consider a *Single Vehicle Routing Problem with Time Windows (SVRPTW)*. A single vehicle with finite capacity performs several routes to visit

a set of geographically dispersed customers with predefined demands and delivery time windows. In each route, the vehicle starts and ends at the same depot, and its capacity is respected. The vehicle is not allowed to arrive after the end of a customer's time window. Arrival before the start of the customer's time window is allowed. In this case, a waiting time is incurred. The vehicle serves the customers according to a predefined sequence. Therefore, the key decision to be made is whether to serve the next customer or return to the depot for stock replenishment. Because of road congestion, the vehicle's travel time is time-dependent. Furthermore, customers demand is non-increasing in time. For example, in case of perishable products, the quality deteriorates as time elapses. Consequently, customers may want less if served later. Choosing the right time to return to the depot is crucial as road congestion can be avoided. Moreover, the triangle inequality is not necessarily satisfied for time-dependent travel times. Consequently, traveling to the next customer via the depot might be faster. However, returning to the depot might also result in a later arrival of the vehicle at the next customers in the sequence. Consequently, due to time windows, the vehicle might not be able to serve some customers. Additionally, opportunities to sell more products are lost as customers' demand decreases with time. The vehicle routing problem at hand is capacitated on two dimensions: the vehicle capacity (truckload) and the time capacity. Time capacity means that the vehicle is only available for a limited amount of time, e.g., due to drivers' availability following the working regulations. Moreover, a tour is not allowed to last more than a certain amount of time. Because of these vehicle and time capacity limitations, it might not be possible to serve all customers. Therefore, the vehicle might skip some customers in the sequence and serve only the ones with the highest contribution to the objective function (i.e., a customer is visited at most ones). Rather than using a single objective function in which each of the optimized dimensions is weighted, we consider a multi-objective cost function. We simultaneously minimize the total travel time including any waiting times (due to time windows), and maximize the total demand fulfilled by the vehicle.

In many practical settings, customers are served according to a predefined sequence. Two examples are provided in Tsirimpas et al. (2008), Tatarakis and Minis (2009), and Minis and Tatarakis (2011). First, the Ex-van deliveries model, in which an Ex-van vehicle visits retail outlets (e.g., super market, Kiosk etc) in a predefined sequence in order to replenish their stock in a regular fashion. At each location, the Ex-van vehicle might decide to precede to the next customer or return back to the depot for stock replenishment. The Ex-van sales model is characterized by a unknown demand, and products short expiration dates (see also Giaglis et al. (2004), and Tatarakis (2007)). Second, material handling systems in a manufacturing shop, in which Automated Guided Vehicles Systems (AGVs) operate along predefined pathways connecting the material warehouse with the workcenters. Again, AGVs are able to return to the

material warehouse to be reloaded.

In this Chapter, we propose a dynamic programming (DP) algorithm to solve the problem at hand. When the customer sequence is not predefined, the SVRPTW is $\mathcal{NP}$-hard as it is a generalization of the TSP which is proven to be $\mathcal{NP}$-hard by Karp (1972). In this chapter, we show that the SVRPTW with a predefined customer sequence is still $\mathcal{NP}-$hard. Moreover, in case of a multi-criteria objective function a number of complicating factors are encountered. As we aim to determine the Pareto set of routes, instead of a single optimal solution, the amount of data that has to be processed in each iteration of the DP increases. The Pareto set of routes might include a large number of solutions making the selection of a solution not straightforward. Therefore, we propose an approximate time-dependent dynamic programming based algorithm (denoted as DP^{ϵ}), where ϵ is the worst case precisions, $0 < \epsilon < 1$. The worst case performance of DP^{ϵ} is guaranteed and its running time is polynomially bounded.

The main contributions of this chapter are twofold:

First, this chapter tackles a specific vehicle routing problem from a multi-objective point of view. In real-life, decision makers might have numerous contradictory and equally important objectives they jointly want to optimize. Our approach determines the set of points representing the compromise solutions between the different conflicting objectives. Moreover, new objectives can easily be introduced in our proposed framework without losing the relevance of the initial ones. Even non-cost driven objectives (i.e., drivers' workload, customers' satisfaction, CO_2-emissions etc) can be considered. Furthermore, road congestion is captured by assuming time-dependent travel times. Transportation and time limitations (customers' time windows, the vehicles' time availability and the limitation on the tours' duration) are taken into account.

Secondly, We prove that the SVRPTW with a predefined customer sequence is $\mathcal{NP}$-complete. We design an approximate time-dependent dynamic programming algorithm DP^{ϵ} with a provable worst case performance guarantee, and a polynomially bounded running time (i.e., an FPTAS). Furthermore, we show that DP^{ϵ} generates an approximate Pareto set of routes with fewer solutions, yet covering well the true Pareto set of routes.

This chapter is organized as follows. Section 4.2 reviews the literature relevant to our problem. In Section 4.3, the problem we intend to solve is formally described. Section 4.4 is devoted to the dynamic programming formulation. Section 4.5 presents the DP based approximation and the main results of the chapter are derived. In Section 4.6, a numerical study is conducted. Finally, Section 4.7 concludes with a summary of the main results.

4.2. Literature Review

Despite its practical importance, the single vehicle routing problem has received relatively little attention in the literature. In Tsirimpas et al. (2008), three single vehicle routing problems with a predefined customer sequence are addressed (when the vehicle has one or multiple compartments, and the pick up and delivery problem). In Tatarakis and Minis (2009), the stochastic (stochastic demand) single vehicle routing problem is considered. Here again, the sequence of customers is predefined and two variants of the problem are dealt with depending on the vehicle's type (whether it has one or multiple compartments). The stochastic single vehicle routing with delivery and pick up is considered in Minis and Tatarakis (2011). In Azi et al. (2007), the single vehicle routing problem with time windows and multiple tours is treated. In Gribkovskaia et al. (2007) and Gribkovskaia et al. (2008), the single vehicle routing problem with pickups and deliveries and the single vehicle routing problem with deliveries and selective pickups, in which it is not necessary to meet all pickup demands, are respectively considered. Süral and Bookbinder (2003) consider a single vehicle routing with unrestricted back-hauls. In Feillet et al. (2004a), a TSP with profits where not all nodes are necessarily visited is addressed. In this chapter, the vehicle might also not visit all nodes.

Contrary to most of the existing literature on single vehicle routing problems, we consider a multi-objective cost function. For an extensive literature review on multi-objective VRP models, we refer to Jozefowiez et al. (2008). In practice, managers might aim to minimize both the distance traveled and maximize the number of customers visited (Ribeiro and Lourenço, 2001), or to minimize both travel time and total customers' waiting time (Hong and Park, 1999). Multi-objective cost functions are very attractive for modeling practical situations in which contradictory objectives need to be optimized simultaneously. However, multi-objective cost functions are usually reduced to a composite single objective cost function by using a weighted sum of the various objectives (Rosenblatt and Sinuany-Stern, 1989). Ulungu and Teghem (1997) and Visee et al. (1998) argue that solutions obtained by the optimization of a composite single objective function are only a small subset of the entire Pareto set of solutions, and therefore could lead to suboptimal managerial decisions.

Traditionally, total travel costs are calculated in terms of distances: the overall distance traveled is minimized. Routes obtained as such, do not guarantee a good and feasible solution when applied in real-life. One major shortcoming is due to the difficulty of taking road time-dependent congestion into account. Despite numerous publications dealing with efficient scheduling methods for vehicle routing, very few have addressed, due to its complexity, the inherent dynamic nature of travel times. Ichoua et al. (2003) present a time-dependent model with simple travel time profiles

that satisfy the FIFO assumption. They adopt a parallel tabu search heuristic to obtain the schedules. Van Woensel et al. (2008) propose a more realistic model by applying queueing theory to better capture the congestion effects on travel times. In Van Woensel and Vandaele (2006) and Van Woensel et al. (2006), real-life data (simulation respectively) is used to validate their queuing approach for traffic flows.

In this chapter, we present a time-dependent dynamic programming (DP) algorithm approach to a specific VRP. Due to their complexity, vehicle routing problems are usually dealt with using (meta-)heuristics (see e.g., Bräysy and Gendreau (2005a); Bräysy and Gendreau (2005b); Taillard et al. (1997) for some good reviews). Next to heuristics, many researchers have adopted exact approaches to handle VRPs (see e.g., Laporte and Nobert (1980) for a good review). In Malandraki and Dial (1996) and Kok et al. (2010), a restricted DP heuristic is proposed to solve the time-dependent traveling salesman problem (TSP). In each iteration of the restricted DP, only a subset (hence "restricted") with a predefined size and consisting of the best solutions is kept and used to compute solutions in the next iteration. In Tsirimpas et al. (2008), and Tatarakis and Minis (2009) suitable dynamic programs are introduced to solve several variants of the single vehicle routing problem with a predefined customer sequence. Time-dependency in dynamic programming was first introduced by Kostreva and Wiecek (1993) to solve a path-planning problem. In Klamroth and Wiecek (2000a) and Klamroth and Wiecek (2001), time-dependent dynamic programming is used to deal with the single machine scheduling and the capital budgeting problems.

4.3. Problem Description

The input for the SVRPTW consists of a graph $G = (V, A)$ where $V = \{0, 1, ..., N\}$ is the set of all nodes, such that $V_c = V \setminus \{0\}$ represents the set of customers that need to be served, and 0 is the depot. $A = \{(i, j) : i \neq j \text{ and } i, j \in V\}$ is the set of all arcs between the nodes. A single vehicle, with a finite capacity Q and a limited time availability T, performs several tours to serve the set of customers V_c according to a predefined sequence $1, 2, \ldots, N$. At each point along the sequence, the vehicle decides on whether to return to the depot and then serve customers in the rest of the sequence, or proceed to the next customers in the sequence. It is allowed to skip customers along the route (i.e., each customer is visited at most ones). To each node i, we associate a hard delivery time window with opening time a_i and closing time b_i. While the vehicle's arrival after the time window closing time is not feasible, its arrival before the opening time is allowed. In this case, a waiting time is incurred. Let $q_i(t)$ be a non-increasing function in time representing demand of node i upon arrival at time t. We assume that the depot has no demand, hence $q_0(t) = 0$, for all t. At

node i, a service time s_i is needed. We denote t_i departure time from node $i \in V$ and $\tau_{ij}(t_i)$ travel time from node i to node $j \in V, j > i$, which depends on t_i. Note that time-dependent travel times do not necessarily satisfy the triangle inequality. The vehicle performs multiple tours during its operating period T. The duration of each tour does not last more than t^{lim} time units, and the quantity delivered to customers should not exceed the vehicle's capacity. Figure 4.1 illustrates the decisions to be taken along the route.

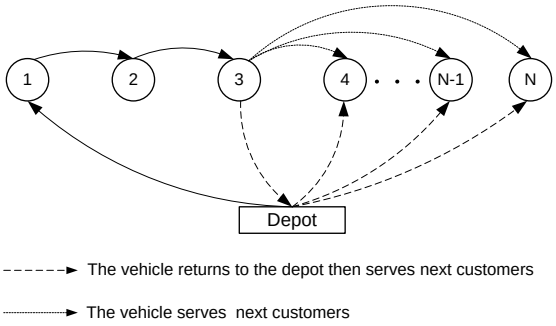

Figure 4.1 The decisions taken along the route.

In the following theorem, we prove that the SVRPTW studied in this chapter is $\mathcal{NP}-$complete.

Theorem 4.1 The SVRPTW with predefined customer sequence is $\mathcal{NP}$-complete even when travel time functions satisfy the triangle inequality, demand is constant over customers time windows, and the vehicle's capacity is unrestricted.

Proof: $\mathcal{NP}$-hardness for the SVRPTW is proved by a reduction from the 2-PARTITION problem (Garey and Johnson, 1979). An instance of the 2-PARTITION problem consists of k positive integers $z_1, \ldots, z_k$ such that $\sum_{i=1}^{k} z_i = 2Z$. the question for the 2-PARTITION problem is whether there exists a subset of indexes $I \subset \{1, \ldots, k\}$ such that $\sum_{i \in I} z_i = Z$?
From an arbitrary instance of 2-PARTITION, we construct the following instance of the SVRPTW with $N = k + 1$ nodes.

- For $1 \leq i \leq k$, time window for node i is such that $a_i = 0$ and $b_i = (2Z+1)k+1$. Its demand is $q_i = 2kz_i$ (constant over time). At node i the service time $s_i = 2kz_i$ is required.

- Node $k+1$ has a time window with opening time is $a_{k+1} = (2Z+1)k$ and closing time is $b_{k+1} = (2Z+1)k+1$. Its demand is $q_{k+1} = 100kZ$ (also constant over time). The service time required at node $k+1$ is 1.
- travel time between every two nodes i and j $(i < j)$ is $\tau_{ij} = 1$ (travel time satisfies the triangle inequality).
- The vehicle's capacity is unrestricted.

In the following we prove that, for the SVRPTW, there exists a schedule that satisfies a total demand of $102kZ$ if and only if the 2-PARTITION instance has answer YES. (Proof of $\Rightarrow$) Assume that $\sum_{i \in I} z_i = Z$ for some $I \subset \{1, \ldots, k\}$. Consider the schedule that serves all customers $i \in I$ together with customer $k+1$. The total travel time for the customers in I is $|I| \leq k$ and the total service time for these customers is $\sum_{i \in I} s_i = \sum_{i \in I} 2kz_i = 2kZ$.
Hence all customers in I can be served within their time window $[0, (2Z+1)k]$. Consequently, the vehicle is able to serve customer $k+1$ at time $b_{k+1} = (2Z+1)k+1$. The total demand delivered is $102kZ$.
(Proof of $\Leftarrow$) Assume that there is a schedule that satisfies a total demand of $102kZ$. Since the total demand of all customers is $4kZ$, this schedule must serve customer $k+1$ within its time window b_{k+1}. let $I \subset \{1, \ldots, k\}$ be the set of all customers served before customer $k+1$. The total service time of all customers in I is at most $b_{k+1} - 1 = (2Z+1)k$. Hence,

$$(2Z+1)k \geq \sum_{i \in I} s_i = 2k \sum_{i \in I} z_i$$

Since Z and all z_i are integers, we have that

$$\sum_{i \in I} z_i \leq Z \tag{4.1}$$

On the other hand, the total demand of the customers in I must be at least

$$102kZ - 100kz = 2kz \tag{4.2}$$

Hence,

$$\sum_{i \in I} z_i \geq Z \tag{4.3}$$

Inequalities 4.1 and 4.3 imply that $\sum_{i \in I} z_i = Z$. This completes the proof of Theorem 4.1. □
In Table 4.1 we further define additional notation used in this chapter.

Variable	Description
t	Time (origin is always taken to be 0)
$c_{ij}(t)$	Vector of costs assigned to the arc $(i,j) \in A$ when leaving node i at time t
$\tau^*_{ij}(t)$	Travel time from node i to node j when leaving node i at time t. It includes waiting and service time at j
P^k_j	Set of Pareto partial routes with $k+1$ nodes starting at the depot and ending at node j. $P^k_{j,m}$ is the m^{th} element of P^k_j
X^k_j	Set of Pareto partial routes with at most $k+1$ nodes starting at the depot and ending at node j. $X^k_j = \bigcup_{l=0}^{k} P^l_j$
$G(X^k_j)$	Set of vector of costs corresponding to the set of partial routes in X^k_j $G(P^k_{j,m})$ is the vector cost of the partial path $P^k_{j,m}$
$\|A\|$	Cardinality of a set A
$[\![u,v]\!]$	The interval of integer numbers between the integer u and the integer v (u and v included)
φ_k	The sum from 1 to k. k is integer and $\varphi_k = \sum_{l=1}^{k} l$
$\lceil x \rceil$	Nearest integer larger or equal to the real number x

Table 4.1 Notation used in this chapter

4.3.1 Travel time and demand functions

We divide the planning horizon into time zones where a different speed is associated with each of these zones. The resulting stepwise speed function is translated into travel time functions that satisfy the First-In First-Out (FIFO) principle. The FIFO principle avoids the undesired effect of passing. Usually traffic networks have a morning and an afternoon congestion period. Therefore, we consider speed profiles that have two periods with relatively low speeds. In the rest of the planning horizon, speeds are relatively high. This complies with data collected for a European highway (Van Woensel and Vandaele, 2006). Figure 4.2 depicts the speed profile for each start time for an arbitrary link. Moreover, it shows how the speed profile is translated into a travel time function using the procedure as described in Ichoua et al. (2003).

While the slopes in the travel time function mean that the traveled distance is traversed using several speeds, the horizontal segments mean that it is traversed using only one speed. The FIFO principle can be stated as follows:

FIFO principle: For every two nodes i and j, and times t and t', $t \leq t'$ implies that:

$$t + \tau_{ij}(t) \leq t' + \tau_{ij}(t')$$

Customers' demand is also considered to be time-dependent. In fact, the quality of the products (e.g., perishable products like fish, fresh milk etc) deteriorates when time elapses. Consequently, customers are willing to receive their orders as early as

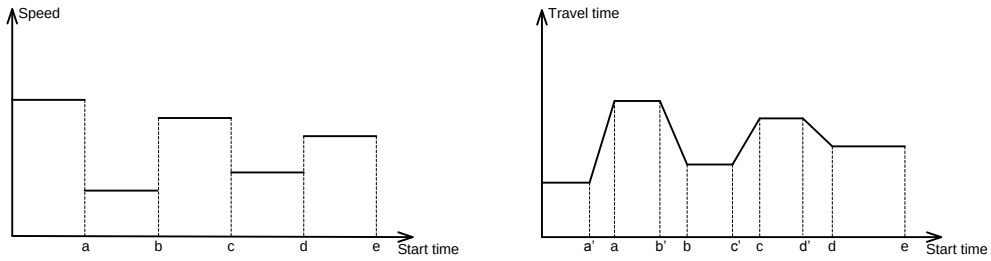

Figure 4.2 Speed and travel time functions.

possible. A later delivery results in a decrease in the quantity customers want to have. Figure 4.3 illustrates demand of customer i as a function of the arrival time. The initial order quantity q_{max} is delivered if the arrival time is before the opening time a_i. After time a_i, demand starts decreasing linearly. At closing time, only a fraction $r.q_{max}$, $0 \leq r \leq 1$, of the initial order is delivered. After closing time, delivery is not allowed.

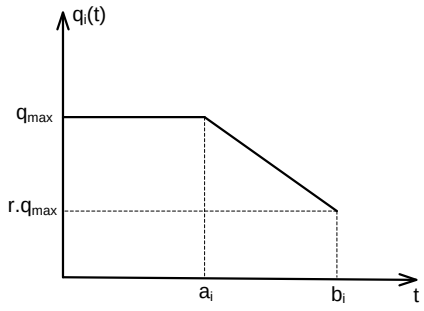

Figure 4.3 Demand function.

4.4. Dynamic Programming for the SVRPTW

To each node $j \in V$, we associate time-dependent costs: a visited node contributes with some travel time (including waiting and service time) and demand delivered, to the objective function. If the vehicle leaves node i at time t_i towards node j, the arrival time at node j is $t_i + \tau_{ij}(t_i)$. The vector travel costs related to the link from node i to node j, at time t_i is defined as follows:

$$c_{ij}(t_i) = \begin{bmatrix} \tau^*_{ij}(t_i) = \tau_{ij}(t_i) + \max(0, a_j - t_i - \tau_{ij}(t_i)) + s_j \\ q_j(t_i + \tau_{ij}(t_i)) \end{bmatrix}$$

The first element is the travel time including waiting and service time at node j. The second element is the demand upon arrival at node j.

For every partial route R, starting at the depot and ending at node $i \in V$. We define $c_{ij}(R)$ as the state-dependent travel costs vector related to the link between node i and node j. The state of the route R is represented by (θ, η, t_i), where θ is the quantity delivered to customers served during the last tour, η denotes the departure time from the last depot visited along the partial route R, and t_i is the departure time from node i. $c_{ij}(R)$ is expressed as:

$$c_{ij}(R) = \begin{cases} c_{ij}(t_i) & \text{if } \begin{cases} t_i + \tau_{ij}(t_i) \leq b_j \\ t_i + \tau^*_{ij}(t_i) \leq \eta + t^{lim} \\ \theta + q_j(t_i + \tau_{ij}(t_i)) \leq Q \end{cases} \\ \begin{bmatrix} +\infty \\ -\infty \end{bmatrix} & \text{otherwise} \end{cases} \tag{4.4}$$

The new state-dependent travel costs vector (4.4) ensures that the vehicle, currently planned to serve node i, may travel to node j, only if the vehicle arrives before the closing time b_j of node j, if the duration of the current tour does not exceed the maximum allowed tour duration t^{lim}, and if enough transportation capacity to serve node j is available. In fact, extremely high costs are assigned to schedules that violate any of the constraints and hence such schedules are infeasible.

For the vector travel costs's first element, we can show (see appendix) that the FIFO property is preserved. Therefore, we state the following lemma:

Lemma 4.1 For every two nodes i and j, and times t and t', $t \leq t'$ implies that:

$$t + \tau^*_{ij}(t) \leq t' + \tau^*_{ij}(t')$$

For each partial route R, let $v(R)$ be the end node of route R, $\Gamma(R)$ the arrival time at $v(R)$ and $D(R)$ the total demand along route R. Furthermore, we define $\max(R)$ as the latest customer visited along the route R. For example, if $R = 0 \rightarrow 1 \rightarrow 0 \rightarrow 4 \rightarrow 7 \rightarrow 0$, then $\max(R) = 7$. A route R_2 is dominated by a route R_1, if any feasible extension of R_2 can be obtained by extending R_1 and the resulting extension of R_1 is better (in all objectives) than the extension of R_2. We formulate the following lemma:

Lemma 4.2 Route R_2 is dominated by route R_1 if:

1. $v(R_1) = v(R_2)$
2. $\Gamma(R_1) \leq \Gamma(R_2)$
3. $D(R_1) \geq D(R_2)$
4. $\max(R_1) \leq \max(R_2)$

Proof: : Let $R_1 = v_0 \rightarrow v_1 \rightarrow ... \rightarrow v_p$ and $R_2 = v'_0 \rightarrow v'_1 \rightarrow ... \rightarrow v'_q$ such that $v(R_1) = v_p = v(R_2) = v'_q$ and $v_0 = v'_0 = 0$. Furthermore, let $R^* = u_0 \rightarrow u_1 \rightarrow ... \rightarrow u_r$ be a feasible extension of R_2 such that $u_r = 0$. Note that other nodes visited along R_1, R_2 and R^* might also represent the depot 0 because routes might contain several tours starting and ending at the depot. Customers along R_1, R_2 and R^* are visited according to a predefined sequence. Therefore, for nodes representing customers (i.e., different from 0), $l < m$ implies $v_l < v_m$, $v'_l < v'_m$ and $u_l < u_m$.
For any two partial routes R_a and R_b, we denote $R_a \oplus R_b$ the extension of partial route R_a by the partial route R_b.
First we prove that R^* is also a feasible extension of R_1. We consider an arbitrary node $u_l \in R^*$. R^* is a feasible extension of R_2. Hence, $v > \max(R_2) > \max(R_1)$. Now, we only need to prove that the time window of node u_l is respected when reached trough R_1. Let R_{u_m} be the partial route of R^* such that $R_{u_l} = u_0 \rightarrow u_1 \rightarrow ... \rightarrow u_m$. The FIFO propriety and Condition 4 of Lemma 4.2 lead to the following:

$$\Gamma(R_1 \oplus R_{u_l}) \leq \Gamma(R_2 \oplus R_{u_l}) \tag{4.5}$$

Hence,

$$\Gamma(R_1 \oplus R_{u_l}) \leq b_{u_l}$$

Hence, R^* is a feasible extension of R_1.
Equation (4.5) is also true for u_r. Hence,

$$\Gamma(R_1 \oplus R_{u_r}) \leq \Gamma(R_2 \oplus R_{u_r})$$

this means that the extension of R_1 dominates that of R_2 in the first objective. This also holds for the second objective as demand is decreasing in time and arrival at customers visited along R^* will always be earlier when reached through R_1 than when reached through R_2. This completes the proof of Lemma 4.2. □

A route R consists of at most N tours, and the total demand fulfilled along R is at most NQ. This is the case when all customers are served, each tour includes only one customer, and each customer gets a quantity Q upon arrival.
The dynamic programming DP works as follows: in the k^{th} iteration, for every node $j \in V$, node $i \in V$ is added to all partial Pareto routes with k visited nodes starting

at the depot and ending at node j (i.e., partial routes in P_j^{k-1}). The newly generated partial routes with $k+1$ visited nodes are added to the set of partial Pareto routes with at most k visited nodes starting at the depot and ending at node i (i.e., partial routes in X_i^{k-1}), all routes are then evaluated, and only Pareto ones are kept which results in the set of partial Pareto routes with at most $k+1$ visited nodes starting at the depot and ending at node i. We distinguish between two cases. First, when node i is the depot (i.e., the vehicle drives back to the depot). In this case, partial routes can always be extended by node i (the resulting partial route is of course discarded if it turns out after evaluation that it is infeasible). Secondly, when node i is a customer (i.e., $i > 0$). In this case, partial routes are extended by node i only if their latest visited customers is smaller than i. DP finishes when no more feasible extensions are possible. DP is formulated as follows:

$$G(X_i^0) = c_{0i}\,(0 \to i)\,, \quad \forall i \in V_c$$

and for all $k \in [\![1, 2N]\!]$:

$$G(X_0^k) = \underset{j \in V_c}{vopt}\left\{G(X_0^{k-1}), \left\{G(P_{j,m}^{k-1}) + c_{j0}(P_{j,m}^{k-1}) : m \in [\![1, |P_j^{k-1}|]\!]\right\}\right\}$$

$$G(X_{i>0}^k) = \underset{j \in V}{vopt}\left\{G(X_{i>0}^{k-1}), \left\{G(P_{j,m}^{k-1}) + c_{ji}(P_{j,m}^{k-1}) : m \in [\![1, |P_j^{k-1}|]\!], \max\left(P_{j,m}^{k-1}\right) < i\right\}\right\}$$

DP is initialized by the cost vector $c_{0i}\,(0 \to i)$ of adding the first customer i to the depot. The first recursion equation handles the case when the node to be added is the depot. The second recursion equation handles the case when the node to be added is a customer.

Solving large instances might be computationally very expensive. Furthermore, the size of the generated Pareto routes might increase exponentially which makes it difficult for the decision maker to select a solution. Therefore, in the next section, we develop an approximate DP based algorithm to deal with these issues.

4.5. Approximating the SVRPTW

In order to reduce the complexity of DP, we impose extra structure to its execution. The set $G(X_i^k)$ may contain many solutions that are very close to each other. Therefore, we trim, in each iteration k and for every node $i \in V$, the generated set of Pareto solutions $G(X_i^k)$. More specifically, the set $G(X_i^k)$ is trimmed by reducing the solutions that are very close to each other, and then the trimmed set $G(\widetilde{X}_i^k)$ is used in

the dynamic program to approximately compute the untrimmed set $G(X_i^{k+1})$. This approach of adding structure to the execution of algorithms was first introduced by Ibarra and Kim (1975). Sahni (1976) applied it to a variety of scheduling problems. Woeginger (2005) applied the trimming method to the problem of scheduling two parallel machines.

Formally, the set $G(X_i^k)$, generated in the k^{th} iteration, can be represented by geometric points in the rectangle $[0, T] \times [0, NQ]$. The rectangle is cut into multiple boxes of exponentially increasing size. Solutions contained by the same box form a cluster of solutions that are considered to be very close to each other. From each cluster, only the solution with the smallest total travel time is retained. The choice for the solution with the smallest total travel time is imposed by time windows. In fact, any feasible extension of a solution in a cluster of solutions is a feasible extension of the solution, from the same cluster, with the smallest total travel time. Let us define Δ_1 and Δ_2 as:

$$\Delta_1 = 1 + \frac{\epsilon}{2\varphi_{2N}} \quad \text{and} \quad \Delta_2 = 1 - \frac{\epsilon}{2\varphi_{2N}} \tag{4.6}$$

where ϵ is a real number between 0 and 1. In the k^{th} iteration of the dynamic program, the cuts on the travel time axis are executed at the coordinates $\Delta_1^{km}, m = 1, 2, ..., L_1$, and the cuts on the quantity delivered axis are executed at the coordinates $\Delta_2^{-kp}, p = 1, 2, ..., L_2$. The values of L_1 and L_2 are chosen such that $\Delta_1^{kL_1} \leq T$ and $\Delta_2^{-kL_2} \leq NQ$. We set:

$$\begin{cases} L_1 = \lceil \frac{\ln T}{k \ln \Delta_1} \rceil \leq \lceil \frac{1}{k}(1 + \frac{2\varphi_{2N}}{\epsilon}) \ln T \rceil \\ L_2 = \lceil \frac{\ln NQ}{k \ln \frac{1}{\Delta_2}} \rceil \leq \lceil \frac{2\varphi_{2N}}{k\epsilon} \ln NQ \rceil \end{cases}$$

Figure 4.4 illustrates the reduction of the set of Pareto solutions. Obviously, the size of the boxes depends on the precision ϵ. In fact, smaller precisions result in smaller boxes. Furthermore, the size of boxes increases along with the execution of the dynamic programming. In fact, boxes are smaller in the early iterations of the dynamic programming. Therefore, the trimming action has more impact in later stages of the execution of the dynamic programming. Boxes with an exponentially increasing size results in polynomial number of boxes. Hence, after the trimming action, only a polynomial number of solutions will be kept. Note that if, for instance, the boxes' size increases linearly, the number of solution kept after trimming will still be exponential. Furthermore, the form of Δ_1 and Δ_2 given in equation (4.6) is justified by two reasons. First, their values are very close to one. Hence, two solutions in the same box are indeed very close to each other. Second, we know the limit of the sequences $\left(1 + \frac{x}{a}\right)^a$ and $\left(1 - \frac{x}{a}\right)^a$ when a goes to infinity. In the k^{th} iteration, the trimmed set of Pareto solutions contains at most $(L_1 + 1) \times (L_2 + 1)$ solutions. We

can compute the complexity of DP^{ϵ} as being proportional to:

$$\begin{aligned}\sum_{k=1}^{2N}\sum_{j=0}^{N}\sum_{i>j}|G(\widetilde{X}_i^k)| &= O\left(N\sum_{k=1}^{2N}L_1L_2\right)\\ &= O\left(\left(\frac{N\varphi_{2N}}{\epsilon}\right)^2\ln T\ln NQ\sum_{k=1}^{2N}\frac{1}{k^2}\right)\end{aligned}$$

The series $\sum_{k=1}^{\infty}\frac{1}{k^2}$ is non-decreasing and converges to $\frac{\pi^2}{6}$. Hence, we have:

$$\sum_{k=1}^{2N}\sum_{j=0}^{N}\sum_{i>j}|G(\widetilde{X}_i^k)| = O\left(\frac{N^6}{\epsilon^2}\ln T\ln NQ\right)$$

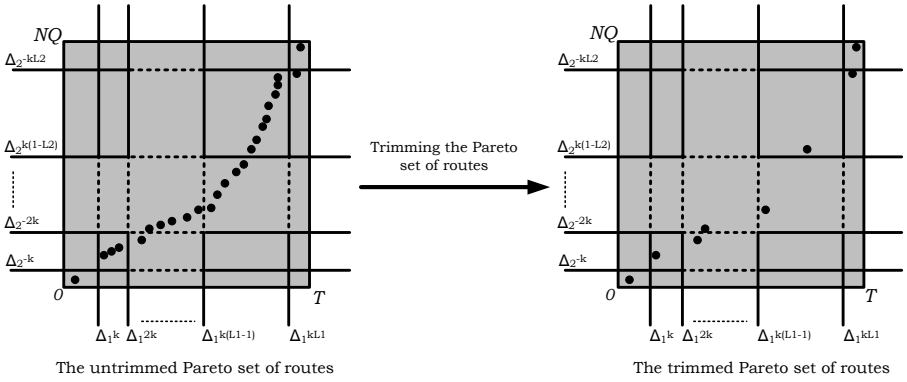

Figure 4.4 The reduction of the set of Pareto-optimal solutions.

We conclude that in any instance of the SVRPW, DP^{ϵ} runs in time polynomial in the size of the instance and in $\frac{1}{\epsilon}$.

Worst case performance guarantee of DP^{ϵ}:

By making an additional assumption concerning the travel times' structure, DP^{ϵ} has a provable worst case performance guarantee. In the case of multi-objective optimization problems, a worst case performance guarantee is such that for every Pareto solution, there exists an approximate-Pareto solution that is not, in all objectives, worse by more than a known factor than the Pareto solution. In our case, the worst case performance guarantee is such that for every solution given by DP, the total travel time including any waiting time is at most a factor $f_1(\epsilon)$ from

that of a DP^ϵ solution, and the total quantity delivered is at least a factor $f_2(\epsilon)$ away from that of the DP solution. $f_1(\epsilon)$ and $f_2(\epsilon)$ are two functions respectively increasing and decreasing in ϵ. To guarantee the worst case precision for DP^ϵ, we impose the following condition on the structure of travel times:

Assumption 4.1 For any two nodes i and j, for any $\alpha \geq 1$, and for any two times $t, t' \in \mathbb{R}$ such that $t \leq \alpha t'$:

$$q_i(t) \geq \frac{q_i(t')}{\alpha} \quad \text{and} \quad \tau_{ij}(t) \leq \alpha \tau_{ij}(t')$$

Assumption 4.1 means that leaving node i towards node j at a later time $\alpha t'$, instead of time t, will not multiply travel time by more than a coefficient α (e.g., in case of stepwise functions, high degree polynomial functions, exponential functions etc). In other words, fast increases in travel time are not allowed. For instance, the travel time functions $\tau_{ij}(t) = t, \tau_{ij}(t) = \sqrt{t}$ and $\tau_{ij}(t) = \ln t$ satisfy Assumption (4.1). Similarly, fast decreases in demand are not allowed. For instance, it can be checked that, for every node $i \in V_c$, demand functions with $r = \frac{b_i}{2b_i - a_i}$ (r is illustrated in Figure 4.3) satisfy Assumption 4.1. if Assumption 4.1 is satisfied, the following lemma follows:

Lemma 4.3 For every two nodes i and j, and every real number $\alpha \geq 1$, it holds that for every time t:

$$q_i(\alpha t) \geq \frac{q_i(t)}{\alpha} \quad \text{and} \quad \tau^*_{ij}(\alpha t) \leq \alpha \tau^*_{ij}(t)$$

In the following lemma, we prove some useful inequalities that hold for points contained by the same box.

Lemma 4.4 Let $k \in [\![0, 2N]\!]$. Let $z = \begin{bmatrix} z_1 \\ z_2 \end{bmatrix}$ and $\tilde{z} = \begin{bmatrix} \tilde{z_1} \\ \tilde{z_2} \end{bmatrix}$ be two points generated in the k^{th} iteration of DP^ϵ. If z and $\tilde{z}$ are in the same box, then:

$$\frac{z_1}{\Delta_1^k} \leq \tilde{z_1} \leq \Delta_1^k z_1 \quad \text{and} \quad \Delta_2^k z_2 \leq \tilde{z_2} \leq \frac{z_2}{\Delta_2^k}$$

Proof: There exists some $r_1 \in [\![1, L_1]\!]$ and $r_2 \in [\![1, L_2]\!]$ such that:

$$\Delta_1^{k(r_1-1)} \leq z_1 \leq \Delta_1^{kr_1} \quad \text{and} \quad \Delta_2^{k(r_2-1)} \leq z_2 \leq \Delta_2^{kr_2} \tag{4.7}$$

We can also write the same for $\tilde{z}$. Hence:

$$\Delta_1^{k(r_1-1)} \leq \tilde{z_1} \leq \Delta_1^{kr_1} \quad \text{and} \quad \Delta_2^{k(r_2-1)} \leq \tilde{z_2} \leq \Delta_2^{kr_2} \tag{4.8}$$

For $\tilde{z}$, we can write the inequalities (4.8) as:

$$\frac{\Delta_1^{kr_2}}{\Delta_1^k} \leq \tilde{z_1} \leq \Delta_1^k \Delta_1^{k(r_1-1)} \quad \text{and} \quad \frac{\Delta_2^{kr_2}}{\Delta_2^k} \leq \tilde{z_2} \leq \Delta_2^k \Delta_2^{k(r_2-1)}$$

Now we use the inequalities (4.7) to complete the proof of Lemma 4.4. □

Now, we formulate a lemma which states that in each iteration k of the DP, there is a solution whose first element is at most, and second element is at least, a known factor away from these of an approximate solution generated in the k^{th} iteration of DP^{ϵ}.

Lemma 4.5 In the k^{th} iteration and for every node $i \in V$ it holds that: For every solution $\begin{bmatrix} x_{i,k} \\ y_{i,k} \end{bmatrix} \in G(X_i^k)$, there exists a solution $\begin{bmatrix} \widetilde{x}_{i,k} \\ \widetilde{y}_{i,k} \end{bmatrix} \in G(\widetilde{X}_i^k)$ such that, for a given start t_0 time at the depot, it holds that:

$$\begin{cases} \widetilde{x}_{i,k} \leq \Delta_1^{\varphi_k} x_{i,k} + (\Delta_1^{\varphi_k} - 1)\, t_0 \\ \widetilde{y}_{i,k} \geq \Delta_2^{\varphi_k} y_{i,k} \end{cases}$$

Proof: : see appendix □

Theorem 4.2 In the k^{th} iteration it holds that: For every solution $\begin{bmatrix} x_{0,k} \\ y_{0,k} \end{bmatrix} \in G(X_0^k)$, there exists a solution $\begin{bmatrix} \widetilde{x}_{0,k} \\ \widetilde{y}_{0,k} \end{bmatrix} \in G(\widetilde{X}_0^k)$ such that, for a given start t_0 time at the depot, it holds that:

$$\begin{cases} \widetilde{x}_{0,k} \leq (1+\epsilon) x_{0,k} + \epsilon t_0 \\ \widetilde{y}_{0,k} \geq (1-\epsilon)\, y_{0,k} \end{cases}$$

Proof: In the k^{th} iteration and according to Lemma 4.5 it holds that, for every solution $\begin{bmatrix} x_{0,k} \\ y_{0,k} \end{bmatrix} \in G(X_0^k)$, there is a solution $\begin{bmatrix} \widetilde{x}_{0,k} \\ \widetilde{y}_{0,k} \end{bmatrix} \in G(\widetilde{X}_0^k)$ such that:

$$\begin{cases} \widetilde{x}_{0,k} \leq \Delta_1^{\varphi_k} x_{0,k} + (\Delta_1^{\varphi_k} - 1)\, t_0 \\ \widetilde{y}_{0,k} \geq \Delta_2^{\varphi_k} y_{0,k} \end{cases}$$

The sequence $\left(1 + \frac{\epsilon}{\varphi_{2N}}\right)^{\varphi_{2N}}$ is increasing and converges to $e^{\frac{\epsilon}{2}}$ when N goes to infinity. Moreover, $\left(1 - \frac{\epsilon}{2\varphi_{2N}}\right)^{\varphi_{2N}}$ is decreasing and converges to $e^{-\frac{\epsilon}{2}}$ when N goes to infinity. Hence, for every $N \geq 1$:

$$\left(1+\frac{\epsilon}{2\varphi_{2N}}\right)^{\varphi_{2N}} \leq e^{\frac{\epsilon}{2}} \quad \text{and} \quad \left(1-\frac{\epsilon}{2\varphi_{2N}}\right)^{\varphi_{2N}} \geq e^{-\frac{\epsilon}{2}}$$

Furthermore, for $0 < \epsilon < 1$, we have:

$$e^{\frac{\epsilon}{2}} \leq 1+\epsilon \quad \text{and} \quad e^{-\frac{\epsilon}{2}} \leq 1-\epsilon$$

Therefore, we have the following important result:

$$\begin{cases} \tilde{x}_{0,k} \leq (1+\epsilon)x_{0,k} + \epsilon t_0 \\ \tilde{y}_{0,k} \geq (1-\epsilon)\, y_{0,k} \end{cases}$$

This completes the proof of Theorem 4.2. □

An additional error arises when the vehicle is dispatched at a later moment $t_0 > 0$. Therefore, although a later dispatch might result in a reduced total travel time (e.g., congestion might be avoided), such a decision results in the additional error ϵt_0. When the vehicle's dispatch time at the depot is 0, DP^ϵ is an FPTAS. It is possible to set different precisions for the different objectives. We can have a precision ϵ_1 for the first objective and a precision $\epsilon_2 \neq \epsilon_1$ for the second objective. In this case, the second objective depends on the first objective precision ϵ_1 as well. This is due to the fact that demand depends on arrival time which is affected by ϵ_1. When the precisions are different for the different objectives, additional conditions should hold to preserve the FPTAS propriety of DP^ϵ.

4.6. Computational Results

For our numerical study, we use the well known Solomon's data sets (Solomon, 1987) that follow a naming convention of $DTm.N$. D is the geographic distribution of the customers which can be R (Random), C (Clustered) or RC (Randomly Clustered). T is the instance type which can be either 1 (instances with tight time windows) or 2 (instances with wide time windows). m denotes the number of the instance and N the number of customers that need to be served. Road congestion is taken into account by assuming that vehicles travel through the network using different speed profiles. We consider speed profiles with two congested periods. Speeds in the rest of the planning horizon (i.e., the depot's time window) are relatively high. We consider speed profiles that comply with data from real life. Furthermore, we assume three types of links: fast, normal and slow. Slow links might represent links within the city center, fast links might represent highways and normal links might represent the transition from highways to city centers. Moreover, without loss of generality, we

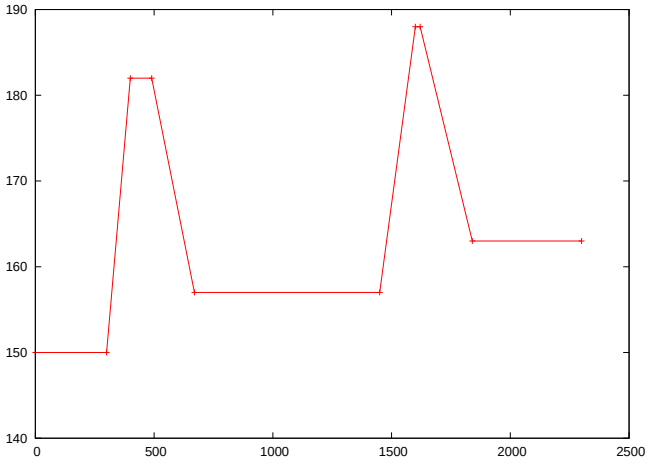

Figure 4.5 Example of a travel time function.

assume that breakpoints are the same for all speed profiles as congestion tends to happen around the same time regardless of the link's type (e.g., rush hours). The choice for a link type is done randomly and remains the same for all instances. The following speed profiles are considered:

	Zone1	**Zone2**	**Zone3**	**Zone4**	**Zone5**
Fast	2	1.67	1.92	1.58	1.83
Normal	1.42	1.08	1.33	1	1.25
Slow	0.92	0.58	0.83	0.5	0.75

Table 4.2 Speed Profiles

where $Zone1 = [0, 0.2T[$, $Zone2 = [0.2T, 0.3T[$, $Zone3 = [0.3T, 0.7T[$, $Zone4 = [0.7T, 0.8T[$ and $Zone5 = [0.8T, T]$. The planning horizon $T = b_0$ is equal to the upper bound of the depot's time window. Figure 4.5 illustrates the travel time function of an arbitrary link from an R instance. Furthermore, we consider demand functions as depicted in Figure 4.3 where, for ever node $i \in V$ we take $r = \frac{b_i}{2b_i - a_i}$. Moreover, q_{max} is two times the demand of customer i as given in the data sets of Solomon. While demand functions satisfy Assumption 4.1, the assumption is relaxed for travel time functions. Customers' should be visited in a predefined sequence defined according to their times windows.

We set the restriction on the tours duration to $t^{lim} = \frac{T}{3}$, and the precision to $\epsilon_1 = \epsilon_2 \in \{0.05, 0.1, 0.3\}$. Furthermore, we consider only type 2 instances with 100 customers.

Type 2 instances have large vehicle capacity and planning horizon, which allows for many long tours.

4.6.1 Comparing DP and DP^ϵ

The detailed Tables 4.3, 4.4 and 4.5 show that the computation time (columns "Time(s)") of the DP^ϵ algorithm decreases when we increases the values of the precision vector ϵ. Moreover, the size of the generated approximate Pareto fronts (columns "Size") decreases too.

	Exact		$\epsilon = 0.05$		$\epsilon = 0.1$		$\epsilon = 0.3$	
Instances	**Time(s)**	**Size**	**Time(s)**	**Size**	**Time(s)**	**Size**	**Time(s)**	**Size**
R201	467	248	245	165	205	124	155	95
R202	650	158	485	146	435	129	330	91
R203	2350	232	1834	181	1409	164	875	140
R204	3798	211	2803	192	2213	178	980	140
R205	1350	189	854	168	697	145	395	114
R206	1054	180	713	173	630	159	451	115
R207	2709	224	1955	196	1612	182	960	158
R208	4103	217	2750	190	1987	183	1143	157
R209	1050	208	670	173	542	153	395	110
R210	1008	294	606	206	496	168	302	139
R211	2367	260	1504	208	1340	185	765	131
Total	20906	2421	14419	1998	11566	1770	6751	1390

Table 4.3 DP vs. DP^ϵ for R instances

	Exact		$\epsilon = 0.05$		$\epsilon = 0.1$		$\epsilon = 0.3$	
Instances	**Time(s)**	**Size**	**Time(s)**	**Size**	**Time(s)**	**Size**	**Time(s)**	**Size**
C201	270	254	105	132	101	87	71	66
C202	340	151	165	118	143	107	95	85
C203	1113	180	575	141	470	126	320	98
C204	2801	163	1107	137	820	129	609	116
C205	405	205	180	165	160	133	112	88
C206	411	215	183	151	162	134	104	101
C207	664	352	275	202	209	166	112	111
C208	634	208	312	171	243	154	167	118
Total	6638	1728	2902	1217	2308	1036	1590	783

Table 4.4 DP vs. DP^ϵ for C instances

In the aggregate Table 4.6, we show the gains (in % with regard to DP) we obtain in computation times, and the reductions in the number of solutions contained by the generated approximate Pareto fronts. Gains with up to 56% are achieved for

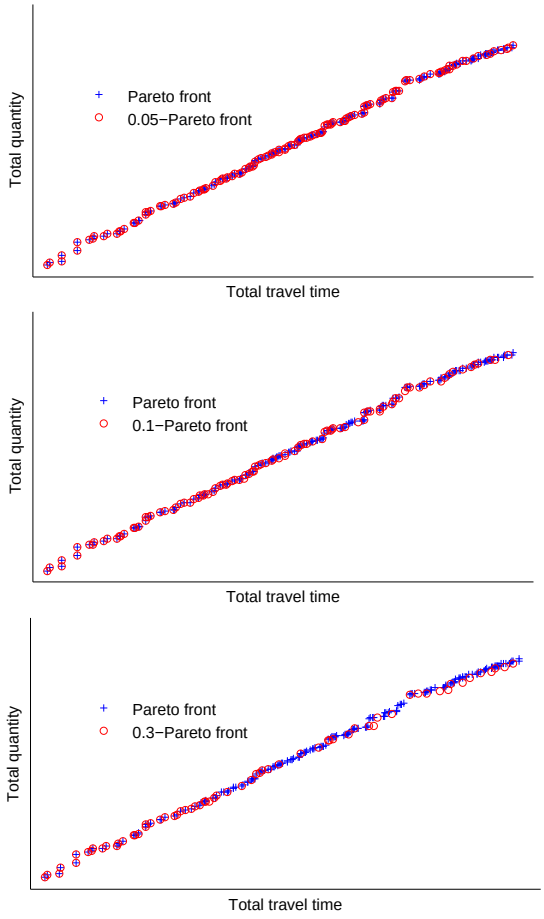

Figure 4.6 Pareto front vs. ϵ-Pareto fronts.

$\epsilon = 0.05$. For an $\epsilon = 0.1$, we gain up to 65% in computation times, and for $\epsilon = 0.3$ up to 76%. We also observe considerable reductions in the number of solutions contained by the approximate Pareto fronts. Figure 4.6 illustrates the Pareto front and the approximate Pareto fronts for the different values of the precision ϵ.

4.7. Conclusions

Vehicle routing problems with time windows are $\mathcal{NP}$-hard. However, some VRPs encountered in practice have special features that moderate their hardness. In this chapter, a VRP with time windows where a single vehicle performs multiple tours

Exact		$\epsilon = 0.05$		$\epsilon = 0.1$		$\epsilon = 0.3$	
Instances	**Time(s)**	**Size**	**Time(s)**	**Size**	**Time(s)**	**Size**	**Time(s)**	**Size**
RC201	324	160	212	137	202	126	145	80
RC202	287	170	180	145	172	126	147	98
RC203	1865	198	920	171	690	164	453	123
RC204	4240	397	1705	235	1235	183	750	117
RC205	427	214	325	212	290	172	193	126
RC206	587	174	413	154	395	140	234	115
RC207	957	216	715	189	596	171	386	131
RC208	1810	227	1346	196	910	173	495	135
Total	10497	1756	5816	1439	4490	1255	2803	925

Table 4.5 DP vs. DP^{ϵ} for RC instances

	$\epsilon = 0.05$		$\epsilon = 0.1$		$\epsilon = 0.3$	
	Time	**Size**	**Time**	**Size**	**Time**	**Size**
R instances	31%	17%	45%	27%	68%	43%
C instances	56%	30%	65%	40%	76%	55%
RC instances	45%	18%	57%	29%	73%	47%

Table 4.6 Gains in computation time and number of solutions

to serve a set of customers with a predefined sequence is considered. We prove the specific vehicle routing problem is still $\mathcal{NP}$-hard, yet in the weak sense. A flexible dynamic programming algorithm is proposed in which many practical features can easily be included. In fact, we take road congestion into account by assuming time-dependent travel times. Furthermore, rather than assuming a single objective cost function, a multi-criteria objective function is considered. In this way, we are provided with the complete set of Pareto solutions instead of one optimal solution. To further reduce computation times a dynamic programming approximation based on the trimming method is introduced. In opposite to Chapters 2 and 3, the trimming method presented in this chapter is iteration dependent. In fact, the impact of the trimming action is more significant towards the end of the dynamic programming. The approximation has again provable worst case precision and runs in time polynomial in the size of the instance and in $\frac{1}{\epsilon}$, where ϵ is the approximation's precision. Hence, the approximation is an FPTAS for the specific VRP studied in this chapter. Considerable gains in computation time are achieved, and the number of solution contained by the approximate Pareto fronts is significantly reduced. Numerical results are provided for the modified Solomon instances with 100 customers, and for a bi-criteria objective function.

Appendix

Proof of Lemma 4.1:

Let t_1 and t_2 be two moments, such that $t_1 \leq t_2$. We have:

$$\begin{aligned} t_1 + \tau^*_{ij}(t_1) - t_2 - \tau^*_{ij}(t_2) \quad &= t_1 + \tau_{ij}(t_1) + \max(0, a_j - t_1 - \tau_{ij}(t_1)) \\ &\quad -t_2 - \tau_{ij}(t_2) - \max(0, a_j - t_2 - \tau_{ij}(t_2)) \end{aligned} \tag{4.9}$$

Observing that for any two real numbers a and b, the following equality is always true:

$$a + \max(0, b - a) = \max(a, b) \tag{4.10}$$

We write:

$$t_1 + \tau^*_{ij}(t_1) - t_2 - \tau^*_{ij}(t_2) = \max(a_j, t_1 + \tau_{ij}(t_1)) - \max(a_j, t_1 + \tau_{ij}(t_1))$$

Because of the FIFO assumption, we have:

$$t_1 + \tau_{ij}(t_1) \leq t_2 + \tau_{ij}(t_2)$$

Hence:

$$\max(a_j, t_1 + \tau_{ij}(t_1)) \leq \max(a_j, t_2 + \tau_{ij}(t_2))$$

And therefore:

$$t_1 + \tau^*_{ij}(t_1) \leq t_2 + \tau^*_{ij}(t_2)$$

This completes the proof of Lemma 4.1. □

Proof of Lemma 4.3:

Let $\alpha \geq 1$ be a real number. For time t, we have:

$$\tau^*_{ij}(\alpha t) = \tau_{ij}(\alpha t) + \max(0, a_j - \alpha t - \tau_{ij}(\alpha t))$$

Again, using Equality 4.10, we obtain:

$$\begin{aligned} \tau^*_{ij}(\alpha t) &= \max(a_j - \alpha t, \tau_{ij}(\alpha t)) \\ &\leq \max(\alpha a_j - \alpha t, \alpha \tau_{ij}(t)) && \text{(using Assumption 4.1)} \\ &= \alpha \tau_{ij}(t) + \max(0, \alpha a_j - \alpha t - \alpha \tau_{ij}(t)) && \text{(using Equality 4.10)} \\ &= \alpha \tau^*_{ij}(t) \end{aligned}$$

This completes the proof of Lemma 4.3. □

Proof of Lemma 4.5:

In the k^{th} iteration, if $\begin{bmatrix} z_1 \\ z_2 \end{bmatrix}$ and $\begin{bmatrix} \tilde{z_1} \\ \tilde{z_2} \end{bmatrix}$ are two points in the same box, then:

$$\frac{z_1}{\Delta_1^k} \leq \tilde{z_1} \leq \Delta_1^k z_1 \quad \text{and} \quad \Delta_2^k z_2 \leq \tilde{z_2} \leq \frac{z_2}{\Delta_2^k} \tag{4.11}$$

To prove Lemma 4.5, we use induction on k.
Let i be a node. From (4.11). For $k = 1$, Lemma 4.5 follows directly from Lemma 4.4.
Let us assume Lemma 4.5 is true for $k-1$.
Let $\begin{bmatrix} x_{i,k} \\ y_{i,k} \end{bmatrix} \in G(X_i^k)$ and t_0 be a start time at the depot. Per definition of the set $G(X_i^k)$, $\begin{bmatrix} x_{i,k} \\ y_{i,k} \end{bmatrix}$ is feasible.

Hence, there exists a feasible point $\begin{bmatrix} x_{j,k-1} \\ y_{j,k-1} \end{bmatrix} \in G(X_j^{k-1})$ such that:

$$\begin{cases} x_{i,k} = x_{j,k-1} + \tau_{ji}^*(t_0 + x_{j,k-1}) \\ y_{i,k} = y_{j,k-1} + q_i(t_0 + x_{j,k-1}) \end{cases}$$

On the other hand, because of the induction assumption, there exists $\begin{bmatrix} \tilde{x}_{j,k-1} \\ \tilde{y}_{j,k-1} \end{bmatrix} \in G(\widetilde{X}_j^{k-1})$ such that:

$$\begin{cases} \tilde{x}_{j,k-1} \leq \Delta_1^{\varphi_{k-1}} x_{j,k-1} + \left(\Delta_1^{\varphi_{k-1}} - 1\right) t_0 \\ \tilde{y}_{j,k-1} \geq \Delta_2^{\varphi_{k-1}} y_{j,k-1} \end{cases}$$

Furthermore, DP^ϵ generates the solution $\begin{bmatrix} \tilde{x}_{j,k-1} + \tau_{ji}^*(t_0 + \tilde{x}_{j,k-1}) \\ \tilde{y}_{j,k-1} + q_i(t_0 + \tilde{x}_{j,k-1}) \end{bmatrix}$ in the k^{th} step. The point $\begin{bmatrix} \tilde{x}_{j,k-1} + \tau_{ji}^*(t_0 + \tilde{x}_{j,k-1}) \\ \tilde{y}_{j,k-1} + q_i(t_0 + \tilde{x}_{j,k-1}) \end{bmatrix}$ might be removed after trimming. However some vector $\begin{bmatrix} \tilde{x}_{i,k} \\ \tilde{y}_{i,k} \end{bmatrix}$, located in the same box as $\begin{bmatrix} \tilde{x}_{j,k-1} + \tau_{ji}^*(t_0 + \tilde{x}_{j,k-1}) \\ \tilde{y}_{j,k-1} + q_i(t_0 + \tilde{x}_{j,k-1}) \end{bmatrix}$ should be left.

From Lemma 4.4, we obtain:

$$\begin{cases} t_0 + \tilde{x}_{i,k} \leq \Delta_1^k \left(t_0 + \tilde{x}_{j,k-1} + \tau_{ji}^*(t_0 + \tilde{x}_{j,k-1})\right) \\ \tilde{y}_{i,k} \geq \Delta_2^k \left(\tilde{y}_{j,k-1} + q_i(t_0 + \tilde{x}_{j,k-1})\right) \end{cases}$$

Because of the FIFO principle and the induction assumption, we have:

$$\begin{cases} t_0 + \tilde{x}_{i,k} \leq \Delta_1^k \left(\Delta_1^{\varphi_{k-1}}(t_0 + x_{j,k-1}) + \tau_{ji}^* \left(\Delta_1^{\varphi_{k-1}}(t_0 + x_{j,k-1})\right)\right) \\ \tilde{y}_{i,k} \geq \Delta_2^k \left(\Delta_2^{\varphi_{k-1}} y_{j,k-1} + q_i \left(\Delta_1^{\varphi_{k-1}}(t_0 + x_{j,k-1})\right)\right) \end{cases}$$

Using Lemma 4.1, we obtain:

$$\begin{cases} t_0 + \tilde{x}_{i,k} \leq \Delta_1^k \left(\Delta_1^{\varphi_{k-1}}(t_0 + x_{j,k-1}) + \Delta_1^{\varphi_{k-1}} \tau_{ji}^*(t_0 + x_{j,k-1})\right) \\ \tilde{y}_{i,k} \geq \Delta_2^k \left(\Delta_2^{\varphi_{k-1}} y_{j,k-1} + \frac{1}{\Delta_1^{\varphi_{k-1}}} q_i (t_0 + x_{j,k-1})\right) \end{cases}$$

Hence,

$$\begin{cases} \tilde{x}_{i,k} \leq \Delta_1^{\varphi_k} x_{i,k} + (\Delta_1^{\varphi_k} - 1) t_0 \\ \tilde{y}_{i,k} \geq \Delta_2^{\varphi_k} y_{i,k} \end{cases}$$

This completes the proof of Lemma 4.5. □

Chapter 5

Branch and Cut and Price for the TDVRPTW

5.1. Introduction

The vehicle routing problem with time windows (VRPTW) concerns the determination of a set of routes starting and ending at a depot, in which the demand of a set of geographically scattered customers is fulfilled. Each route is traversed by a vehicle with a fixed and finite capacity, and each customer must be visited exactly once. The total demand delivered in each route should not exceed the vehicle's capacity. At customers, hard time windows are imposed, meaning that service at a customer is only allowed to start within its time window. The solution to the VRPTW consists of the set of routes with the least traveled distance.

Due to its practical relevance, the VRPTW has been extensively studied in the literature (Toth and Vigo, 2002). Consequently, many (meta-) heuristics and exact methods have been successfully developed to solve it. However, most of the existing models are time-independent, i.e., a vehicle is assumed to travel with constant speed throughout its operating period. Because of road congestion, vehicles hardly travel with constant speed (Ichoua et al., 2003). Obviously, solutions derived from time-independent models to the VRPTW could be infeasible when implemented in real-life. In fact, in real-life, road congestion results in tremendous delays. However, for many routes these delays are predictable. Therefore, feasibility can largely be guaranteed by considering time-dependent travel times when dealing with the VRPTW.

In this chapter, we consider the time-dependent vehicle routing problem with time

windows (TDVRPTW). We take road congestion into account by assuming time-dependent travel times: depending on the departure time at a customer a different travel time is incurred. We divide the planning horizon into time zones (e.g., morning, afternoon etc) where a different speed is associated with each of these zones. The resulting stepwise speed function is translated into travel time functions that do not allow overtaking. Such travel time functions satisfy the First-In First-Out (FIFO) principle (see also Ichoua et al. (2003)). Because of the time-dependency, the vehicles' dispatch times at the depot are crucial. In fact, a later dispatch time at the depot might result in a reduced travel time as congestion might be avoided. In this chapter, we aim to determine the set of routes with the least total travel time. Therefore, we decide on the sequence in which customers are visited along the routes, and vehicles' dispatch times at the depot.

Despite numerous publications dealing with the vehicle routing problem, very few addressed the inherent time-dependent nature of this problem. Additionally, to our knowledge, all existing algorithms are based on (meta-) heuristics, and no exact approach has been provided for the TDVRPTW. In this chapter, we solve the TDVRPTW exactly using a set partitioning model. We solve the linear relaxation of the set partitioning model using column generation. While the master problem of the column generation approach remains unchanged, compared to that of the VRPTW (as time-dependency is implicitly included in the set of feasible routes) the pricing problem is translated into a time-dependent elementary shortest path problem with resource constraints (TDESPPRC), where several resource variables governs time and path elementarity aspects. To guarantee integrality, the column generation algorithm is embedded in a branch-and-bound framework. Furthermore, in each node, we use cutting planes in the pricing problem to obtain better lower bounds and to ensure that the solution is feasible with respect to vehicle capacity. This results in a branch-and-cut-and-price (BCP) algorithm. Time-dependency in travel times increases the complexity of the pricing problem. In fact, the set of feasible solutions increases as the cost of a generated column (i.e., route) does not depend only on the visited customers, but also on the vehicles' dispatch time at the depot. The pricing problem in case of the VRPTW is usually solved by means of a labeling algorithm (Desrochers, 1986). In this chapter, we develop a time-dependent labeling (TDL) algorithm such that in each label the arrival time function (i.e., function of the departure time from the depot) of the corresponding partial path is stored. The TDL algorithm generates columns that have negative reduced cost together with their best dispatch time at the depot. To improve the performance of the TDL algorithm, new dominance criteria tailored to our BCP framework are introduced to discard labels not leading to routes in the final optimal solution. To accelerate the BCP algorithm, two fast heuristics are designed to easily find columns with negative reduced cost. Furthermore, we relax the pricing problem by allowing non-elementary paths. The resulting pricing problem is a time-

dependent shortest path problem with resource constraints (TDSPPRC). Although the TDSPPRC results in worse lower bounds, it is easier to solve and integrality is still guaranteed by branch-and-bound. Moreover, as shown in our numerical experiments, the TDSPPRC works well for instances with tight time windows.

The main contributions of this chapter are summarized as follows. First, we present an exact method for the TDVRPTW. We propose a branch-and-cut-and price algorithm to determine the set of routes with the least total travel time. Contrary to the VRPTW, the pricing problem is translated into a TDESPPRC and solved by a time-dependent labeling algorithm. Second, when considering time-dependent travel times, standard dominance tests taken directly from the VRPTW become weak and time consuming. In this chapter, we introduce new dominance criteria by exploiting the structure of the arrival time function.

The chapter is organized as follows. Section 5.2 reviews the literature relevant to our problem. In Section 5.3, a formal description of the studied problem is provided. In Section 5.4, the column generation algorithm is described. In Section 5.5, a detailed description of the labeling algorithm used to solve the pricing problem is provided. In Section 5.6, extensive numerical experiments are conducted. Finally, Section 5.7 concludes with a summary of the main results.

5.2. Literature Review

An abundant number of publications is devoted to the vehicle routing problem (see Laporte (1992), Toth and Vigo (2002), and Laporte (2007) for some reviews). For good reviews on the VRPTW, the reader is referred to Bräysy and Gendreau (2005a,b); Kallehauge (2008) and Gendreau and Tarantilis (2010). The majority of these publications assume a time-independent environment where vehicles travel with a constant speed throughout their operating period. Perceiving that vehicles operate in a stochastic and dynamic environment, more researchers moved their effort towards the optimization of the time-dependent vehicle routing problems. Nevertheless, literature on this subject remains scarce.

In the context of dynamic vehicle routing, we mention the work of Bertsimas and Simchi-Levi (1996), Bertsimas and Ryzin (1991) and Bertsimas and Ryzin (1993a) where a probabilistic analysis of the vehicle routing problem with stochastic demand and service time is provided. Malandraki and Dial (1996), Hill and Benton (1992) and Ichoua et al. (2003) tackle the vehicle routing problem where vehicles' travel time depends on the time of the day, and Malandraki and Daskin (1992) considers a time-dependent traveling salesman problem. Time-dependent travel times have been modeled by dividing the planning horizon into a number of zones, where a different

speed is associated with each of these time zones (see Ichoua et al. (2003) and Jabali et al. (2009)). In Van Woensel et al. (2008), traffic congestion is captured using a queuing approach. Malandraki and Dial (1996) and Malandraki and Daskin (1992) models travel time using stepwise function, such that different time zones are assigned different travel times. Fleischmann et al. (2004) emphasized that modeling travel times as such leads to the undesired effect of passing. That is, a later start time might lead to an earlier arrival time. As in Ichoua et al. (2003), we consider travel time functions that adhere to the FIFO principle. Such travel time functions do not allow passing.

While several successful (meta-) heuristics and exact algorithms have been developed to solve the VRPTW, algorithms designed to deal with the TDVRPTW are somewhat limited to (meta-) heuristics. In fact, most of the existing algorithms are based on tabu search (Ichoua et al., 2003; Van Woensel et al., 2008; Jabali et al., 2009; Maden et al., 2010). In Malandraki and Daskin (1992) mixed integer linear formulations for the time-dependent vehicle routing problem are presented and several heuristics based on nearest neighbor and cutting planes are provided. Donati et al. (2008) and Balseiro et al. (2011) propose algorithms based on an ant colony system, and Haghani and Jung (2005) present a genetic algorithm. In Hashimoto et al. (2008) a local search algorithm for the TDVRPTW is developed and a dynamic programming is embedded in the local search to determine the optimal starting for each route. Androutsopoulos and Zografos (2009) consider a multi-criteria routing problem, they propose an approach based on the decomposition of the problem into a sequence of elementary itinerary subproblems that are solved by means of dynamic programming. Malandraki and Dial (1996) present a restricted dynamic programming for the time-dependent traveling salesman problem. In each iteration of the dynamic programming, only a subset with a predefined size and consisting of the best solutions is kept and used to compute solutions in the next iteration. Tang (2008) emphasizes the difficulty of implementing route improvement procedures in case of time-dependent travel times and proposes efficient ways to deal with that issue.

Column generation has been successfully implemented for the VRPTW. For an overview of column generation algorithms, the reader is referred to Lübbecke and Desrosiers (2005). Column generation in the context of the VRPTW was first introduced by Desrochers et al. (1992). Later, Kohl et al. (1999) introduced subtour elimination constraints and 2-path cuts into the column generation approach and Cook and Rich (1999) applied the more general k-path cuts. In the nineties, the pricing problem of choice was the shortest path problem with resource constraints and two cycle elimination, in Irnich and Villeneuve (2006) an algorithm for k-cycle elimination was introduced which led to tighter bounds and, Feillet et al. (2004b) and Chabrier (2006) proposed algorithms for the elementary shortest path problem with

resource constraints (ESPPRC) which further improved lower bounds. Righini and Salani (2006) and Righini and Salani (2008) proposed various techniques to speed up the ESPPRC algorithm, including bi-directional search and partial elementarity. Jespen et al. (2008) further improved lower bounds by proposing a column generation algorithm with valid inequalities based on the master problem variables (up to that paper inequalities had been expressed in the variables of the equivalent compact formulation). To accelerate the pricing problem solution, Desaulniers et al. (2008) proposed a tabu search heuristic for the ESPPRC. Furthermore, elementarity is relaxed for a subset of nodes, and both 2-path and subset-row inequalities are used. Recently, Baldacci et al. (2011) introduce a new route relaxation, called *ng*-route, used to solve the pricing problem. Their framework proves to be very effective in solving difficult instances of the VRPTW with wide time windows, they solve all but one of the 56 famous Solomon instances. It is also worth mentioning the column generation algorithm of Bettinelli et al. (2010) that consider the dispatch time from the depot as a decision variable (as we do) but assumes time-independent travel times.

5.3. Problem Description

Consider a graph $G = (V, A)$ where $V = \{0, 1, ..., N, N+1\}$ is the set of nodes and $V_c = V/\{0, N+1\}$ represents the set of customers while node 0 and $N+1$ represents the depot. Node 0 and $N+1$ will be the start and end, respectively, of any route. $A = \{(i, j) : i \neq j \quad \text{and} \quad i, j \in V\}$ is the set of all arcs between the nodes. Let K be the set of homogeneous vehicles each with a finite capacity Q. Let $[a_i, b_i]$ be the time window, q_i be the demand and s_i be the service time of node $i \in V$. We assume $s_0 = s_{N+1} = q_0 = q_{N+1} = 0$. We denote $\tau_{ij}(t)$ travel time from node i to node j given a departure from node i a time t.

5.3.1 Travel time and arrival time functions

To each arc $a \in A$, we associate a speed profile. The speed profile divides the planning horizon into time zones, each with a constant speed. The resulting stepwise speed function is translated into travel time functions that satisfy the First-In First-Out (FIFO) principle. Figure 5.1 depicts a speed profile and the corresponding travel time function for some arc (i, j). The speed profile shows the expected speed on arc (i, j) at a given point in time, and the travel time function shows the expected time needed for traveling from i to j at a given departure time from node i. We call the points *a, b, c, d* and *e* where speeds change *speed breakpoints*. Speed breakpoints are also breakpoints in the travel time function. The other *travel time breakpoints* are

determined as the start time to arrive exactly at a speed breakpoint (e.g, a' is the start time to exactly arrive at time a) using the procedure as described in Ichoua et al. (2003). The travel time function is piecewise linear and can easily be represented by the coordinates at the breakpoints.

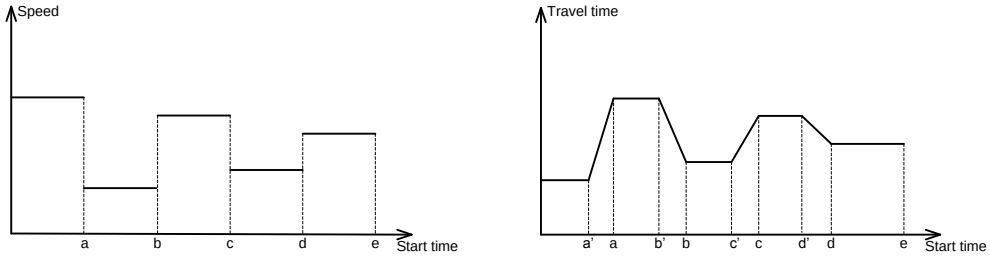

Figure 5.1 Speed and travel time functions.

Given a partial path starting at the depot 0 and ending at some node i, the arrival time at i depends on the dispatch time t at the depot. Due to the FIFO property of the travel time functions, a later dispatch at the depot results in a later arrival at node i. Therefore, if a route is infeasible for some dispatch time t at the depot (i.e., time windows are violated), it will also be infeasible for any dispatch time $t' > t$ at the depot. Moreover, if we define $\delta_i(t)$ as the arrival time function at node i given a dispatch time t at the depot, $\delta_i(t)$ will be non-decreasing in t. Given a partial path $(v_0, v_1, v_2, \ldots, v_k)$ with $v_0 = 0$, we can recursively calculate the arrival time at each node of the path as follows:

$$\delta_{v_i}(t) = \begin{cases} t & \text{if } i = 0 \\ \delta_{v_{i-1}}(t) + \tau_{v_{i-1},v_i}(\delta_{v_{i-1}}(t)) & \text{if } i \in \{1, \ldots, k\} \end{cases} \tag{5.1}$$

where $\delta_{v_0}(t)$ simply represents the arrival time at the depot given a dispatch time t at the same depot. It should be clear that the arrival time function is also a piecewise linear function as each calculation in the second step involves two piecewise linear functions. Figure 5.2 depicts the recursive calculation of the arrival time functions using Equation (5.1). Again, we can completely represent an arrival time function using the *arrival time function breakpoints* resulting from either breakpoints of travel time functions, breakpoints of the arrival time function of the parent node, or from time windows. The duration of the path given a departure time t at the depot can easily be computed as $\delta_{v_k}(t) - t$. The departure time t^* at the depot that results in the shortest duration of the partial path can be calculated as:

$$t^* = \arg\min_{t \in T} \{\delta_{v_k}(t) - t\} \tag{5.2}$$

where T is the set of feasible departure times at the depot for the particular path. Since $\delta_{v_k}(t) - t$ itself is a piecewise linear function it is clear that the minimum duration can be computed by just considering the breakpoints of the arrival time function.

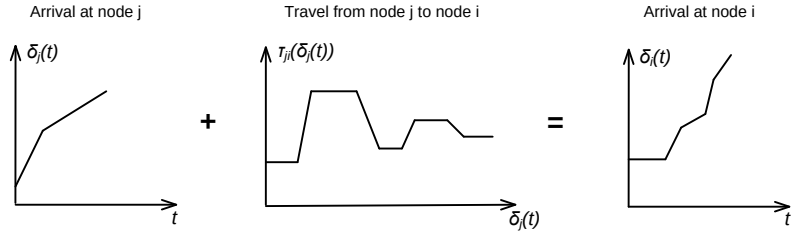

Figure 5.2 Arrival time functions.

To summarize, the TDVRPTW studied in this chapter differs from the classical VRPTW in exactly two ways: 1) instead of using constant travel times between each pair of nodes (i, j) in the graph, we use travel times that depend on the departure time at node i, and 2) instead of considering an objective function that minimizes the sum of arc costs, we minimize the total duration of the routes used in the solution. The choice of the objective function has a significant impact on the exact algorithm proposed. Minimizing the classical VRPTW objective function would have resulted in a much simpler pricing problem as there would be no need to schedule the departure time at the depot. In that case, there would be no need for extra resources in the labeling algorithm and existing VRPTW algorithms based on the set partitioning formulation could easily be modified to solve the problem by updating the mechanism for calculating travel times. It should also be noted that our algorithm could handle an objective function that combines the classical objective function with the duration minimization objective by adding two appropriately weighted terms in the objective function. However, we have not experimented with such an objective function as we believe the pure duration minimization provides clearer results.

5.4. Set Partitioning Formulation and Column Generation

To derive the set partitioning formulation for the TDVRPTW, we define Ω as the set of feasible paths. A feasible path is defined by the sequence of customers visited along it, and the dispatch time at the depot. To each path $p \in \Omega$, we associate the cost c_p which is the duration of p defined as its end time e_p minus its start time s_p. For each path $p \in \Omega$, we let σ_{ip} be a constant that measures the number of times customer i

is visited by the path p. Furthermore, if y_p is a binary variable that takes the value 1 if and only if the path p is included in the solution, the TDVRPTW is formulated as the following set partitioning problem:

$$\min \sum_{p \in \Omega} c_p y_p \tag{5.3}$$

subject to:

$$\sum_{p \in \Omega} \sigma_{ip} y_p = 1 \qquad \forall i \in V_c \tag{5.4}$$

$$y_p \in \{0,1\} \quad \forall p \in \Omega. \tag{5.5}$$

The objective function (5.3) minimizes the duration of the chosen routes. Constraint (5.4) guarantees that each node is visited exactly once. We use column generation to solve the LP-relaxation of (5.3)–(5.5): starting with a small subset $\Omega' \subseteq \Omega$ of variables, we generate additional variables for the master problem (the LP-relaxation of (5.3)–(5.5)) by solving a pricing subproblem that search for variables with negative reduced cost. Let $\pi_i, i \in V_c$ be the dual variables associated with constraints (5.4). The reduced cost of a variable (path) is defined as

$$\bar{c}_p = c_p - \sum_{i \in V_c} \sigma_{ip} \pi_i = e_p - s_p - \sum_{i \in V_c} \sigma_{ip} \pi_i \tag{5.6}$$

Let x_{ij}^p be a binary variable that takes the value 1 if and only if arc (i,j) is traversed along path p, then we can write σ_{ip} as:

$$\sigma_{ip} = \sum_{(i,j) \in \gamma^+(i)} x_{ij}^p \tag{5.7}$$

where $\gamma^+(i)$ is the set of arcs originating from node i. Hence,

$$\bar{c}_p = e_p - s_p - \sum_{i \in V_c} \left(\pi_i \sum_{(i,j) \in \gamma^+(i)} x_{ij}^p \right) \tag{5.8}$$

$$= e_p - s_p - \sum_{(i,j) \in A} \pi_i x_{ij}^p \tag{5.9}$$

with $\pi_0 = \pi_{N+1} = 0$. Our pricing problem is a Time-Dependent Elementary Shortest Path Problem with Resource Constraints (TDESPPRC). We consider resources related to time and elementarity. We do not consider capacity in the pricing problem. Instead, we ensure that each route obeys the capacity constraint using additional constraints in the master problem.

5.4.1 Capacity cuts

We have chosen to handle the capacity constraint using valid inequalities in the master problem instead of handling it directly in the pricing problem. This leads to a slightly simpler pricing problem, but could also lead to a weaker lower bound.

For each arc (i, j) in a two-index formulation of the VRPTW, binary variables x_{ij} are defined that takes value 1 if and only if the arc is traversed in the solution. If we define, for every set $S \subseteq V_c$, $A(S) = \{(i, j) \in A : i \in S, j \in S\}$ and let $r(S) = \lceil \sum_{i \in S} q_i/Q \rceil$ be a lower bound on the number of vehicles needed to serve the customers in the set S. The capacity inequality, well known from the capacitated vehicle routing problem (see e.g., Lysgaard et al. (2004)), can be stated as follows:

$$\sum_{(i,j) \in A(S)} x_{ij} \leq |S| - r(S) \tag{5.10}$$

It is easy to express a solution to the LP-relaxation of (5.3)–(5.5) in the variables x_{ij} on which the capacity cut can be separated, and transform the cut into the variables of the set partitioning formulation. See for example Kohl et al. (1999) or Fukasawa et al. (2006) for details. Consider, for example, k capacity constraints defined by the sets $S_1, \ldots, S_k$ with corresponding dual variables $\lambda_i, i \in \{1, \ldots, k\}$. The objective of the pricing problem becomes

$$\bar{c}_p = e_p - s_p - \sum_{(i,j) \in A} \pi_i x_{ij}^p - \sum_{l=1}^{k} \sum_{(i,j) \in A(S_l)} \lambda_l x_{ij}^p$$

For each arc $(i, j) \in A$, it is possible to aggregate the contributions of the dual variables π and λ into one constant φ_{ij} such that the objective of the pricing problem becomes

$$\bar{c}_p = e_p - s_p - \sum_{(i,j) \in A} \varphi_{ij} x_{ij}^p,$$

which is the form of the pricing problem we are using in our labeling algorithm. We use the code of Lysgaard (2003) to separate the capacity inequalities. Moreover, we manage cuts and column pools as described in Ropke and Cordeau (2009).

5.4.2 Branching

The branch-and-bound tree is explored using a best bound strategy. The algorithm branches on the arc variables x_{ij}. It looks for pairs $(i, j), i, j \in V_c$ such that $x_{ij}^* + x_{ji}^*$ is close to 0.5 (x^* is the current fractional solution expressed in the arc variables) and imposes the branch

$$x_{ij} + x_{ji} \leq \lfloor x_{ij}^* + x_{ji}^* \rfloor \ \vee \ x_{ij} + x_{ji} \geq \lceil x_{ij}^* + x_{ji}^* \rceil$$

If $x^*_{ij} + x^*_{ji}$ is integer for all pairs $(i,j), i,j \in V_c$, then the algorithm looks for an arc $(i,j) \in A$ for which x^*_{ij} is fractional and branches on that instead. The algorithm uses strong branching, that is, the impact of branching on several candidates is investigated each time a branching decision has to be made. For each branch candidate, we estimate the lower bound in the two child nodes by solving the associated LP-relaxation using a quick pricing heuristic. Separation procedures are not invoked when estimating the lower bound of the child nodes. The algorithm performs the branch that maximizes the lower bound in the weakest of the two child nodes. The algorithm considers 15 branch candidates in the first 10 nodes of the branch-and-bound tree, and 10 candidates in the rest.

5.5. The Pricing Problem

In this chapter, we solve the pricing problem by means of a time-dependent labeling (TDL) algorithm which is a modification of the labeling algorithm applied to the elementary shortest path problem with resource constraints. To speed up the labeling algorithm, a bi-directional search is performed in which labels are extended both forward from the start depot (i.e., node 0) to its successors, and backward from the end depot (i.e., node N+1) to its predecessors. While forward labels are extended to some fixed time t_m (e.g., the middle of the planning horizon) but not further, backward labels are extended too, but are allowed to directly cross t_m. At the end, forward and backward labels are merged to construct complete tours. The running time of a labeling algorithm depends on the length of partial paths associated with its labels. A bi-directional search avoids generating long paths and therefore usually limits running times. The reader is referred to Righini and Salani (2006) for more detail on the bi-directional search.

5.5.1 The forward TDL algorithm

In the forward TDL algorithm, labels are extended from the start depot (i.e., node 0) to its successors. The extension to a node is allowed if it is feasible and if the earliest arrival time (including waiting and service time) at that node is no further than t_m. We associate the following components to a forward label L_f:

$v(L_f)$	the last node visited on the partial path represented by L_f
$c(L_f)$	the sum of the dual variables associated with arcs traversed along the partial path represented by L_f
$\delta_{L_f}(t)$	arrival time at $v(L_f)$ through the partial path represented by L_f when service time at $v(L_f)$ the departure time at the depot is t. It includes both waiting time and
$S(L_f)$	set of nodes visited along the partial path represented by L_f

The basic operation in the forward labeling algorithm is the extension of a label L'_f along an arc $(v(L'_f), j)$ to a node j to create a new label L_f. The arrival time function associated with the new label L_f is computed as follows:

$$\delta_{L_f}(t) = \delta_{L'_f}(t) + \tau_{v(L'_f)j}(\delta_{L'_f}(t)) \tag{5.11}$$

which amounts to constructing a new piecewise linear function from two existing piecewise linear functions, furthermore, we have:

$$S(L_f) = S(L'_f) \cup \{j\} \quad \text{and} \quad c(L_f) = c(L'_f) - \varphi_{v(L'_f)j} \tag{5.12}$$

The extension of label L'_f to L_f is feasible if:

$$S(L'_f) \cap \{j\} = \emptyset \text{ and } \delta_{L_f}(0) \leq \min(t_m, b_j + s_j). \tag{5.13}$$

When $v(L_f) = n + 1$ the reduced cost of the path corresponding to L_f is:

$$\overline{c}(L_f) = \min_{t \in T}\{\delta_{L_f}(t) - t\} + c(L_f)$$

where T is the domain definition of function $\delta_{L_f}(t)$. The TDL algorithm is a complete enumeration in which, for every label, all possible extensions are derived and stored. It ends when all labels are processed. However, the number of labels that can be processed could easily be very large. To reduce the number of labels, a dominance test is introduced. To define the dominance test, we let $E(L_f)$ denote the set of all partial paths departing at node $v(L_f)$ at time $\delta_{L_f}(0)$ and reaching node $n+1$ without violating time windows and reusing nodes from $S(L_f)$. If $L \in E(L_f)$, we denote $L_f \oplus L$ as the label resulting from extending L_f by L. In case of the forward TDL algorithm, dominance is defined as follows:

Definition 5.1 Label L^2_f is dominated by label L^1_f if:

1. $v(L^1_f) = v(L^2_f)$
2. $E(L^2_f) \subseteq E(L^1_f)$

3. $\overline{c}(L_f^1 \oplus L) \leq \overline{c}(L_f^2 \oplus L), \forall L \in E(L_f^2)$

Definition 5.1 states that any feasible extension of label L_f^2 is also feasible for label L_f^1. Furthermore, extending L_f^1 should always result in a better route. However, it is not straightforward to verify the conditions of Definition 5.1 as it requires the computation and the evaluation of all feasible extensions of both labels L_f^1 and L_f^2. Therefore, sufficient dominance criteria that are computationally less expensive are desirable. In Proposition 5.1, the sufficient conditions (1) to (5) are introduced. Condition (3) is needed because of the elementarity of paths. Condition (4), in addition to the FIFO assumption, guarantees that time windows of nodes visited along any feasible extension of L_f^2 are respected when reached through L_f^1. In Condition (5), $t(L_f)$ denotes the latest feasible start time at the depot of the partial path represented by label L_f. Condition (5) ensures that no cheaper route can be obtained by extending L_f^2 regardless of the departure time at the depot. Proposition 5.1 is formally stated as follows:

Proposition 5.1 Label L_f^2 is dominated by label L_f^1 if:

1. $v(L_f^1) = v(L_f^2)$
2. $c(L_f^1) \leq c(L_f^2)$
3. $S(L_f^1) \subseteq S(L_f^2)$
4. $\delta_{L_f^1}(t) \leq \delta_{L_f^2}(t), \quad \forall t \in [0, t(L_f^2)]$
5. $t(L_f^2) \leq t(L_f^1)$

Proof: Proof of Proposition 5.1: First we prove that $E(L_f^2) \subseteq E(L_f^1)$.
Let $L \in E(L_f^2)$, then $S(L) \cap S(L_f^2) = \emptyset$. As $S(L_f^1) \subseteq S(L_f^2)$, we should also have:

$$S(L) \cap S(L_f^1) = \emptyset \tag{5.14}$$

Now we will show that customers' time windows along the partial path represented by L are respected when reached trough L_f^1.
Let i be a node visited on the partial path represented by L, and $L_i \subseteq L$ be the partial path with i as the current node and the same start node as L. Furthermore, let $t \leq t(L_f^2)$ be some start time at the depot. It follows from Condition (4) and the FIFO principle that:

$$\begin{aligned} \delta_{L_f^1 \oplus L_i}(t) &= \delta_{L_f^1}(t) + \delta_{L_i}(\delta_{L_f^1}(t)) \\ &\leq \delta_{L_f^2}(t) + \delta_{L_i}(\delta_{L_f^2}(t)) \\ &= \delta_{L_f^2 \oplus L_i}(t) \end{aligned}$$

Since $\delta_{L_f^2 \oplus L_i}(t) \leq b_i$, we have:

$$\delta_{L_f^1 \oplus L_i}(t) \leq b_i \tag{5.15}$$

Equations 5.14 and 5.15 imply that $E(L_f^2) \subseteq E(L_f^1)$.
Now we will show that $\overline{c}(L_f^1 \oplus L) \leq \overline{c}(L_f^2 \oplus L)$
We know that: $c(L_f^1) \leq c(L_f^2)$. Hence,

$$\begin{aligned} c(L_f^1 \oplus L) &= c(L_f^1) + c(L) \\ &\leq c(L_f^2) + c(L) \\ &= c(L_f^2 \oplus L) \end{aligned}$$

We conclude that for all $t \leq t(L_f^2)$:

$$\delta_{L_f^1 \oplus L}(t) - t + c(L_f^1 \oplus L) \leq \delta_{L_f^2 \oplus L}(t) - t + c(L_f^2 \oplus L)$$

Hence, and since $t(L_f^2) \leq t(L_f^1)$,

$$\min_{t \leq t(L_f^1)} \left\{ \delta_{L_f^1 \oplus L}(t) - t \right\} + c(L_f^1 \oplus L) \leq \min_{t \leq t(L_f^2)} \left\{ \delta_{L_f^2 \oplus L}(t) - t \right\} + c(L_f^2 \oplus L)$$

Hence, label L_f^1 dominates label L_f^2. □

Dominance as introduced in Proposition 5.1 is weak and will probably not sufficiently reduce the number of labels processed by the TDL algorithm. In fact, $S(L_f^1) \subseteq S(L_f^2)$ typically implies $c(L_f^1) \geq c(L_f^2)$ (e.g., if no capacity inequalities have been added to the master problem then the value of $c(L_f)$ is determined by the dual variables π which, although unrestricted in sign, typically are positive). Hence, typically conditions (2) and (3) are only both true in case of equality. Furthermore, very cheap labels representing partial paths with a very long duration, not leading to a route in the optimal solution will often not be dominated. In Figure 5.3, the numbers associated with the arcs represent travel times and the numbers associated with the nodes represent dual variables π_i (we disregard dual variables corresponding to capacity inequalities in this example), and both partial paths P_1 and P_2 end at the same node 4. Because of Condition (2), the label representing partial path P_2 will not be dominated by the one representing partial path P_1. However, a path's reduced cost is equal to its duration reduced by the sum of the dual variables corresponding to the nodes visited along that path. Therefore, extending P_1 clearly results in a better final route. Another pitfall of Proposition 5.1 is that cheap labels are not able to dominate more expensive labels with, for some departure time at the depot, a shorter duration. In Figure 5.4, because of Condition (4), the label representing partial path P_2, with cost -100, will not be dominated by the one representing partial path P_1 with cost -3000. The range of dispatch times at the depot, in which partial path P_2

has a shorter duration, has a width of 500 time units. Clearly, for any starting time at the depot in this range, it is possible to find an earlier (but no more than 500 time units earlier) starting time at the depot that results in the same arrival time at the end node for both P_1 and P_2. Leaving the depot earlier might increase P_1's duration. However, given P_1's new start time, its duration will be no more than 500 time units longer than P_2's duration. Therefore, the extension of P_1 will result in a better final route.

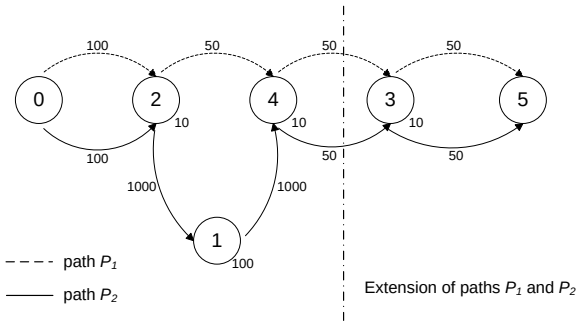

Figure 5.3

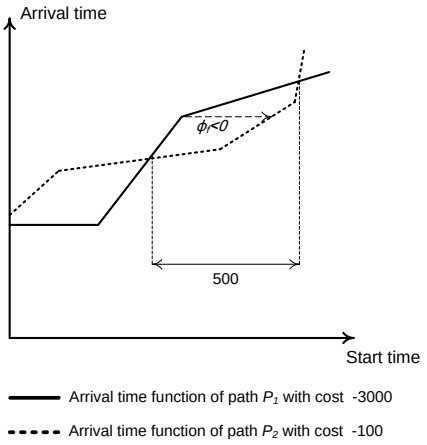

Figure 5.4

In Proposition 5.2, we improve dominance in two directions. First, for every label L_f, we extend $S(L_f)$ to the set $\widetilde{S}(L_f)$ by adding nodes that are unreachable from $v(L_f)$. This technique is well known from e.g., Feillet et al. (2004b). The triangle inequality is not satisfied for time varying travel times as traveling directly to a node is not necessarily the shortest path. Consequently, a node that can not be directly reached

from the end node might be indirectly reached via a diverted route. However, if we define the earliest arrival time as $t_e = \min_{j \in V} \{\delta_{L_f}(0) + \tau_{v(L_b)j}(\delta_{L_f}(0))\}$, any node j with $b_j < t_e$ will be unreachable from the partial path corresponding to L_f. This test can be done quickly, although we might fail to find all unreachable nodes. Second, we relax Condition (2) by adding a quantity ϕ_f to the cost $c(L_f^2)$ of label L_f^2. ϕ_f measures how much the start time of the partial path represented by label L_f^1, can be postponed (in case ϕ_f is positive) or expedited (in case ϕ_f is negative) and still arrive at the end node at the same time as when reaching the end node through the partial path represented by label L_f^2. ϕ_f is illustrated in Figure 5.4. In order to define ϕ_f formally, we need the following definitions: let $\delta_{L_f}^{-1}(t_a)$ be the latest departure time from the depot that guarantees arrival at node $v(L_f)$ at time t_a, that is $\delta_{L_f}^{-1}(t_a) = \max\{t \leq t(L_f) : \delta_{L_f}(t) = t_a\}$. The function $\delta_{L_f}^{-1}(t_a)$ is defined on the domain $A_{\delta_{L_f}^{-1}} = \{t_a \in \mathbb{R} : \exists t \leq t(L_f) : \delta_{L_f}(t) = t_a\}$. With these definitions we can define ϕ_f as:

$$\phi_f = \min\left\{t(L_f^1) - t(L_f^2), \min_{t_a \in \mathcal{T}_a}\left\{\delta_{L_f^1}^{-1}(t_a) - \delta_{L_f^2}^{-1}(t_a)\right\}\right\} \quad \text{and} \quad \mathcal{T}_a = A_{\delta_{L_f^1}^{-1}} \cap A_{\delta_{L_f^2}^{-1}}$$

Proposition 5.2 is stated as follows:

Proposition 5.2 Label L_f^2 is dominated by label L_f^1 if:

1. $v(L_f^1) = v(L_f^2)$
2. $c(L_f^1) \leq c(L_f^2) + \phi_f$
3. $S(L_f^1) \subseteq \widetilde{S}(L_f^2)$
4. $\delta_{L_f^1}(0) \leq \delta_{L_f^2}(0)$

Intuitively, ϕ_f measures the difference in the durations of the partial paths represented by labels L_f^1 and L_f^2, when arrival times at $v(L_f^1)$ and $v(L_f^1)$ are equal. Therefore, the dominance test in Proposition 5.2 compares labels in a more efficient way (i.e the cost of a partial path is its duration reduced by the sum of the dual variables). Moreover, time windows are handled correctly.

Proof: Proof of Proposition 5.2: Similarly to Proposition 5.1, and by using the fact that $\delta_{L_f^1}(0) \leq \delta_{L_f^2}(0)$ and $S(L_f^1) \subseteq \widetilde{S}(L_f^2)$, we can prove that any feasible extension to L_f^2 is also feasible for L_f^1.
Let $L \in E(L_f^2)$, and $t \leq t(L_f^2)$ be some start time at the depot.

Now, let t^* be such that:

$$t^* = \begin{cases} \delta^{-1}_{L^1_f}(\delta_{L^2_f}(t)) - t & \text{if} \quad \delta_{L^2_f}(t) \in A_{\delta^{-1}_{L^1_f}} \\ t(L^1_f) - t(L^2_f) & \text{otherwise} \end{cases}$$

t^* is illustrated in Figure 5.5, and can also be written as:

$$t^* = \begin{cases} \delta^{-1}_{L^1_f}(\delta_{L^2_f}(t)) - \delta^{-1}_{L^2_f}(\delta_{L^2_f}(t)) & \text{if} \quad \delta_{L^2_f}(t) \in A_{\delta^{-1}_{L^1_f}} \\ t(L^1_f) - t(L^2_f) & \text{otherwise} \end{cases}$$

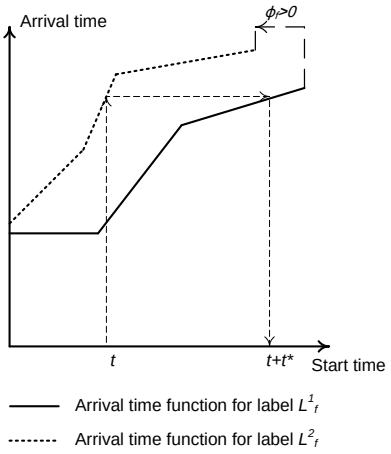

Figure 5.5

Postponing the start time of L^1_f at the depot by t^* (i.e., the start time at the depot is $t + t^*$ instead of t) results in an arrival time at the current node that is equal to the arrival time at the same current node reached through L^2_f, and when the start time at the depot is t . Furthermore, $t + t^* \leq t(L^1_f)$. Therefore:

$$\delta_{L^2_f}(t) = \delta_{L^1_f}(t + t^*)$$

Consequently:

$$\begin{aligned} \delta_{L^2_f \oplus L}(t) &= \delta_{L^2_f}(t) + \delta_L(\delta_{L^2_f}(t)) \\ &= \delta_{L^1_f}(t + t^*) + \delta_L(\delta_{L^1_f}(t + t^*)) \\ &= \delta_{L^1_f \oplus L}(t + t^*) \end{aligned}$$

Now we will show that $\overline{c}(L_f^1 \oplus L) = \overline{c}(L_f^2 \oplus L)$
Obviously $\phi_f \leq t^*$, Hence,

$$\begin{aligned} \delta_{L_f^1 \oplus L}(t + t^*) - (t + t^*) &= \delta_{L_f^2 \oplus L}(t) - t - t^* \\ &\leq \delta_{L_f^2 \oplus L}(t) - t - \phi_f \end{aligned}$$

Furthermore, we know that: $\phi_f \geq c(L_f^1) - c(L_f^2)$.
Hence,

$$\delta_{L_f^1 \oplus L}(t + t^*) - (t + t^*) + c(L_f^1 \oplus L) \leq \delta_{L_f^2 \oplus L}(t) - t + c(L_f^2 \oplus L)$$

We conclude that for every $t \leq t(L_f^2)$, there exists $\tilde{t} = t + t^* \leq t(L_f^1)$ such that:

$$\delta_{L_f^1 \oplus L}(\tilde{t}) - (\tilde{t}) + c(L_f^1 \oplus L) \leq \delta_{L_f^2 \oplus L}(t) - t + c(L_f^2 \oplus L)$$

Hence,

$$\min_{t \leq t(L_f^1)} \left\{ \delta_{L_f^1 \oplus L}(t) - t \right\} + c(L_f^1 \oplus L) \leq \min_{t \leq t(L_f^2)} \left\{ \delta_{L_f^2 \oplus L}(t) - t \right\} + c(L_f^2 \oplus L)$$

Hence, label L_f^1 dominates label L_f^2. □

5.5.2 The backward TDL algorithm

In the backward TDL algorithm, labels are extended from the depot (i.e., node $n+1$) to its predecessors. The extension of a label is allowed if it is feasible and if the latest possible departure time at the end node is larger than t_m. To a label L_b, we associate the following components:

$v(L_b)$	the first node visited on the partial path represented by L_b
$c(L_b)$	the sum of the dual variables associated with arcs traversed along the partial path represented by L_b
$\delta_{L_b}(t)$	arrival time at the depot through the partial path represented by L_b and when leaving node $v(L_b)$ at time t
$S(L_b)$	set of nodes visited along the partial path represented by L_b

Let $t(L_b)$ be the latest feasible start time of the arrival time function $\delta_{L_b}(t)$. The set of feasible extensions $E(L_b)$ of L_b is the set of partial paths such that when departing at the depot (i.e., node 0) at time 0, reach node $v(L_b)$ at some time $t \leq t(L_b)$ (t includes waiting and service at $v(L_b)$) without violating time windows. The basic operation in the backward labeling algorithm is the extension of a label L'_f along an arc $(j, v(L'_b))$

to a node j to create a new label L_b. The arrival time function associated with the new label L_b is computed as follows:

$$\delta_{L_b}(t) = \delta_{L'_b}(t + \tau_{jv(L'_b)}(t)) \tag{5.16}$$

Furthermore, we have:

$$S(L_b) = S(L'_b) \cup \{j\} \quad \text{and} \quad c(L_b) = c(L'_b) - \varphi_{jv(L'_b)} \tag{5.17}$$

The latest departure time $t(L_b)$ at node j, such that the arrival at node $v(L'_b)$ is exactly its latest possible departure time $t(L'_b)$, can be calculated using the procedure as described in Ichoua et al. (2003).

The extension of L'_b with node j is feasible if:

$$S(L'_b) \cap \{j\} = \emptyset, \quad t(L_b) \geq a_j + s_j \quad \text{and} \quad t(L'_b) \geq t_m \tag{5.18}$$

Again, as illustrated in Figure 5.6, arrival time functions are non-decreasing linear stepwise functions. Moreover, arrival time functions are completely defined by their breakpoints. Arrival time function breakpoints result from travel time functions breakpoints, breakpoints calculated as departure time at the start node to hit a breakpoint of the arrival time function of the destination node, or from time windows. Furthermore, dominance can be defined in the same way as in the case of the forward TDL algorithm. To avoid redundancy, we only present the improved dominance criteria.

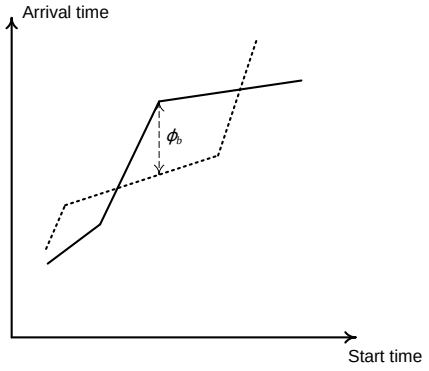

Figure 5.6

In Proposition 5.3, $\widetilde{S}(L_b)$ denotes the set of visited nodes along the partial path represented by label L_b extended by nodes that are unreachable from $v(L_b)$. In fact,

if we define the latest departure time such that arrival time at $v(L_b)$ is its latest possible start time $t(L_b)$ as $t_l = \max_{j \in V} \left\{ \max \left\{ t : t + \tau_{jv(L_b)}(t) = t(L_b) \right\} \right\}$, any node j with $a_j + s_j > t_l$ will be unreachable from the partial path corresponding to L_b. Furthermore, we relax Condition (2) by adding a quantity ϕ_b to the cost $c(L_b^2)$. ϕ_b is a real number related to, given a departure time at node $v(L_b^1)$, how early (in case ϕ_b is negative) or late (in case ϕ_b is positive) arrival at the depot takes place when traversing the partial path represented by label L_b^1 instead of the partial path represented by label L_b^2. Note that ϕ_b is conceptually different from ϕ_f as it is related to arrival time at the end node (i.e., the depot) instead of departure time at the depot. In the forward search, we can not relate ϕ_f to the arrival time at the end node as this might be different from the depot. Therefore, any gains in terms of arrival time do not guarantee a gain in the final complete tour. In fact, gains can easily be lost by possible waiting time due to time windows. If we define ϕ_b as:

$$\phi_b = \min \left\{ \delta_{L_b^2}(t(L_b^2)) - \delta_{L_b^1}(t(L_b^1)), \min_{t_d \leq t(L_b^2)} \left\{ \delta_{L_b^2}(t_d) - \delta_{L_b^1}(t_d) \right\} \right\}$$

we state Proposition 5.3 as follows:

Proposition 5.3 Label L_b^2 is dominated by label L_b^1 if:

1. $v(L_b^1) = v(L_b^2)$
2. $c(L_b^1) \leq c(L_b^2) + \phi_b$
3. $S(L_b^1) \subseteq \widetilde{S}(L_b^2)$
4. $t(L_b^1) \geq t(L_b^2)$

Proof: Proof of Proposition 5.3: Similarly to Proposition 5.2, and by using the fact that $t(L_b^1) \geq t(L_b^2)$ and $S(L_b^1) \subseteq \widetilde{S}(L_b^2)$, we can prove that any feasible extension to L_b^2 is also feasible for L_b^1.
Let $L \in E(L_b^2)$, and t be a feasible start time for L at the depot such that $\delta_L(t) \leq t(L_b^2)$.
Now, for every $t_d \leq t(L_b^2)$, let $\Delta(t_d)$ be such that:

$$\Delta(t_d) = \min \left\{ \delta_{L_b^2}(t(L_b^2)) - \delta_{L_b^1}(t(L_b^1)), \delta_{L_b^2}(t_d) - \delta_{L_b^1}(t_d) \right\}$$

Obviously $\phi_b \leq \Delta(\delta_L(t))$, Hence,

$$\begin{aligned} \delta_{L_b^2 \oplus L}(t) - \delta_{L_b^1 \oplus L}(t) &= \delta_{L_b^2}(\delta_L(t)) - \delta_{L_b^1}(\delta_L(t)) \\ &\geq \Delta(\delta_L(t)) \\ &\geq \phi_b \end{aligned}$$

Furthermore, we know that: $\phi_b \geq c(L_b^1) - c(L_b^2)$.
Hence,

$$\delta_{L_b^1 \oplus L}(t) - t + c(L_b^1 \oplus L) \leq \delta_{L_b^2 \oplus L}(t) - t + c(L_b^2 \oplus L)$$

We conclude that:

$$\min_{t \in D_{L_b^1 \oplus L}} \left\{ \delta_{L_b^1 \oplus L}(t) - t \right\} + c(L_b^1 \oplus L) \leq \min_{t \in D_{L_b^2 \oplus L}} \left\{ \delta_{L_b^2 \oplus L}(t) - t \right\} + c(L_b^2 \oplus L)$$

Hence, label L_b^1 dominates label L_b^2. □

5.5.3 Merging forward and backward labels

When all forward and backward labels are processed, they are joined to construct feasible tours with negative reduced cost. A forward label L_f and a backward label L_b are joined if $L_b \in E(L_f)$. The resulting label $L = L_f \oplus L_b$ has the following attributes:

- $v(L) = n + 1$
- $c(L) = c(L_f) + c(L_b)$
- $S(L) = S(L_f) \cup S(L_b)$
- $\delta_L(t) = \delta_{L_b}(\delta_{L_f}(t))$, for all $t \in D_{\delta_{L_f}}$ such that $\delta_{L_f}(t) \in D_{\delta_{L_b}}$

The bi-directional TDL algorithm guarantees the generation of all paths with negative reduced cost. Without loss of generality, let $P = v_0 \rightarrow ... \rightarrow v_p$ be a path in the optimal solution. We formally prove the correctness of the bi-directional TDL algorithm in the following proposition:

Proposition 5.4 Let v_i be a node in P. P can be found as $P = P_f \oplus P_b$ where $P_f = v_0 \rightarrow ... \rightarrow v_i$ is generated by the forward TDL algorithm and $P_b = v_i \rightarrow ... \rightarrow v_p$ is generated by the backward TDL algorithm.

Proof: Proof of Proposition 5.4: Let P be a path in the optimal solution and v_i a node visited along it. We need to prove that path $P_f = v_0 \rightarrow ... \rightarrow v_i$ is generated by the forward TDL algorithm and $P_b = v_i \rightarrow ... \rightarrow v_p$ is generated by the backward

TDL algorithm.
We will prove this by contradiction. Assume that path P_f is not generated by the TDL algorithm. Let L_f be the label corresponding to path P_f. We know that $\delta_{L_f}(0) \leq t_m$. Moreover, path P is feasible and P_f is a partial path of P. Therefore, P_f is also feasible. The only remaining reason for why P_f is not generated is that L_f is removed after domination. Let L_f^* be the label that dominates L_f, and P_f^* the corresponding partial path.
$P_f^* \oplus P_b$ is a better final path than P, which contradicts the optimality of P. Therefore, P_f should be generated by the TDL algorithm.
Similarly, we can prove that P_b should be generated by the backward TDL algorithm.
□

When we solve the pricing problem, we need to generate many different columns with negative reduced cost and not only the one with the least reduced cost. Obviously, the bi-directional TDL algorithm can generate duplicate columns. In fact, a path can be spliced at different nodes. However, any node that can be defined uniquely makes sure that a path spliced at that node is found only once.

Let $P = v_0 \rightarrow v_1 \rightarrow v_2 \rightarrow ... \rightarrow v_i \rightarrow v_{i+1} \rightarrow ... \rightarrow v_{p-1} \rightarrow v_p$ be an arbitrary path where v_0 is the start depot and v_p is the end depot. Moreover, let L be the label representing P. For every $j \geq 0$, let L_j be the label representing the partial path $P_j = 0 \rightarrow v_1 \rightarrow ... \rightarrow v_j$. Furthermore, let $v_i, v_{i+1} \in S(L)$ such that node v_{i+1} is visited directly after node v_i. We define a *splicing node* of path P as follows:

Definition 5.2 Node v_i is a splicing node of path P if $\delta_{L_i}(0) \leq t_m$, and

- $\delta_{L_{i+1}}(0) > t_m$, or
- $\delta_{L_{i+1}}(0) \leq t_m$ and $v_{i+1} = N + 1$

Proposition 5.5 The splicing node of P exists and is unique.

Proof: Proof of Proposition 5.5: First, we prove the existence of the splicing node. We distinguish between two cases:

Case 1: $\forall v_j \in S(L)$ such that $j > 0$, it holds that $\delta_{L_j}(0) > t_m$.
In this case v_0 (the start depot) is the splicing node of P. In fact, it holds that:

$\delta_{L_0}(0) = 0 \leq t_m$ and $\delta_{L_1}(0) > t_m$.

Case 2: $\exists v_j \in S(L)$ for some $j > 0$ such that $\delta_{L_j}(0) \leq t_m$. We again distinguish between two cases:

Case 2.1: $\forall v_k \in S(L)$ such that $k > j$, it holds that $\delta_{L_k}(0) \leq t_m$.
In this case v_{p-1} is a splicing node. In fact, it holds that: $\delta_{L_{p-1}}(0) \leq t_m$, and $\delta_{L_p}(0) \leq t_m$ and $v_p = N + 1$

Case 2.2: $\exists v_k \in S(L)$ for some $k > j$ such that $\delta_{L_k}(0) > t_m$.
In this case there exists a node v_l, $j \leq l \leq k - 1$, such that: $\delta_{L_l}(0) = 0 \leq t_m$ and $\delta_{L_{l+1}}(0) > t_m$.
Hence, we proved the existence of a splicing node of P

Now, we prove the uniqueness of the splicing node of P. We will prove this by contradiction. Let v_i be a splicing node of P. Let's assume there exists another splicing node $v_s \neq v_i$. v_s is either visited before v_i or after it.
In case v_s is visited after v_i (and $v_s \neq N + 1$), $\delta_{L_s}(0) \geq \delta_{L_{i+1}}(0) > t_m$. This contradicts the definition of a splicing node. L_s is the label representing the partial path $P_s = v_0 \rightarrow ... \rightarrow v_i \rightarrow v_{i+1} \rightarrow ... \rightarrow v_s$.
In case v_s is visited before v_i, $\delta_{L_s}(0) < \delta_{L_{i+1}}(0) \leq t_m$. This again contradicts the definition of a splicing node.
Hence, we proved the uniqueness of the splicing node. □

P is an arbitrary path. Therefore, any path with negative reduced cost is uniquely spliced at its splicing node.

5.5.4 The pricing problem heuristics

Branch-and-price algorithms can be accelerated using heuristics to solve the pricing problem. In fact, the heuristic will search for easy to find paths with negative reduced cost and add them to the master problem. When the heuristics fail to find any more paths with negative reduced cost, the exact algorithm is called. Ideally, for every node in the branching tree, the exact algorithm is called only once to check that no more paths with negative reduced cost exist. In our BCP framework, we use two heuristics. First, a greedy heuristic that extends each label to the node with the smallest travel time. Second, a truncated labeling heuristic in which only a limited number of labels

is stored. Moreover, for the truncated heuristic, relaxed dominance criteria are used. In fact, we relax the condition on the sets of visited customers. Furthermore, we dominate label L_2 by label L_1 if:

$$\min_{t \in D_{\delta_{L_1}}} \{\delta_{L_1}(t) - t\} \leq \min_{t \in D_{\delta_{L_2}}} \{\delta_{L_2}(t) - t\} \tag{5.19}$$

and

$$\min_{t \in D_{\delta_{L_1}}} \{\delta_{L_1}(t) - t\} + c(L_1) \leq \min_{t \in D_{\delta_{L_1}}} \{\delta_{L_2}(t) - t\} + c(L_2) \tag{5.20}$$

The number of stored labels can be increased each time the heuristic fails to find paths with negative reduced cost (e.g., we start with 250, then we increase the number of labels to 500 labels, and finally to 1000 labels).

5.5.5 The TDSPPRC as the pricing problem

Relaxing the TDESPPRC by allowing non-elementary paths results in the Time-Dependent Shortest Path Problem with Resources Constraints (TDSPPRC) as the pricing problem. The TDSPPRC can be solved by the same bi-directional TDL algorithm, where in each label L the attributes $v(L)$, $c(L)$ and $\delta_L(t)$ are stored. $v(L)$ is the last node (in case of the forward search) or the first node (in case of the backward search) visited along L. $c(L)$ is the sum of the dual variables corresponding to the arcs traversed along L. $\delta_L(t)$ is the arrival time at the end node when leaving the start node at time t. Note that the attribute $S(L)$ is dropped as it is no more needed, because paths' elementarity is relaxed. Obviously, the number of paths with negative reduced cost that should be searched for increases. However, the dominance tests for both the forward and the backward becomes much more stronger as Condition (3) is relaxed in Propositions 5.2 and 5.3, resulting in an easier pricing problem. Furthermore, the TDSPPRC results in worse lower bounds obtained from solving the LP-relaxation of the master problem.

5.6. Computational Results

The BCP algorithm is implemented on a Intel(R) Core(TM)2 CPU, 2.13 GHz, 3 GB of RAM computer. We set 72 hours as the limit for computation times. The LP solver CLP from the open source framework COIN (COIN CLP, 2011) is used to solve the linear programming relaxation of the master problem. For our numerical study, we use the well known Solomon's data sets (Solomon, 1987) that follow a naming convention of $DTm.N$. D is the geographic distribution of the customers which can be R (Random), C (Clustered) or RC (Randomly Clustered). T is the instance type

which can be either 1 (instances with tight time windows) or 2 (instances with wide time windows). m denotes the number of the instance and N the number of customers that need to be served. Road congestion is taken into account by assuming that vehicles travel through the network using different speed profiles. We consider speed profiles that comply with data from real life, with two congested periods. Speeds in the rest of the planning horizon (i.e., the depot's time window) are relatively high. Furthermore, we assume three types of links: fast, normal and slow. Slow links might represent links within the city center, fast links might represent highways and normal links might represent the transition from highways to city centers. Moreover, without loss of generality, we assume that breakpoints are the same for all speed profiles as congestion tends to happen around the same time regardless of the link's type (e.g., rush hours).The link type is chosen randomly and remains the same for all instances. The following speed profiles are considered:

	Zone1	**Zone2**	**Zone3**	**Zone4**	**Zone5**
Fast	1.5	1	1.67	1.17	1.33
Normal	1.17	0.67	1.33	0.83	1
Slow	1	0.33	0.67	0.5	0.83

Table 5.1 Speed Profiles.

Where $Zone_1 = [0, 0.2T[$, $Zone_2 = [0.2T, 0.3T[$, $Zone_3 = [0.3T, 0.7T[$, $Zone_4 = [0.7T, 0.8T[$ and $Zone_5 = [0.8T, T]$, where the planning horizon $T = b_{n+1}$ is the upper bound of the depot's time window. Figure 5.7 illustrates the travel time function of an arbitrary link from an R instance. Travel time breakpoints are calculated using the procedure as described in Ichoua et al. (2003). The choice of assigning a fast, normal or slow category for each arc was done as in the data set used by Ichoua et al. (2003). The complete data set used in this chapter is available at `www.diku.dk/~sropke`.

5.6.1 TDESPPRC vs. TDSPPRC

In Tables 5.2, 5.3 and 5.4, we report the instances for which we could at least solve the root node. The first column indicates the name of the instances. The columns denoted as "Root LB" show the lower bounds obtained in the root node. Moreover, the columns denoted as "Best LB" and "UB" indicate, respectively, the best lower and upper bounds found all over a branching tree. In the column "Time", we report the time (in seconds) spend to solve an instance, and in the column "Tree" we report the size of the branching trees. Most of the reported instances are solved to optimality and even when the instance is not solved to optimality we often find a good upper bound.

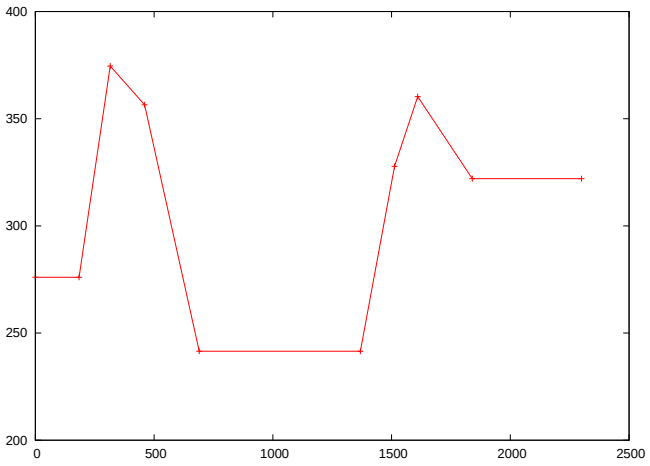

Figure 5.7 Travel time function for an R instance.

However, especially when dealing with 50 and 100 customers, there are many instances for which we fail to solve even the root node, simply because the pricing problem is too difficult. We report results for both the TDESPPRC and the TDSPPRC as the pricing problem. We observe, that the TDSPPRC performs well in case of instances with tight time windows. However, the lower bounds obtained in the root node are worse, which explains the large search trees. In general, as illustrated in the column "Nb. Instances" of Table 5.5, the TDESPPRC is able to solve more instances than the TDSPPRC. The columns "Avg. Root LB" and "Avg. Best LB" of Table 5.5 show, respectively, the average of the root lower bound and the average of the best lower bound of the instances for which both TDESPPRC and TDSPPRC are able to produce a lower bound. Furthermore, the average computation time (in seconds) over the instances that are solved to optimality by both TDESPPRC and TDSPPRC, and the average tree are reported in the columns "Avg. Times" and "Avg. Tree" respectively. Totally, we can solve about 70% of the instance with 25 customers, 48% of the instances with 50 customers and 18% of the instances with 100 customers. For instances with 25 customers, we solved all the ones with tight time windows (i.e., type 1) to proven optimality with the exception of instance C104.25 which is solved with a gap less than 1%. Moreover, for instance with 25 customers, we can also solve many instance of type 2 (with wide time windows). For instances with 50 and 100 customers, most of the solved instances are of type 1. Instances that are not solved to optimality are marked in bold.

	with TDESPPRC					with TDSPPRC				
Instance	Root LB	Best LB	UB	Time(s)	Tree	Root LB	Best LB	UB	Time(s)	Tree
R101	9010.5	9010.5	9010.5	0.1	0	9010.5	9010.5	9010.5	0.2	0
R102	7464.2	7464.2	7464.2	0.6	0	7464.2	7464.2	7464.2	1.2	0
R103	6522.5	6526.1	6526.1	10.0	2	6522.5	6526.1	6526.1	9.4	2
R104	5883.7	5910.6	5910.6	66.0	4	5841.2	5910.6	5910.6	76.2	12
R105	7190.5	7190.5	7190.5	0.2	0	7157.0	7190.5	7190.5	2.2	6
R106	6512.4	6512.4	6512.4	1.4	0	6470.4	6512.4	6512.4	5.4	2
R107	5963.3	5963.3	5963.3	5.0	0	5890.1	5963.3	5963.3	35.1	4
R108	5641.7	5651.8	5651.8	56.9	2	5593.5	5651.8	5651.8	116.2	8
R109	6105.6	6105.6	6105.6	0.9	0	6068.5	6105.6	6105.6	7.5	4
R110	5909.5	5909.5	5909.5	1.3	0	5849.3	5909.5	5909.5	51.7	18
R111	6000.7	6028.9	6028.9	23.5	8	5936.8	6028.9	6028.9	148.8	58
R112	5541.6	5602.5	5602.5	170.2	26	5446.5	5602.5	5602.5	1237.0	308
C101	24684.2	24684.2	24684.2	0.5	0	24684.2	24684.2	24684.2	0.2	0
C102	24472.8	24504.9	24504.9	140.3	6	24470.8	24504.9	24504.9	956.4	6
C103	24350.5	24370.9	24370.9	57775.3	6	24304.9	24370.9	24370.9	22226.4	40
C104	-	-	-	-	-	**24124.0**	**24222.5**	**24224.0**	-	**392**
C105	24663.7	24672.5	24672.5	12.7	4	24663.7	24672.5	24672.5	11.3	4
C106	24684.2	24684.2	24684.2	0.7	0	24684.2	24684.2	24684.2	0.3	0
C107	24412.7	24499.8	24499.8	63.2	10	24412.7	24499.8	24499.8	38.7	12
C108	24303.8	24392.6	24392.6	356.1	36	24282.2	24392.6	24392.6	1789.8	298
C109	24231.7	24312.6	24312.6	12791.7	172	24179.1	24312.6	24312.6	39390.3	1130
RC101	6720.4	7004.3	7004.3	8.4	14	6702.9	7004.3	7004.3	20.4	28
RC102	5926.3	6162.2	6162.2	27.2	18	5836.5	6162.2	6162.2	281.0	44
RC103	5422.4	5422.4	5422.4	7.7	0	5271.9	5422.4	5422.4	1143.5	124
RC104	5174.9	5174.9	5174.9	23.5	0	4960.5	5174.9	5174.9	10216.8	544
RC105	6781.2	6810.7	6810.7	3.2	2	6756.1	6810.7	6810.7	13.3	6
RC106	5590.5	5590.5	5590.5	0.9	0	5477.9	5590.5	5590.5	95.1	36
RC107	5005.4	5005.4	5005.4	3.7	0	4963.9	5005.4	5005.4	154.1	18
RC108	4893.0	4893.0	4893.0	24.1	0	4819.1	4893.0	4893.0	1357.1	58
R201	7881.1	7907.2	7907.2	18.0	2	7774.7	7907.2	7907.2	63.6	14
R202	7091.7	7091.7	7091.7	6151.9	0	**6468.5**	**6541.5**	-	-	**8**
R205	6471.8	6471.8	6471.8	19109.5	0	**6114.0**	**6165.0**	-	-	**6**
R209	**5505.7**	**5526.0**	**5531.4**	-	4	-	-	-	-	-
C201	25581.0	25581.0	25581.0	1.5	0	25581.0	25581.0	25581.0	0.3	0
C202	24728.5	24728.5	24728.5	41213.6	0	-	-	-	-	-
C205	25056.3	25056.3	25056.3	106.0	0	25021.2	25056.3	25056.3	2388.0	12
C206	24928.5	24928.5	24928.5	1926.3	0	24882.2	24928.5	24928.5	210379.0	10
C208	24747.6	24747.6	24747.6	9869.3	0	**24668.8**	**24668.8**	-	-	**2**
RC201	8291.78	8350.4	8350.5	712.4	2	7751.3	8350.5	8350.5	4759.1	514
RC202	7409.0	7409.0	7409.0	71371.8	0	-	-	-	-	-
RC205	7602.4	7602.4	7602.4	1493.3	0	**6754.6**	-	-	-	**14**

Table 5.2 Instances with 25 customers.

5.6.2 Bi-directional TDL vs. mono-directional TDL

In Table 5.6, we illustrate the gains of using a bi-directional search over the mono-directional search. We report the best found lower and upper bounds in the columns "Best LB" and "UB" respectively. Moreover, in the column "Time" we report the time (in seconds) needed to solve an instance. Clearly, the performance of the bi-directional TDL algorithm is far better that of the mono-directional version. The mono-directional hardly solves any instance with 50 customers or instance with 25 customers and wide time windows. This is mainly due to the fact that the number of labels that have to be processed in the bi-directional TDL algorithm is considerably restricted compared to the mono-directional TDL algorithm.

	with TDESPPRC					with TDSPPRC				
Instance	Root LB	Best LB	UB	Time(s)	Tree	Root LB	Best LB	UB	Time(s)	Tree
R101	16027.5	16027.5	16027.5	0.7	0	16027.5	16027.5	16027.5	1.3	0
R102	13400.0	13400.0	13400.0	17.3	0	13400.0	13400.0	13400.0	13.4	0
R103	11616.7	11639.3	11639.3	2172.7	4	11603.5	11639.3	11639.3	520.1	10
R104	**10159.9**	**10206.6**	-	-	**18**	10111.3	10224.3	10224.3	62313.9	448
R105	12749.3	12755.5	12755.5	20.0	2	12740.9	12755.5	12755.5	32.1	4
R106	11715.4	11735.9	11735.9	108.7	2	11668.9	11735.9	11735.9	307.3	12
R107	10926.9	10972.5	10972.5	5878.0	12	10787.9	10972.5	10972.5	11501.0	316
R108	**9863.2**	**9932.3**	-	-	**18**	**9768.0**	**9936.3**	-	-	**1956**
R109	11246.2	11252.7	11252.7	26.4	2	11025.6	11252.7	11252.7	2769.3	594
R110	10617.6	10703.6	10703.6	1097.1	44	10519.6	10703.6	10703.6	21299.9	2116
R111	10691.4	10755.5	10755.5	2138.1	16	10545.1	10755.5	10755.5	38777.3	2532
R112	10030.6	10118.0	10118.0	50790.1	138	**9865.5**	**10085.7**	-	-	**10934**
C101	49062.7	49192.9	49192.9	173.4	6	49062.7	49192.9	49192.9	62.8	6
C102	48360.9	48495.7	48495.7	172486.0	26	48319.2	48495.7	48495.7	12377.3	34
C103	-	-	-	-	-	48151.2	48201.3	48201.3	134496.0	42
C105	48626.0	48759.1	48759.1	900.5	8	48626.0	48759.1	48759.1	124.7	8
C106	48511.6	48629.4	48629.4	323.6	8	48511.6	48629.4	48629.4	171.3	12
C107	48298.9	48432.5	48432.5	7255.7	16	48298.9	48432.5	48432.5	345.1	16
C108	48096.5	48169.2	48169.2	9632.0	6	48077.3	48169.2	48169.2	1686.2	26
C109	**48052.3**	**48120.9**	-	-	**46**	**48004.7**	**48112.7**	**48124.4**	-	**854**
RC101	13269.6	13877.1	13877.1	1486.4	646	13190.9	13877.1	13877.1	6595.2	3080
RC102	11880.3	12389.5	12389.5	14546.2	2082	**11660.7**	**12304.0**	**12480.5**	-	**32052**
RC103	10841.2	11275.4	11275.4	46533.9	1092	**10500.0**	**10966.3**	-	-	**11040**
RC104	9792.8	9792.8	9792.8	2515.0	0	**9530.0**	**9651.9**	-	-	**790**
RC105	12531.9	12939.8	12939.8	1420.2	254	12373.2	12939.8	12939.8	11763.6	2272
RC106	11253.2	11706.6	11706.6	73613.2	14106	**10985.9**	**11348.6**	-	-	**23262**
RC107	10253.9	10564.0	10564.0	4204.9	138	**10074.3**	**10408.8**	-	-	**18384**
RC108	**9719.2**	**9940.6**	**10281.7**	-	**1272**	**9593.8**	**9731.7**	-	-	**6548**
R201	-	-	-	-	-	**13537.5**	**13908.9**	-	-	**366**
C201	-	-	-	-	-	49633.9	49633.9	49633.9	22.3	0

Table 5.3 Instances with 50 customers.

5.6.3 Number of routes vs. number of vehicles

When travel times are time-dependent, and the objective is minimizing the total travelled time instead of the total travelled distance, vehicles' scheduling (i.e., optimizing vehicles' dispatch times at the depot) becomes crucial. Table 5.7 shows the optimal solution for the instance RC201.25. The solution includes 4 routes for which both customers' sequence as well as dispatch times at the depot are reported. Furthermore, we observe that some routes do not overlap in the time dimension, such routes can be performed by the same vehicle. Therefore, the routes in Table 5.7 can be performed using only two vehicles.

5.7. Conclusions

In real-life, vehicles operate in a dynamic environment. In fact, traffic networks are subject to congestion, and therefore travel times are time-dependent. In this chapter, we capture predictable traffic congestion (e.g., congestion due to traffic density) by assuming that vehicles travel at different speeds throughout the planning horizon. In our context, this leads to the Time-Dependent Vehicle Routing Problem with Time Windows (TDVRPTW). When dealing with the TDVRPTW a number of

	with TDESPPRC					with TDSPPRC				
Instance	Root LB	Best LB	UB	Time(s)	Tree	Root LB	Best LB	UB	Time(s)	Tree
R101	26737.8	26737.8	26737.8	12.2	0	26737.8	26737.8	26737.8	26723.0	0
R102	23213.7	23256.2	23256.2	75493.2	8	23201.9	23256.2	23256.2	5845.6	16
R103	-	-	-	-	-	20322.5	20420.1	-	-	474
R105	21570.0	21702.2	21702.2	14031.1	190	21559.1	21702.2	21702.2	9097.6	404
R106	**20117.0**	**20160.0**	-	-	**10**	**20002.6**	**20192.1**	-	-	**1226**
R107	-	-	-	-	-	**18244.0**	**18377.5**	-	-	**216**
R109	-	-	-	-	-	**18777.7**	**18980.0**	-	-	**5348**
R110	**18337.1**	**18385.2**	-	-	**30**	-	-	-	-	-
C101	97801.1	97801.1	97801.1	2422.0	0	97801.1	97801.1	97801.1	70825.0	0
C102	-	-	-	-	-	97532.6	97562.6	97562.6	157656.0	10
C105	97564.8	97729.2	97729.2	50577.8	8	97564.8	97729.2	97729.2	1247.1	8
C106	97463.3	97592.5	97592.5	118794.0	18	97421.5	97592.5	97592.5	14925.1	90
C107	97326.5	97369.1	97369.1	258487.0	10	97326.5	97369.1	97369.1	1485.6	8
RC101	**24229.8**	**24645.9**	**24652.8**	-	**7472**	**24108.4**	**24535.9**	-	-	**5062**
RC102	**21907.9**	**22161.1**	-	-	**808**	**21734.1**	**21987.3**	-	-	**1470**
RC103	-	-	-	-	-	**19514.5**	**19673.9**	-	-	**184**
RC104	-	-	-	-	-	**18422.1**	**18463.5**	-	-	**8**
RC105	22887.1	23207.4	23207.4	63631.0	972	**22617.9**	**22990.8**	-	-	**2488**
RC106	**20921.6**	**21109.7**	-	-	**2912**	**20500.1**	**20793.3**	-	-	**2126**
RC107	**19580.3**	**19686.7**	-	-	**118**	**19180.4**	**19322.9**	-	-	**332**
RC108	-	-	-	-	-	**18582.9**	**18643.1**	-	-	**92**
C201	-	-	-	-	-	95919.7	96015.8	96015.8	15630.6	22

Table 5.4 Instances with 100 customers.

	Nb. instances	Avg. Root LB	Avg. Best LB	Avg. Time(s)	Avg. Tree
with TDESPPRC	70	15446.6	15524.9	14022.7	28
with TDSPPRC	62	15327.4	15493.4	7467.8	261

Table 5.5 Aggregate comparison between TDESRRPC and TDSPPRC

complicating factors arise. In fact, the optimal solution depends on the sequence in which customers are visited as well as on dispatch times at the depot. Furthermore, the triangle inequality is not satisfied for time-dependent travel times as shorter travel times can be obtained by taking diverted routes. Moreover, insights on the structure of the instances are lost as the link between two geographically close customers can be seen as a long link when traversed during a congested period.

In this chapter we present the first exact method for solving the TDVRPTW. Considering time-dependent travel times increases the complexity of the pricing problem. In fact, a resource should be added where arrival time functions are stored as a function of dispatch time at the depot. Furthermore, standard dominance tests become difficult and weak. We introduce a new stronger dominance test allowing the domination of more labels. Computational results show that some instances with up to 100 customers can be solved to optimality but also that several instances with only 25 customers remain unsolved.

	with Bi-directional			with Mono-directional		
Instance	**Best LB**	**UB**	**Time(s)**	**Best LB**	**UB**	**Time(s)**
R101	9010.5	9010.5	0.1	9010.5	9010.5	0.1
R102	7464.2	7464.2	0.5	7464.2	7464.2	0.5
R103	6526.1	6526.1	10.0	6526.1	6526.1	9.4
R104	5910.6	5910.6	66.0	5910.6	5910.6	93.0
R105	7190.5	7190.5	0.2	7190.5	7190.5	0.3
R106	6512.5	6512.5	1.4	6512.5	6512.5	1.8
R107	5963.3	5963.3	5.0	5963.3	5963.3	6.7
R108	5651.8	5651.8	56.9	5651.8	5651.8	132.4
R109	6105.6	6105.6	0.9	6105.6	6105.6	2.1
R110	5909.5	5909.5	1.3	5909.5	5909.5	1.8
R111	6028.9	6028.9	23.5	6028.9	6028.9	28.0
R112	5602.5	5602.5	170.2	5602.5	5602.5	171.0
C101	24684.2	24684.2	0.5	24684.2	24684.2	22.3
C102	24504.9	24504.9	140.3	24504.9	24504.9	1740.4
C103	24370.9	24370.9	57775.3	-	-	-
C105	24672.5	24672.5	12.7	24672.5	24672.5	414.3
C106	24684.2	24684.2	0.7	24684.2	24684.2	14.5
C107	24499.8	24499.8	63.2	24499.8	24499.8	13452.6
C108	24392.6	24392.6	356.1	24392.6	24392.6	29205.7
C109	24312.6	24312.6	12791.7	-	-	-
RC101	7004.3	7004.3	8.4	7004.3	7004.3	10.3
RC102	6162.2	6162.2	27.1	6162.2	6162.2	35.2
RC103	5422.4	5422.4	7.7	5422.4	5422.4	26.2
RC104	5174.9	5174.9	23.5	5174.9	5174.9	113.9
RC105	6810.7	6810.7	3.2	6810.7	6810.7	3.8
RC106	5590.5	5590.5	0.9	5590.5	5590.5	1.7
RC107	5005.4	5005.4	3.7	5005.4	5005.4	9.3
RC108	4893.0	4893.0	24.1	4893.0	4893.0	64.5
R201	7907.2	7907.2	18.0	7907.2	7907.2	62794.1
R202	7091.7	7091.7	6151.9	-	-	-
R205	6471.8	6471.8	19109.5	-	-	-
R209	5526.0	5531.4	-	-	-	-
C201	25581.0	25581.0	1.5	25581.0	25581.0	1683.3
C202	24728.5	24728.5	41213.6	-	-	-
C205	25056.3	25056.3	106.0	25056.3	25056.3	59975.9
C206	24928.5	24928.5	1926.3	-	-	-
C208	24747.6	24747.6	9869.3	-	-	-
RC201	8350.4	8350.5	712.4	-	-	-
RC202	7409.0	7409.0	71371.8	-	-	-
RC205	7602.4	7602.4	1493.3	-	-	-

Table 5.6 Bi-directional vs. Mono-directional (TDESPPRC, 25 customers)

Start time	**End time**	**Cost**	**Route**
7680	8698.5	1018.5	24, 25
853.4	3415.6	2562.2	14, 5, 2, 12, 11, 16, 15, 7
5761.9	8246.1	2484.2	10, 20, 13, 17, 4, 1
2865	5175.6	2310.6	23, 21, 18, 19, 22, 9, 6, 8, 3

Table 5.7 Solution of RC201.25

Chapter 6

Conclusions

The stochastic and dynamic real-world in which organizations are operating increases the complexity of operational decisions. To improve the reliability and the feasibility of these decisions, it is crucial to take real-life uncertainties into account. On top of this, many other aspects contribute to the complexity of the decision making process. These aspects come in the form of governmental regulations (e.g., working regulations, regulations on CO_2 emissions etc), customer restrictions (e.g., delivery time windows), or intern rules (e.g., level workload). Clearly, organizations have multiple objectives, and hence need to generate the set of plans that capture the interactions between these objectives instead of a unique optimal plan. In this thesis, we developed exact, approximation and heuristic solution procedures to account for the dynamic nature of optimization problems by considering time-dependent cost parameters. Moreover, the proposed methods allow for optimizing multiple criteria objective functions.

Considering the dynamic and multi-objective nature of optimization problems, a number of complicating factors affect both the quality and the easiness of obtaining a solution. First, the number of Pareto solutions increases with the size of the problem, mainly with the number of objectives. Therefore, algorithms designed to solve multi-objective optimization problems process more data during their execution, which requires both more storage memory and computation effort. Moreover, multi-objective decision making does not end when the Pareto front is found. In practice, only a single solution (or a region of solutions), taking decision makers preferences into account, needs to be implemented. Therefore, after generating the complete Pareto front, a selection of a solution (or a region of solutions) should be carried out. This post-optimization process of selecting a solution is a complicated problem in itself. However, it is out of the scope of this thesis.

Secondly, including time-dependent cost parameters adds an additional dimension to the optimization problem. Consequently, the search for a solution is done in a larger feasible space. In fact, a solution does not only depend on the type of the decision (e.g., accept or reject an order), but is also dependent upon the time this decision is taken. Moreover, insights into the structure of the optimization problems are usually lost. In case of the vehicle routing problem, an algorithm might suggest to visit the closest customer. In case of time-independent travel times, the closest customer is well defined in terms of distance. However, when travel times are time-dependent, two customers that are geographically close to each other might be seen as far from each other when the link between them is traversed in a congested period. Furthermore, useful properties that facilitate the search through the feasible space become invalid. For instance, the triangle inequality is not satisfied for time-dependent travel time as it might be shorter to travel to the next customer by taking a diverted route rather than by traveling directly. In this thesis, we studied two interesting optimization problems, e.g., the vehicle routing problem with time windows and the knapsack problem. Our study focuses on two aspects, time and multiple objectives.

In the VRPTW, the inclusion of time is reflected by capturing the effect of road congestion on travel times and on customers' demand. In fact, traveling during a congested period results in higher travel times, and hence in later arrival times at customers. Therefore, time has a significant impact on customer service (i.e., delivery within time windows) and on the profitability (i.e., the quantity sold) of the planned routes. In the knapsack problem, time is included by considering items' profits that are time varying. Clearly, time has a major effect on the profitability of the chosen schedule as not only the selected items are determinant, but also the time on which they are included in the schedule is crucial.

In the context of multi-objective optimization, we introduced a generic framework to deal with multi-objective scheduling problems. Realizing that real-life optimization problems involve the simultaneous optimization of several objectives, the aim is to determine the set of Pareto (or non-dominated) solutions that capture the trade-offs between these objectives. Multi-objective optimization problems are at least as hard as their single-objective version. Consequently, finding the Pareto front is usually computationally very difficult, and hence multi-objective optimization problems are often solved by means of heuristic solution procedures. Two main drawbacks of the existing solution methods are the poor quality and the insufficient coverage capability of the generated solutions.

First, we introduce an efficient and flexible approximation framework for multi-objective scheduling problem with a generic state-dependent cost structure (e.g., time-dependent cost parameters are a special case), and for which a dynamic programming solution is possible. The approximation produces good quality approximate Pareto

fronts. In fact, for every solution in the approximate Pareto front, the worst case performance guarantee is provable. Moreover, the approximate Pareto fronts contain fewer solutions. However, the approximate Pareto fronts represent a good coverage of the real Pareto fronts in the sense that every Pareto solution has its counterpart in the approximate Pareto front. Furthermore, the process of selecting and implementing a solution is facilitated. The quality of the approximate Pareto fronts can be controlled for each objective separately as the error allowed on each objective is an input to the algorithm and can be tuned by the decision maker. Two multi-objective scheduling problems with time-dependent cost parameters are treated in this thesis, namely, the time-dependent multi-objective single vehicle routing problem with time windows, a predefined customer sequence and multiple depot returns, and the time-dependent multi-objective knapsack problem.

Secondly, we present the first exact solution procedure for solving the time-dependent VRPTW. The algorithm is based on a branch-and-cut-and-price framework. We consider the minimization of route duration. Even in the time-independent setting, minimizing route duration is a more complicated problem than minimizing route distance. Considering time-dependent travel times increases the complexity of the pricing problem. In fact, an additional resource needs to be added in the labeling algorithm in order to store arrival times as a function of dispatch time at the depot. Moreover, standard dominance tests become difficult and weak. We design a labeling algorithm able to deal with the additional complexities caused by time-dependent travel times. Furthermore, we introduce a new and stronger dominance test to discard more labels that do not lead to a path in the final optimal solution.

6.1. Discussion

Incorporating time and multiple objectives in scheduling and routing problems and designing frameworks that deal with the additional complexity is the overall main contribution of this thesis. We developed both approximations and an exact solution procedure. Additionally, heuristic methods are incorporated in the exact solution procedure to decrease CPU times. Furthermore, we analyzed the performance of our algorithms and derived insights when possible.

In Chapter 2, we considered a generic multi-objective scheduling problem for which a dynamic programming formulation is possible. Moreover, we considered realistic cost parameters that depend upon the state of the studied system. In Chapters 3 and 4, the state of the system is a function of time. For instance in case of the VRPTW, travel costs depend on the state of the traffic network in which the vehicles are operating. The state of the traffic is described in terms of road congestion, which

depends on the time of the day. The state of the network, and hence travel costs are time-dependent.

In Chapter 3, numerical experiments were conducted on an objective function with 4 objectives. In most cases, it was not possible to compute the Pareto front. Therefore, we designed an approximation algorithm that produces approximate Pareto fronts for which the quality is provable for each objective. In fact, the precision ϵ of the approximation is an input to the algorithm and can be tuned by the decision maker. When the precision is set to 5%, for all objectives, almost all approximate Pareto fronts are calculated. When the precision is set to 10% all instances are solved. Furthermore, a significant decrease in the CPU time (up to 97%) and in the size of the approximate Pareto fronts (up to 95%) is achieved.

In Chapter 4, we considered an objective function with two objectives. A slightly different approximation algorithm is designed. The trimming action is more severe towards the end of the dynamic programming. To correctly handle time windows, from each cluster defined by the trimming action, we keep the solution with the least total travel time. For all instances, it was possible to generate the Pareto fronts. This is probably partially due to the fact that we only have two objectives, and to the existence of time windows, which reduces the number of feasible solutions. Furthermore, again considerable gains are achieved in CPU times (up to 65% for a precision of 10%), and the size of the produced approximate Pareto fronts is significantly smaller than the true Pareto fronts (up to 40% for a precision of 10%).

In general, there is a trade-off between the precision, CPU times and the size of the produced fronts. On the one hand, low errors result in good quality solutions, but slow algorithms and large approximate Pareto fronts. On the other hand, large errors imply fast algorithms and less dense approximate Pareto fronts, but worse solutions. From the numerical results, a precision of 10% represents a good compromise.

In Chapter 5, we considered the TDVRPTW with a single objective function, and time-dependent travel costs. We aimed to determine the set of routes with the least total duration. In this thesis, we present the first exact solution procedure for the TDVRPTW. The corresponding optimization problem entails a routing component (i.e., deciding on the customer sequence) and a scheduling component (i.e., deciding on the vehicles' dispatch time at the depot). Consequently, the optimization of the TDVRPW is harder than the optimization of the VRPTW. As a matter of fact, the additional complexity is reflected in the designed BCP framework. More specifically, the labeling algorithm that solves the pricing problem suffers from incorporating time-dependent travel times. In general, more routes are needed to serve all customers, which can be explained by two facts. First, more routes are needed to create more flexibility to avoid congestion. Secondly, the considered objective function (i.e., routes

duration) forces the vehicles to go back to the depot and wait if waiting time at the next customer is very high. However, due to the scheduling component of the decision, routes are defined by a start and an end time. Consequently, routes that are non-overlapping in time are performed by the same vehicle, which decreases the number of vehicles and hence congestion in the traffic network. Note that in case of the VRPTW, the solution implies that each route is performed by exactly one vehicle. For our numerical results, we modified Solomon's data sets by adding time-dependency. Our algorithm is able to solve about 70% of the instances with 25 customers, 47% of the instances with 50 customers and 18% of the instances with 100 customers.

6.2. Future Research

The research presented in this thesis can be extended in many directions. In the context of multi-objective optimization, decision makers are interested in implementing one solution, or a limited region of solutions, that fits their preferences. In this line of thought, if these preferences are well-defined in advance, the dynamic programming approximation can be tailored in such a way that the precision outside the area defined by the decision makers preferences is relaxed. This will reduce CPU time, and thus provide more flexibility to tighten the precision in the area defined by the decision makers preferences. Furthermore, other important objectives can be included in the optimization problem. For instance, approaching the problem of CO_2 emissions in the TDVRP from a multi-objective perspective is a very interesting research question.

The branch-and-cut-and-price framework developed in Chapter 5 can be enhanced in many ways. First, the dominance test can further be improved to discard more labels in the labeling algorithm. One way to do that, is to search for all nodes that are unreachable from the end node of a label. Finding all unreachable nodes from a label implies solving a shortest path problem from the end node of that label to all other nodes that can still be included in the corresponding partial path. Moreover, a bounding test can be implemented to eliminate non-dominated labels that, when extended in the most beneficial way, does not improve the current known feasible solution. Furthermore, valid inequalities for the set partitioning problem can be implemented to tighten the lower bound obtained from solving the LP master problem. However, adding theses inequalities result in a harder pricing problem.

In Chapter 5, we consider a single objective function. Modifying the BCP framework to deal with the multi-objective variant of the TDVRPTW is an interesting extension. One possible way to achieve this, is to adopt an $\epsilon-$constraint method as explained in Bérubé et al. (2009). In other words, our BCP framework can be used to solve

a series of single objective subproblems, where all but one objective are transformed into constraints. To find all the routes in the Pareto front, the right hand side of the resulting constraints is incrementally changed and the new subproblems are solved again.

In the context of the time-dependent VRPTW, it is very relevant to account for driving regulations in our solution procedures. Driving regulations have a significant impact on the feasibility of the planned routes. Consequently, incorporating break scheduling into the BCP framework is a very interesting research question. All existing research in this area assumes time-independent travel time, and is based on heuristic solution procedures. Intuitively, there is an incentive to schedule breaks during the congested periods and drive during in the rest of the operating period.

Bibliography

Androutsopoulos, K. N., K. G. Zografos. 2009. Solving the multi-criteria time-dependent routing and scheduling in a multimodal fixed scheduled network. *European Journal of Operational Research* **192** 18–28.

Azi, N., M. Gendreau, J. Y. Potvin. 2007. An exact algorithm for a single-vehicle routing problem with time windows and multiple routes. *European Journal of Operational Research* **178** 755–766.

Balas, E., E. Zemel. 1980. An algorithm for large 0-1 knapsack problems. *Operations Research* **28** 1130–1154.

Baldacci, R., A. Mingozzi, R. Roberti. 2011. New route relaxation and pricing strategies for the vehicle routing problem. *Operations Research, to appear* .

Balseiro, S. R., I. Loiseau, J. Ramonet. 2011. An ant colony algorithm hybridized with insertion heuristics for the time dependent vehicle routing problem with time windows. *Computers and Operations Research* **38** 954–966.

Barnhard, C., E. L. Johnson, G. L. Nemhauser, M. W. P. Savelsbergh, P. H. Vane. 1998. Branch-and-price: Column generation for solving huge integer programs. *Operations Research* **46** 316–329.

Bazgan, C., H. Hugot, D. Vanderpooten. 2009a. Implementing an efficient fptas for the 0-1 multi-objective knapsack problem. *European Journal of Operational Research* **198** 47–56.

Bazgan, C., H. Hugot, D. Vanderpooten. 2009b. Solving efficiently the 0-1 multi-objective knapsack problem. *Computers and Operations Research* **36** 260–279.

Bellman, R. 1956. Dynamic programming. *Princeton University Press, Princeton, NJ* .

Bellman, R. 1962. Dynamic programming treatment of the travelling salesman problem. *J. Assoc. Comput. Machinery* **9** 61–63.

Ben Abdelaziz, F., S. Krichen, J. Chaouadi. 1999. *Meta-heuristics: advances and trends in local search paradigms for optimization*, chap. A hybrid heuristic for multiobjective knapsack problems. Kluwer Academic Publishers, 205–212.

Bertsimas, D. J., G. Van Ryzin. 1991. A stochastic and dynamic vehicle routing problem in the euclidian plane. *Operations Research* **39** 601–615.

Bertsimas, D. J., G. Van Ryzin. 1993a. Stochastic and dynamic vehicle routing problems in the euclidean plane with multiple capcitated vehicles. *Operations Research* **41** 60–76.

Bertsimas, D. J., D. Simchi-Levi. 1996. A new generation of vehicle routing research: robust algorithms, addressing uncertainty. *Operations Research* **44** 286–304.

Bérubé, J. F., M. Gendreau, J. Y. Potvin. 2009. An exact $\epsilon-$constraint method for bi-objective combinatorial optimization problems: Application to the traveling salesman problem with profits. *European Journal of Operational Research* **194** 39–50.

Bettinelli, A., A. Ceseli, G. Righini. 2010. A branch-and-cut-and-price algorithm for the multi-depot heterogeneous vehicle routing problem with time windows. *Transportation Research Part C, In Press* .

Bräysy, O., M. Gendreau. 2005a. Vehicle routing problem with time windows, part i: Route construction and local search algorithms. *Transportation Science* **39** 104–118.

Bräysy, O., M. Gendreau. 2005b. Vehicle routing problem with time windows, part ii: Metaheuristics. *Transportation Science* **39** 119–139.

Camerini, P.M., C. Vercellis. 1984. The matroidal knapsack: a class of (often) well-solvable problems. *Operations Research Letters* **3** 157–162.

Captivo, M. E., J. Clímaco, J. Figueira, E. Martins, J. L. Santos. 2003. Solving bicriteria 0Ũ-1 knapsack problems using a labeling algorithm. *Computers and Operations Research* **30** 1865–1886.

Carraway, R. L., T. L. Morin, H. Moskowitz. 1990. Generalized dynamic programming for multicriteria optimization. *European Journal of Operational Research* **44** 95–104.

Chabrier, A. 2006. Vehicle routing problem with elementary shortest path based column generation. *Computers and Operations Research* **33** 2972–2990.

Christofides, N., A. Mingozzi, P. Toth. 1976. Exact algorithms for the vehicle routing problem, based on spanning tree and shortest path relaxation. *Mathematical Programming* **20** 255–282.

COIN CLP. 2011. COIN-OR linear programming solver. `https://projects.coin-or.org/Clp`.

Cook, W., J. L. Rich. 1999. A parallel cutting plane algorithm for the vehicle routing problem with time windows. *Technical Report TR99-04, Computational and Applied Mathematics, Rice University, Housten, USA* .

Coyle, J. J., E. J. Bardi, Jr. C. J. Langley. 1996. West Publishing Company, USA.

Da Silva, C. G., J. C. N. Climaco, J. R. Figueira. 2008. Core problems in the bi-criteria 0-1 knapsack problems. *Computers and Operations Research* **35** 2292–2306.

Dantzig, G. B. 1957. Discrete variable extremum problems. *Operations Research* **5** 266–277.

Dantzig, G. B., J. H. Ramser. 1959. The truck dispatching problem. *Management Science* **2** 80–91.

Dantzig, G. B., P. Wolfe. 1960. Decomposition principle for linear programs. *Operations Research* **8** 101–111.

Desaulniers, G., F. Lessard, A. Hadjar. 2008. Tabu search, partial elementarity, and generalized k-path inequalities for the vehicle routing problem with time windows. *Transportation Science* **42** 387–404.

Desrochers, M. 1986. La fabrication d'horaire de travail pour les conducteurs d' autobus par une methode de generation de colonnes. *PhD thesis, Universite de Montréal, Montréal, Canada.*

Desrochers, M., J. Desrosiers, M. Solomon. 1992. A new optimization algorithm for the vehicle routing problem with time windows. *Operations Research* **40** 342–354.

Donati, A. F., R. Montemanni, N. Casagrande, A. E. Rizzoli, L. M. Gambardella. 2008. Time dependent vehicle routing problem with a multi ant colony system. *Eurorpean Journal of Operational Research* **185** 1174–1191.

Ecker, J. G., M. Kupferschmid, C. E. Lawrence, A. A. Reilly, A. C. H. Scott. 2002. An application of nonlinear optimization in molecular biology. *European Journal of Operational Research* **138** 452Ű–458.

Edgeworth, F. Y. 1881. Mathematical psychics: An essay on the application of mathematics to the moral sciences. *C. Kegan Paul and Co., London.*

Ehrgott, M. 2005. *Multicriteria Optimization.* Springer.

Erlebach, T., H. Kellerer, U. Pferschy. 2002. Approximating multiobjective knapsack problems. *Management Science* **48** 1603–1612.

Farina, M., K. Deb, P. Amato. 2004. Dynamic multi-objective optimization problems: Test cases, approximations, and applications. *IEEE Trans. on Evolutionary Computation* **8** 425–442.

Fayard, D., G. Plateau. 1975. Resolution of the 0-1 knapsack problem: Comparison of methods. *Mathematical Programming* **8** 277–307.

Feillet, D., P. Dejax, M. Gendreau. 2004a. Traveling salesman problems with profits. *Transportation Science* **39** 188–205.

Feillet, D., P. Dejax, M. Gendreau, C. Gueguen. 2004b. An exact algorithm for the elementary shortest path problem with resource constraints: Application to some vehicle routing problems. *Networks* **44** 216–229.

Ferreira, J. C., C. M. Fonseca, A. Gasper-Cunha. 2007. Methodology to select solutions from the pareto-optimal set: a comparative study. *Proceeding GECCO '07 Proceedings of the 9th annual conference on Genetic and evolutionary computation, London*.

Fleischmann, B., M. Gietz, S. Gnutzmann. 2004. Time-varying travel times in vehicle routing. *Transportation Science* **38** 160–173.

Freville, A. 2004. The multidimensional 0-1 knapsack problem: An overview. *European Journal of Operational Research* **155** 1–21.

Fukasawa, R., H. Longo, J. Lysgaard, M. Poggi de Aragão, M. Reis, E. Uchoa, R. F. Werneck. 2006. Robust branch-and-cut-and-price for the capacitated vehicle routing problem. *Mathematical Programming* **16** 491–511.

Garey, M. R., D. S. Johnson. 1978. 'strong' np-completeness results: Motivation, examples, and implications. *Journal of the ACM* **25** 499Ű–508.

Garey, M. R., D. S. Johnson. 1979. *Computers and In tractability: A Guide to the Theory of NP-Completeness*. Freeman, san Francisco.

Gendreau, M., C. D. Tarantilis. 2010. Solving large-scale vehicle routing problems with time windows: The state of the art. Tech. rep., CIRRELT.

Giaglis, G. M., I. Minis, A. Tatarakis, A. Zeimpekis. 2004. Minimizing logistics risk through real-time vehicle routing and mobile technologies: Research to-date and future trends. *International Journal of Physical Distribution and Logistics Management* **34** 749–764.

Greenberg, H. J., W.E. Hart, G. Lancia. 2004. Opportunities for combinatorial optimization in computational biology. *Informs Journal On Computing* **16** 211Ű–231.

Gribkovskaia, I., G. Laporte, S. Aliaksandr. 2008. The single vehicle routing problem with deliveries and selective pickups. *Computers and Operations Research* **35** 2908–2924.

Gribkovskaia, I., Ø. sr. Halskau, G. Laporte, M. Vlĉek. 2007. General solutions to the single vehicle routing problem with pickups and deliveries. *European Journal of Operational Research* **180** 568–584.

Haghani, A., S. Jung. 2005. A dynamic vehicle routing problem with time-dependent travel times. *Computers and Operations Research* **32** 2959–2986.

Hamacher, H. W., G. Ruhe. 1994. On spanning tree problems with multiple objectives. *Annals of Operations Research* **52** 209–230.

Hansen, M. 1997. Solving multiobjective knapsack problems using mots. *Conference Paper presented at MIC'97*. 9pp.

Hashimoto, H., M. Yagiura, T. Ibaraki. 2008. An iterated local search algorithm for the time-dependent vehicle routing problem with time windows. *Discrete Optimization* **5** 434–456.

Held, M., R. M. Karp. 1962. A dynamic programming approach to sequencing problems. *J. Soc. Indust. Appl. Math.* **10** 196–210.

Hill, A.V., W.C. Benton. 1992. Modeling intra-city time-dependent travel speeds for vehicle scheduling problems. *European Journal of Operational Research* **43** 343–351.

Hong, S. C., Y. B. Park. 1999. A heuristic for bi-objective vehicle routing with time window constraints. *International Journal of Production Economics* **62** 249–258.

Horowitz, E., S. Sahni. 1974. Computing partitions with applications to the knapsack problem. *Journal of ACM* **21** 277–292.

Ibarra, O. H., C. E. Kim. 1975. Fast approximation algorithms for the knapsack and sum of subset problems. *Journal of the ACM* **22** 463–468.

Ichoua, S., M. Gendreau, J. Y. Potvin. 2003. Vehicle dispatching with time-dependent travel times. *European Journal of Operational Research* **144** 379–396.

Irnich, S., D. Villeneuve. 2006. The shortest path problem with resource constraints and k-cycle elimination for $k \geq 3$. *INFORMS Journal on Computing* **18** 391–406.

Ishibuchi, H., N. Tsukamoto, Y. Nojima. 2008. Behavior of evolutionary many-objective optimization. *Tenth International Conference on Computer Modeling and Simulation, UKSIM'2008*. 266–271.

Jabali, O., T. van Woensel, A.G. de Kok, C. Lecluyse, H. Permans. 2009. Time-dependent vehicle routing subject to time delay perturbations. *IIE Transaction* **41**

1049–1066.

Jespen, M., B. Petersen, S. Spoorendonk, D. Pisinger. 2008. Subset-row inequalities applied to the vehicle-routing problem with time windows. *Operations Research* **56** 497–511.

Jorritsma, P., J. Berveling, L. Harms, J. Kolkman, C. Koopmans, M. Lijsen, H. van der Loop, M. J. Olde Kalter, H. van Ooststroom, J. Visser, P. Warffemius. 2008. *Mobiliteitsbalans*. Kennisinstotuut voor Mobiliteitsbeleid.

Jozefowiez, N., F. Semet, E-G. Talbi. 2007. The bi-objective covering tour problem. *Computers and Operations Research* **34** 1929–Ũ1942.

Jozefowiez, N., F. Semet, El. Talbi. 2008. Multi-objective vehicle routing problems. *European Journal of Operational Research* **189** 293–309.

Kallehauge, B. 2008. Formulations and exact algorithms for the vehicle routing problem with time windows. *Computers and Operations Research* **35** 2307–2330.

Karp, R. M. 1972. Reducibility among combinatorial problems. *In R.E. Miller and J. W. Thatcher, editors, Complexity of Computer Computations, Advances in Computing Research*. 85–103.

Kellerer, H., U. Pferschy, D. Pisinger. 2005. *Knapsack Problems*. Springer Verlag.

Klamroth, K., M. M. Wiecek. 2000a. Dynamic programming approaches to the multiple criteria knapsack problem. *Naval Research Logistics* **47** 57–76.

Klamroth, K., M. M. Wiecek. 2001. A time-dependent multiple criteria single-machine scheduling problem. *European Journal of Operational Research* **135** 17–26.

Klamroth, K., M.M. Wiecek. 2000b. Time-dependent capital budgeting with multiple criteria. *In: Haimes, Y.Y. and Steuer, R.E. (Eds.), Research and Practice in Multiple Criteria Decision Making.Lecture Notes in Economics and Mathematical Systems,Springer-Verlag*, vol. 487. 421–432.

Kohl, N., J. Desrosiers, O. B. G. Madsen, M. M. Solomon, F. Soumis. 1999. 2-path cuts for the vehicle routing problem with time windows. *Transportation Science* **33** 101–116.

Kok, L. A., C. M. Meyer, H. Kopfer, J. M. J. Schutten. 2010. A dynamic programming heuristic for the vehicle routing problem with time windows and european community social legislation. *Transportation Science* **44** 442–454.

Kolen, A. W. J., A. H. G. Rinnoy Kaan, H. W. J. M. Trienekens. 1987. Vehicle routing with time windows. *Operations Research* **135** 17–26.

Kostreva, M. M., M. M. Wiecek. 1993. Time dependency in multiple objective

dynamic programming. *Journal of Mathematical Analysis and Applications* **173** 289–307.

Laporte, G. 1992. The vehicle routing problem: An overview of exact and approximate algorithms. *European Journal of Operational Research* **59** 345–358.

Laporte, G. 2007. What you should know about the vehicle routing problem. *Naval Research Logistics* **54** 811–819.

Laporte, G., Y. Nobert. 1980. A cutting planes algorithm for the m-salesmen problem. *Journal of the Operational Research Society* **31** 1017–1023.

Laporte, G., Y. Nobert. 1987. Exact algorithms for the vehicle routing problem. *Annals of Discrete Mathematics* **31** 147–184.

Laumanns, M., L. Thiele, E. Zitzler. 2004. An adaptive scheme to generate the Pareto front based on the epsilon-constraint method. Tech. rep., Computer Engineering and Networks Laboratory (TIK), Swiss Federal Institute of Technology (ETH) Zurich.

Lemesre, J., C. Dhaenens, E.-G. Talbi. 2006. Méthode parallèle par partitions: Passage d'une méthode exacte bi-objectif à une méthode exacte multi-objectif. *ROADEF'06 Proceedings*.

Lemesre, J., C. Dhaenens, E-G. Talbi. 2007a. An exact parallel method for a bi-objective permutation flowshop problem. *European Journal of Operational Research* **177** 1641–1655.

Lemesre, J., C. Dhaenens, E-G. Talbi. 2007b. Parallel partitioning method (PPM) : a new exact method to solve bi-objective problems. *Computers and Operational Research* **34** 2450–2462.

Liefooghe, A., M. Basseur, L. Jourdan, E-G. Talbi. 2007. Combinatorial optimization of stochastic multi-objective problems: An application to the flow-shop scheduling problem. *EMO'2007 Evolutionary Multi-criterion Optimization*, vol. LNCS 4403. Springer, 457–471.

Lübbecke, M. E., J. Desrosiers. 2005. Selected topics in column generation. *Operations Research* **53** 1007–1023.

Lysgaard, J. 2003. Cvrpsep: A package of separation routines for the capacitated vehicle routing problem. Tech. rep., Department of Management Science and Logistics, Aarhus School of Business, Denmark.

Lysgaard, J., A. N. Letchford, R. W. Eglese. 2004. A new branch-and-cut algorithm for the capacitated vehicle routing problem. *Mathematical Programming* **100** 423–445.

Maden, W., R. Eglese, D. Black. 2010. Vehicle routing and scheduling with time-varying data: A case study. *Journal of the Operational Research Society* **61** 515–522.

Magazine, M. J., O. Oguz. 1981. A fully polynomial approximation algorithm for the 0Ű-1 knapsack problem. *European Journal of Operational Research* **8** 270Ű–273.

Malandraki, C., M.S. Daskin. 1992. Time dependent vehicle routing problems: formulations, properties and heuristic algorithms. *Transportation Science* **26** 185–200.

Malandraki, C., R. B. Dial. 1996. A restricted dynamic programming heuristic algorithm for the time dependent traveling salesman problem. *European Journal of Operational Research* **90** 45–55.

Mandow, L., E. Millan. 1996. Goal programming and heuristic search. R. Caballero, F. Ruiz, R. Steuer, eds., *Second Int. Conf. on Multi-Objective Programming and Goal Programming MOPGP'96*. Springer-Verlag, Torremolinos, Spain, 48–56.

Martello, S., D. Pisinger, P. Toth. 1999. Dynamic programming and strong bounds for the 0-1 knapsack problem. *Management Science* **45** 414–424.

Martello, S., P. Toth. 1988. A new algorithm for the 0-1 knapsack problem. *Management Science* **34** 633–644.

Martello, S., P. Toth. 1990. *Knapsack problems: algorithms and computer implementations*. Wiley, New York.

Miettinen, K. 1999. *Nonlinear multiobjective optimization*. Kluwer.

Minis, I., A. Tatarakis. 2011. Stochastic single vehicle routing problem with delivery and pick up and a predefined customer sequence. *European Journal of Operational Research. In Press* .

Nauss, R.M. 1976. An effcient algorithm for the 0-1 knapsack problem. *Management Science* **23** 27–31.

Papadimitriou, C. H., M. Yannakakis. 2002. On the approximability of trade-offs and optimal access of web services. *IEEE Symp. on Foundations of Computer Science* 86–92.

Pareto, V. 1896. Cours d'économie politique. *Rouge, Lausanne, Switzerland*.

Pisinger, D. 1995. Algorithms for the knapsack problem. *PhD Thesis* .

Pisinger, D. 1997. A minimal algorithm for the 0-1 knapsack problem. *Operations Research* **45** 758–767.

Pisinger, D. 2000. A minimal algorithm for the bounded knapsack problem. *INFORMS Journal on Computing* **34** 75–84.

Pisinger, D. 2007. The quadratic knapsack problem: A survey. *Discrete Applied Mathematics* **155** 623–648.

Ribeiro, R., H. R. Lourenço. 2001. A multi-objective model for a multi-period distribution management problem. *Meta-heuristics International Conference* **MIC'2001** 97–101.

Righini, G., M. Salani. 2006. Symmetry helps: Bounded bi-directional dynamic programming for the elementary shortest path problem with resource constraints. *Discrete Optimization* **3** 255–273.

Righini, G., M. Salani. 2008. New dynamic programming algorithms for the resource constrained elementary shortest path problem. *Networks* **51** 155–170.

Ropke, S., J. F. Cordeau. 2009. Branch and cut and price for the pickup and delivery problem with time windows. *Transportation Science* **43** 267–286.

Rosenblatt, M. J., Z. Sinuany-Stern. 1989. Generating the discrete efficient frontier to the capital budgeting problem. *Operations Research* **37** 384–394.

Sahni, K. S. 1976. Algorithms for scheduling independent tasks. *Journal of the ACM* **23** 116–127.

Sahni, S. 1975. Approximate algorithms for 0-1 knapsack problem. *Journal of the ACM* **22** 115–124.

Savelsbergh, M. W. P. 1985. Local search in routing problems with time windows. *Annals of Operations Research* **4** 285–305.

Sayin, S., S. Karabati. 1999. A bicriteria approach to the two-machine flow shop scheduling problem. *European Journal of Operational Research* **113** 435–449.

Schrank, D., T. Lomax. 2007. The 2007 urban mobility report. *The Texas A&M University System* .

Sen, T., M. E. Raiszadeh, P. Dileepan. 1988. A branch and bound approach to the bicriterion scheduling problem involving total flowtime and range of lateness. *Management Science* **34** 254–260.

Serafini, P. 1986. Some considerations about computational complexity for multiobjective combinatorial problems. *In Jahn, J., Krabs, W. (Eds.), Recent Advances and Historical Development of Vector Optimization, LNMES. Springer-Verlag*, vol. 294. 222–232.

Solomon, M. M. 1987. Algorithms for the vehicle routing and scheduling problems

with time window constraints. *Operations Research* **35** 254–265.

Steuer, R. 1986. *Multiple criteria optimization: Theory, computation and application.* Wiley, New York.

Stewart, B. S., C. C. White. 1991. Multiobjective A*. *Journal of the ACM* **38** 775–814.

Süral, H., J. H. Bookbinder. 2003. The single-vehicle routing problem with unrestricted backhauls. *Networks* **41** 127Ű–136.

Taillard, E., P. Badeau, M. Gendreau, F. Geurtin, J. Y. Potvin. 1997. A tabu search heuristic for the vehicle routing problem with soft time windows. *Transportation Science* **31** 170–186.

Talbi, E-G. 2009. *Metaheuristics: from design to implementation.* Wiley.

Tang, H. 2008. Efficcient implementation of improvement procedures for vehicle routing with time-dependent travel times. *Transportation Research Record* 66–75.

Tatarakis, A. 2007. A class of single vehicle routing problems with predefined customer sequence and depot returns. *PhD Thesis, University of the Aegean, Greece* .

Tatarakis, A., I. Minis. 2009. Stochastic single vehicle routing with a predefined customer sequence and multiple depot returns. *European Journal of Operational Research* **197** 557–571.

Toth, P. 1980. Dynamic programmming algorithms for the 0-1 knapsack problem. *Computing* **25** 29–45.

Toth, P., D. Vigo. 2002. *The vehicle Routing Problem*, vol. 9. SIAM Monographs on Discrete Mathematics and Applications. SIAM, Philadelphia.

Tsirimpas, P., A. Tatarakis, E. G. Kyriakidis. 2008. Single vehicle routing with a predefined customer sequence and multiple depot returns. *European Journal of Operational Research* **187** 483–495.

Ulungu, E. L., J. Teghem. 1995. The two phase method: An efficient procedure to solve bi-objective combinatorial optimization problems. *Foundations of Computing and Decision Sciences*, vol. 20. 149–165.

Ulungu, E. L., J. Teghem. 1997. Solving multi-objective knapsack problems by a branch and bound procedure to solve the bi-objective knapsack problem. *Multicriteria analysis, J. N. Climaco(Editor), Springer-Verlag, New york* 269–278.

Ulungu, E.L., J. Teghem, P.H. Fortemps, D. Tuyttens. 1999. Mosa method: A tool for solving multi-objective combinatorial optimization problems. *Journal of Multi-Criteria Decision Analysis* **8** 221–236.

Van Woensel, T., L. Kerbache, H. Peremans, N. Vandaele. 2008. Vehicle routing with dynamic travel times: a queueing approach. *European Journal of Operational Research* **186** 990–1007.

Van Woensel, T, N. Vandaele. 2006. Empirical validation of a queueing approach to uninterrupted traffic flows. *4OR, A Quarterly Journal of Operations Research* **4** 59–72.

Van Woensel, T, B. Wuyts, N. Vandaele. 2006. Validating state-dependent queueing models for uninterrupted traffic flows using simulation. *4OR, A Quarterly Journal of Operations Research* **4** 159–174.

Visee, M., J. Teghem, M. Pirlot, E.L. Ulungu. 1998. Two-phases method and branch and bound procedures to solve the bi-objective knapsack problem. *Journal of Global Optimization* **12** 139–155.

White, D. J. 1982. The set of efficient solutions for multiple-objectives shortest path problems. *Computers and Operations Research* **9** 101–107.

Woeginger, G. J. 2000. When does a dynamic programming formulation guarantee the existence of a fully polynomial time approximation scheme (fptas)? *INFORMS Journal on Computing* **12** 57–75.

Woeginger, G. J. 2005. A comment on schedulling two parallel machines with capacity constraints. *Dicrete Optimization* **2** 269–275.

Wolsey, L. A. 1998. *Integer Programming*. John Wiley & Sons, Inc.

Zeleny, M. 1982. *Multiple criteria problem solving*. McGraw-Hill, New York.

Zitzler, E., L. Thiele. 1999. Multiobjective evolutionary algorithms: A comparative case study and the strength pareto approach. *IEEE Transactions on Evolutionary Computation* **3** 257–271.

Appendices

A. Dantzig-Wolfe Decomposition

By closely looking at the arc flow based formulation of the VRPTW presented in Section 1.2 of Chapter 1, we can observe that only constraints (1.2) are coupling all the vehicles, while the other constraints are dealing with each vehicle separately. Such a structure suggests adopting a Dantzig-Wolfe decomposition (DWD) to break up the overall problem into a *master problem* and a subproblem, also called the *pricing subproblem*, for each vehicle. DWD for linear programs was introduced by Dantzig and Wolfe (1960). For DWD for integer programs see for example Wolsey (1998). In case of the VRPTW, the master problem is a difficult integer programming problem, therefore it is usually relaxed and solved. The pricing subproblem is translated into a constrained shortest path problem. Often, the master problem is stated as a set partitioning problem without describing the underlying DWD on which it is based. In the following, we shortly describe the steps of the DWD. For more details, see, e.g., Barnhard et al. (1998) and Lübbecke and Desrosiers (2005).

Keeping the same notation as presented in Section 1.2 of Chapter 1, we let Ω^k be the set of all feasible paths corresponding to vehicle $k \in K$. In other words, every path $p \in \Omega^k$ corresponds to an elementary path that respects the capacity of vehicle k and time windows constraints, therefore satisfying constraints (1.3)-(1.10). Let x^k_{ijp} be the binary variable that takes the value 1 if and only if the arc between nodes i and j is traversed along path p by vehicle k. Any solution x_{ijk} to the arc flow based formulation presented in Section 1.2 of Chapter 1 can be written as a convex combination of a finite number of elementary paths such that:

$$x_{ijk} = \sum_{p \in \Omega^k} x^k_{ijp} y^k_p \quad \forall k \in K, \forall (i,j) \in A \tag{A.1}$$

where $\sum_{p \in \Omega^k} y^k_p = 1$ and $y^k_p \geq 0$ for all $k \in K$ and $p \in \Omega^k$.

Furthermore, we can define the cost c_p^k of a path, and the number of times σ_{ip}^k a customer i appears in path p traversed by vehicle k as:

$$c_p^k = \sum_{(i,j)\in A} c_{ij} x_{ijp}^k \quad \forall k \in K, \forall p \in \Omega^k \tag{A.2}$$

$$\sigma_{ip}^k = \sum_{j\in V} x_{ijp}^k \quad \forall k \in K, \forall i \in V_c, \forall p \in \Omega^k \tag{A.3}$$

Now substituting (A.2) and (A.3) into (1.1) and (1.2) leads to the following formulation:

$$\min \sum_{k\in K} \sum_{p\in\Omega^k} c_p^k y_p^k \tag{A.4}$$

subject to:

$$\sum_{k\in K} \sum_{p\in\Omega^k} \sigma_{ip}^k y_p^k = 1 \quad \forall i \in V_c \tag{A.5}$$

$$\sum_{p\in\Omega^k} y_p^k = 1 \quad \forall k \in K \tag{A.6}$$

$$y_p^k \geq 0 \quad \forall k \in K, \forall p \in \Omega^k \tag{A.7}$$

The mathematical formulation (A.4)-(A.7) is the LP relaxation of a set partitioning problem. If we assume a set of homogeneous vehicles, the index k can be dropped leading to the classical LP relaxation of the set partitioning formulation that can be written as follows:

$$\min \sum_{p\in\Omega} c_p y_p \tag{A.8}$$

subject to:

$$\sum_{p\in\Omega} \sigma_{ip} y_p = 1 \quad \forall i \in V_c \tag{A.9}$$

$$y_p \geq 0 \quad \forall p \in \Omega \tag{A.10}$$

Because the set Ω of feasible path can be very large, the LP relaxation of the set partitioning formulation (A.8)-(A.10), also called the LP master problem, is usually solved by means of column generation.

B. Column Generation

As we are looking for an integer solution, solving the the LP master problem provides a lower bound on the value of the set partitioning problem. To overcome the huge

number of columns (i.e., paths) in the LP master problem, the formulation (A.8)-(A.10) is solved using only a small subset $\Omega' \subseteq \Omega$ of columns resulting in a *restricted* LP master problem. Usually, we start with columns visiting only one customer, meaning paths with the form *depot-i-depot*, where i is a customer. Generating new columns is done by solving a pricing subproblem by using the information available from the current solution of the restricted LP master problem, more specifically, the vector of dual variables π corresponding to constraints (A.9). In case of the VRPTW, the pricing problem is an Elementary Shortest Path Problem with Resource Constraints (ESPPRC), where the constrained resources are vehicles' capacity and time windows. By modifying the objective function of the pricing problem, we can identify the columns with negative reduced cost which, when added to the restricted LP master problem, improve its objective function. The reduced cost of a column in the pricing problem is

$$\bar{c}_p = c_p - \sum_{i \in V} \sigma_{ip} \pi_i \quad \forall p \in \Omega, \tag{B.1}$$

The reduced cost of a column can also be expressed in the x variables of the original arc based formulation (1.1)-(1.10) as

$$\bar{c}_p = \sum_{(i,j) \in A} (c_{ij} - \pi_j) x_{ij} \quad \forall p \in \Omega \tag{B.2}$$

Therefore, solving the pricing problem correspond to solving the following ESPPRC with the modified cost,

$$\min \bar{c}_p = \sum_{(i,j) \in A} (c_{ij} - \pi_j) x_{ij} \tag{B.3}$$

subject to:

$$\sum_{i \in V_c} x_{0i} = 1 \tag{B.4}$$

$$\sum_{(j,i) \in \gamma^+(j)} x_{ji} = \sum_{(i,j) \in \gamma^-(j)} x_{ij} \qquad \forall j \in V_c \tag{B.5}$$

$$\sum_{i \in V_c} x_{iN+1} = 1 \tag{B.6}$$

$$\omega_i + s_i + \tau_{ij} \leq \omega_j + (1 - x_{ij}) M \qquad \forall (i,j) \in A \tag{B.7}$$

$$a_i \leq \omega_i \leq b_i \qquad \forall i \in V \tag{B.8}$$

$$\sum_{i \in V} q_i \sum_{(i,j) \in \gamma^+(i)} x_{ij} \leq Q \tag{B.9}$$

$$w_i \geq 0 \qquad \forall i \in V \tag{B.10}$$

$$x_{ij} \in \{0,1\} \qquad \forall (i,j) \in A \tag{B.11}$$

The column generation terminates when no columns with negative reduced cost exist, that is, when $\bar{c}_p \geq 0$. However, solving the LP master problem mostly provides a fractional lower bound on the value of the integer master problem. Therefore, column generation is embedded in a *branch-and-bound* framework to guarantee integrality. The gap between the obtained lower bound and the integer optimal value has an important impact on the size of the branching tree. To strengthen the lower bound obtained by solving the LP master problem, *cutting planes* are added to the integer master problem. To ensure integrality, it may be necessary to perform branching. For traditional integer programs branching is usually performed by choosing a fractional variable and create two branches, one where the value of the chosen variable is less than its rounded down value, and another where the value of the variable is greater than its rounded up value. For the VRP, branching is done by setting a fractional variable to 0 for the one branch and to 1 for the other branch. Branch-and-bound frameworks has been used extensively to solve the VRP. The reader is referred to Laporte and Nobert (1987) for a review on the branch-and-bound algorithms proposed in the literature. For a detailed overview of column generation algorithms, the reader is referred to Lübbecke and Desrosiers (2005).

C. Cutting Planes

For general integer programs, cutting planes are valid inequalities that cut off a fractional solution of their LP relaxation without losing any of the feasible integer solutions. In case of the VRPTW, adding cuts to the integer master problem can significantly improve the lower bound obtained by solving the LP master problem resulting in smaller branching trees (i.e., the gap between the obtained lower bound and the optimal solution can be closed easily). Valid inequalities of the original problem (1.1)-(1.10) can be easily reformulated into the integer master problem. In other words, valid inequalities of the original formulation are also valid inequalities for the integer master problem. For the VRPTW a valid inequality is expressed as a linear combination of the original variables, hence it is of the form

$$\sum_{k \in K} \sum_{(i,j) \in A} \alpha_{ij} x_{ijk} \leq \alpha_0, \tag{C.1}$$

or, in case vehicles are identical

$$\sum_{(i,j) \in A} \alpha_{ij} x_{ij} \leq \alpha_0. \tag{C.2}$$

When reformulated into the master problem, inequality (C.2) is rewritten as follows

$$\sum_{p \in \Omega} \sum_{(i,j) \in A} \alpha_{ij} x_{ijp} y_p \leq \alpha_0. \tag{C.3}$$

If $\lambda \leq 0$ is the dual variable corresponding to (C.2), the reduced cost of a column is expressed as follows:

$$\bar{c}_p = \sum_{(i,j)\in A} (c_{ij} - \pi_j - \lambda\alpha_{ij})x_{ijp}. \tag{C.4}$$

Compared to (B.2) an additional coefficient is subtracted from the cost of the edge (i, j). However, the complexity of the pricing problem remains the same if we consider the edge costs $\bar{c}_{ij} = c_{ij} - \pi_j - \lambda\alpha_{ij}$.

Valid inequalities can also be added in the set partitioning formulation (A.8)-(A.10). However, adding valid inequalities in the set partitioning results in a much more complicated pricing problem as the corresponding dual variables can not be expressed in the variables of the original formulation (i.e., the x variables). Consequently, additional resources are needed to handle the additional cost component in the objective function of the pricing problem. For more detail, see for instance Jespen et al. (2008).

Summary

Time and Multiple Objectives in Scheduling and Routing Problems

Many optimization problems encountered in practice are multi-objective by nature, i.e., different objectives are conflicting and equally important. Many times, it is not desirable to drop some of them or to optimize them in a composite single objective or hierarchical manner. Furthermore, cost parameters change over time which makes optimization problems harder. For instance, in the transport sector, travel costs are a function of travel time which changes depending on the time of the day a vehicle is travelling (e.g., due to road congestion). Road congestion results in tremendous delays which lead to a decrease in the service quality and the responsiveness of logistic service providers.

In Chapter 2, we develop a generic approach to deal with Multi-Objective Scheduling Problems (MOSPs) with State-Dependent Cost Parameters. The aim is to determine the set of Pareto solutions that capture the trade offs between the different conflicting objectives. Due to the complexity of MOSPs, an efficient approximation based on dynamic programming is developed. The approximation has a provable worse case performance guarantee. Even though the generated approximate Pareto front consist of fewer solutions, it still represents a good coverage of the true Pareto front. Furthermore, considerable gains in computation times are achieved.

In Chapter 3, the developed methodology is validated on the multi-objective time-dependent knapsack problem. In the classical knapsack problem, the input consists of a knapsack with a finite capacity and a set of items, each with a certain weight and a cost. A feasible solution to the knapsack problem is a selection of items such that their total weight does not exceed the knapsack capacity. The goal is to maximize the single objective function consisting of the total profit of the selected items. We extend the classical knapsack problem in two ways. First, we consider time-dependent profits (e.g., in a retail environment profit depends on whether it is Christmas or not).

Secondly, we consider the joint optimization of a multi-objective cost function.

In Chapter 4, we subject a Vehicle Routing Problem with Time Windows to traffic congestion. In other words, we consider time varying travel times rather than the standard constant travel time setting. Moreover, we assume a non-increasing demand over time. The goal is to jointly minimize the total time travelled and maximize the total demand fulfilled. In this chapter, we assume that the customer sequence is predefined. However, we show that the problem at hand is still NP-hard, and we develop a slightly different approximation algorithm to generate good approximate Pareto fronts.

In Chapter 5, we develop the first exact algorithm for the Time-Dependent Vehicle Routing Problem with Time Windows (TDVRPTW). The goal is to determine the set of routes with the least total duration. The corresponding optimization problem entails a routing component (i.e., deciding on the customer sequence) and a scheduling component (i.e., deciding on the vehicles' dispatch time at the depot). We design a Branch-and-Cut-and-Price framework where the master problem (i.e., the set partitioning problem) is solved by means of column generation and the pricing problem (i.e., the time-dependent elementary shortest path problem with resource constraints) is solved using a time-dependent labeling algorithm.

About the author

Said Dabia was born in Oujda, a city on the north east of Morocco, on June 3, 1977. He received his Bachelor's degree (1998) in Mathematics and Physics from the center of *Classes Préparatoires aux Grandes Écoles*, Oujda, Morocco. In 2001, he received his *Ingenieur D'État* degree in industrial engineering from *l'École Supérieure des Industries du Textile et de l'Habillement*, Casablanca, Morocco. Said worked for several years as logistic engineer. In 2005, he started with the Master program Operations Management and Logistics at Eindhoven University of Technology, Eindhoven, The Netherlands. He graduated with a Master project on capacitated inventory systems under the supervision of Gudrun Kiesmüller and Nico Dellaert. In 2007, Said started his PhD research with a project on time and multiple objectives in scheduling and routing problems, under the supervision of Ton de Kok and Tom van Woensel. He carried out a part of his research at the Technical University of Denmark where he collaborated with Stefan Ropke throughout April-July, 2010. On January 9, 2012, Said defends his PhD thesis at Eindhoven University of Technology. From September 1, 2011, Said is product development manager at Eyefreight.