Contributions to Column-Generation Approaches in Combinatorial Optimization

Dissertation
zur Erlangung des Grades eines Doktors der
wirtschaftlichen Staatswissenschaften
(Dr. rer. pol.)
des Fachbereichs Recht- und Wirtschaftswissenschaften
der Johannes Gutenberg-Universität Mainz

vorgelegt von
Dipl.-Math. Christian Tilk
in Mainz

vorgelegt im Jahre 2016

Tag der mündlichen Prüfung: 14.12.2016

Contents

List of Papers

- Christian Tilk[1], Stefan Irnich[1] (2016). "Dynamic Programming for the Minimum Tour Duration Problem". *Transportation Science*. Forthcoming. `http://dx.doi.org/10.1287/trsc.2015.0626`

- Christian Tilk, Nicola Bianchessi[1,2], Michael Drexl[1,3], Stefan Irnich[1], Frank Meisel[4] (2016). "Branch-and-Price-and-Cut for the Active-Passive Vehicle-Routing Problem". *Transportation Science*. Forthcoming.

- Christian Tilk[1] (2016). "Branch-and-Price-and-Cut for the Vehicle Routing and Truck Driver Scheduling Problem". Technical Report LM-2016-04, Chair of Logistics Management, Johannes Gutenberg University Mainz, Mainz, Germany. *Submitted to OR Spectrum*

- Christian Tilk[1], Timo Gschwind[1], Ann-Kathrin Rothenbächer[1], Stefan Irnich[1] (2016). "Asymmetry Helps: Dynamic Half-Way Points for Solving Shortest Path Problems with Resource Constraints Faster". Technical Report LM-2016-05, Chair of Logistics Management, Johannes Gutenberg University Mainz, Mainz, Germany. *Submitted to European Journal of Operational Research*

- Christian Tilk[1], Stefan Irnich[1] (2016). "Combined Column-and-Row-Generation for the Optimal Communication Spanning Tree Problem". Technical Report LM-2016-03, Chair of Logistics Management, Johannes Gutenberg University Mainz, Mainz, Germany. *Submitted to European Journal of Operational Research*

[1]Johannes Gutenberg-Universität Mainz, Lehrstuhl für BWL insb. Logistikmanagement, Mainz, Germany
[2]Department of Information Engineering, University of Brescia, Italy
[3]Fraunhofer Centre for Applied Research on Supply Chain Services SCS, Nuremberg, Germany
[4]Chair of Supply Chain Management, Faculty of Business, Economics and Social Sciences, University Kiel, Germany

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Since several decades column-generation algorithms were successfully applied for solving a wide variety of combinatorial optimization problems (Lübbecke and Desrosiers, 2005). Most such algorithms rely on a Dantzig-Wolfe decomposition of a problem-specific compact integer formulation (Dantzig and Wolfe, 1960). Therein, the original variables are redefined resulting in a so-called (integer) master program that typically contains an exponential number of variables. The basic idea of column generation is to work on a *restricted master program* (RMP) that considers only a reasonably small subset of variables, since there are often too many for explicitly representing the model and most of them will have a value equal to zero in an optimal solution anyway. Missing negative reduced-cost variables (=columns) are added dynamically to the model by solving the so-called pricing problem (=subproblem). The solution of the linear relaxation of the master program is obtained by alternating between re-optimizing the restricted master program and solving the subproblem as long as no more variables with negative reduced cost exist. In case the master problem contains integrality constraints on some of its variables, column generation and branch-and-bound are combined resulting in a so-called branch-and-price algorithm (Barnhart *et al.*, 1998).

The advantages of column-generation based approaches over a compact formulation can mainly be attributed to the following two aspects: First, a compact formulation of a mixed-integer program may have a weak linear relaxation and/or a symmetric structure that causes branch-and-bound to perform poorly because the problem barely changes after branching. A reformulation with a huge number of variables can eliminate these issues (Barnhart *et al.*, 1998). In particular, the linear relaxation of the master program generally produces tighter lower bounds than those obtained with the linear relaxation of the compact formulation if the formulation of the subproblem does not have the integrality property (Lübbecke and Desrosiers, 2005). Second, complicating constraints or even non-linearities that are present in the compact formulation may be hidden in the subproblem in column-generation approaches. Moreover, highly effective solution procedures for the subproblem are often available (Lübbecke and Desrosiers, 2005). The main drawbacks of column-generation algorithms are related to instability issues like, e.g., the heading-in effect (generation of irrelevant columns in the first iterations because of poor dual information), the plateau effect (no improvement of the RMP value for several iterations), the tailing-off effect (slow convergence of the column-generation process), and the oscillation of the duals (Vanderbeck, 2005).

The thesis at hand deals with column-generation approaches for four different combinatorial optimization problems: The *minimum tour duration problem* (MTDP), the

active-passive vehicle routing problem (APVRP), the *vehicle routing and truck driver scheduling problem* (VRTDSP), and the *optimal communication spanning tree problem* (OCSTP). While the first three problems belong to the family of *vehicle routing problems* (VRPs), the last one is in the family of constrained spanning tree problems. Moreover, we deal with the *shortest path problem with resource constraints* (SPPRC) that often occurs as a subproblem in column-generation approaches for VRPs.

We will now introduce the five aforementioned optimization problems before we state the contributions of this thesis. VRPs have been the subject of intensive study for more than half a century now. Different variants of the VRP are studied in more than thousands of scientific papers and a lot of commercial vehicle routing software is available. Standard variants of the VRP, like the *vehicle routing problem with time windows* (VRPTW), are well studied and effective solution algorithms can be found in the literature (see, e.g., Chapter 5 in Toth and Vigo, 2014). But research on more complex VRP variants is constantly ongoing, motivated by unsolved theoretical problems as well as continuous input from logistics practice (Drexl, 2012a).

The MTDP is a variant of the traveling salesman problem with time windows, where the objective is the minimization of the tour duration instead of cost minimization. Strictly speaking, the MTDP does not belong to the family of VRPs. However, it can be seen as a VRP with a single vehicle. The minimization of the tour duration and cost minimization are generally conflicting objectives (Vidal *et al.*, 2011). Savelsbergh (1992) pointed out that it is important to be able to handle them. Indeed, vehicles are often a scarce resource so that their cost is a function of the time the vehicle is in use. From a practical point of view, it is also important, since truck drivers are often paid according to their working time.

One generic class of VRPs that is receiving more and more interest is denoted as *vehicle routing problems with multiple synchronization constraints* (VRPMSs): In classical VRPs, synchronization is necessary between the vehicles with respect to which vehicle visits which customer. VRPMSs has additional synchronization requirements with regard to spatial, temporal, and load aspects (Drexl, 2012b). Many applications in the area of transport logistics involve problems in which the execution of transport requests calls for a joint operation of different resources such as trucks, tractors, drivers, trailers, semi-trailers, swap bodies, containers, or accompanying staff. A typical example of a VRPMS is the APVRP where not just a manned truck but also an empty container is required for executing a request.

Another rising field of research is the simultaneous planning of vehicle routes and truck driver schedules since many governments worldwide have imposed hours of service regulations for truck drivers to avoid fatigue-related accidents. These regulations ensure that break and rest periods are regularly taken, i.e., they define a minimum amount of break and rest times for truck drivers as well as a maximum driving time between two break or rest periods. In the VRTDSP, the current hours of service regulations in the United States are included in a VRPTW (Goel and Irnich, 2016).

Constrained tree optimization problems arise in many applications, pose significant modeling and algorithmic challenges, are building blocks for constructing many complex models, and provide a concrete setting for illustrating many key ideas from the field of

combinatorial optimization (Magnanti and Wolsey, 1995). The OCSTP is a constrained spanning tree problem with the objective to minimize the sum of the communication costs between all pairs of vertices. Besides a broad range of applications in designing telecommunication and transportation networks, the OCSTP occurs as a subproblem in some network hub location problems (Contreras and Fernández, 2012).

Finally, the SPPRC consists in finding a shortest path from a source to a sink that satisfies a set of constraints defined over a set of resources (Irnich and Desaulniers, 2005). It was introduced by Desrochers (1986) as a subproblem of a bus driver scheduling problem. Since formulations of the SPPRC does not possess the integrality property, column-generation formulations with SPPRC subproblems have generally tighter linear relaxation bounds than those obtained with a compact formulation. The flexibility in the design of the resource constraints allows modeling complex cost structures (e.g., tour duration minimization) and a wide variety of feasibility constraints (e.g., hours of service regulations).

1.1 Contribution

The contribution of this thesis is to develop effective column-generation algorithms that fully exploit the individual, problem-specific structures. This task can be roughly divided into two subtasks: Defining appropriate Dantzig-Wolfe decompositions and designing effective algorithms to solve the subproblems. In VRPs, the master program is typically a (possible extended) set partitioning formulation and the subproblem a SPPRC that is mostly solved with a dynamic programming based labeling algorithm. Whereas constrained spanning tree problems can be decomposed in many different ways resulting in quite different master programs and subproblems. Next, we elaborate on the contributions of each single paper in this thesis. For more details, the reader is referred to the particular chapters.

For dealing with the MTDP, we provide a new labeling algorithm with three resources for which effective dominance and bounding procedures are applicable. In particular, we define consistent *resource extension functions* (REFs) so that the labeling can be reversed consistently, and such that the forward labeling or any of its relaxations provides bounds for the backward labeling and vice versa. These bounds are strengthened by using a penalty method (Christofides *et al.*, 1981), where good penalties are generated using column generation. Moreover, two new relaxations are defined and the concept of combining different relaxations is discussed.

For the APVRP, we present, to the best of our knowledge, the first algorithm that considers explicitly the temporal interdependencies between active and passive vehicles. The algorithm is based on a non-trivial network representation that models the logical relationships between the different transport tasks necessary to fulfill a request as well as the synchronization of the movements of active and passive vehicles. Moreover, we contribute to the development of branch-and-price methods in general, in that we solve, for the first time, a *ng*-path relaxation of a pricing problem with linear vertex costs by means of a bidirectional labeling algorithm.

Next, we present a sophisticated branch-and-price-and-cut algorithm for the VRTDSP that is based on the work of Goel and Irnich (2016). We extend their labeling algorithm by means of defining a bidirectional labeling with a non-trivial merge procedure. Different acceleration techniques are used to speed up the solution process of the pricing problem. In addition, different families of known valid inequalities for the VRPTW are adapted and used to further strengthen the linear relaxation of the master program. A detailed computational study shows the impact of the different speed-up techniques.

Our next contribution focuses on the acceleration of labeling algorithms for SPPRCs. Righini and Salani (2006) and several subsequent works have shown that bounded bidirectional labeling algorithms are often superior to their monodirectional counterparts. However, the bidirectional labeling is often very much unbalanced, e.g., due to asymmetric forward and backward labeling procedures or even due to asymmetric instance data. We exploit this asymmetry in forward and backward label extensions and introduce a so-called dynamic half-way point. We demonstrate the impact of using dynamic halfway points by experiments with the VRPTW, the electric vehicle routing problem with time windows, and the VRTDSP.

When solving the OCSTP with column generation, we decompose two different compact formulations, resulting in two completely different master programs and subproblems. Due to the huge number of variables and constraints in both formulations, we apply combined column-and-row generation. Moreover, one subproblem has a formulation with integrality property, while the other has not. Hence, there is the tradeoff that the former subproblems can be solved much faster while the latter generally lead to a stronger linear relaxation (Lübbecke and Desrosiers, 2005). Our main goal is the comparison of both reformulations to decide which of the column-and-row-generation algorithms are more effective from a computational point of view.

1.2 Outline

This thesis by publication comprises five articles that have either been published in or submitted to scientific journals. The remainder of the thesis is organized as follows. Chapter 2 addresses the new labeling procedure for the MTDP. Section 2.2 briefly surveys exact solution algorithms to the MTDP and related problems. In Section 2.3, we present our new labeling formulation. The computation of upper bounds with the help of a heuristic is presented in Section 2.4. Section 2.5 discusses different relaxations including the two new ones. Moreover, a relaxation-based bounding procedure is presented. In order to further improve the lower bounds, we apply a penalty method in Section 2.6. Section 2.7 reports the computational results. Concluding remarks are given in Section 2.8.

Chapter 3 focuses on presenting a new and effective branch-price-and-cut algorithm for the APVRP. Related literature is reviewed in Section 3.2. A formal description of the APVRP is given in Section 3.3. The corresponding extended set-partitioning formulation is provided in Section 3.4. The branch-and-price-and-cut algorithm including the labeling algorithm is presented in Section 3.5, and the method is computationally eval-

uated in Section 3.6. Finally, Section 3.7 summarizes the paper and discusses potential avenues for further research.

In Chapter 4, the work on the VRTDSP is presented. Section 4.2 briefly reviews the literature on VRPs with hours of service regulations. In Section 4.3, the extensive formulation is given. Section 4.4 presents the pricing problem and its solution. Families of valid inequalities are discussed in Section 4.5. Section 4.6 presents a computational analysis of the algorithmic components. Concluding remarks are given in Section 4.7.

Chapter 5 deals with the dynamic half-way point. In Section 5.2, we formally describe bidirectional labeling for SPPRCs while emphasizing the differences of using a static or a dynamic half-way point. The VRPTW, the electric vehicle routing problem with time windows, and the VRTDSP are briefly described in Section 5.3. The impact of using a dynamic half-way point is computationally evaluated in Section 5.4. Finally, concluding remarks are given in Section 5.5.

In Chapter 6, two column-and-row generation algorithms for the OCSTP are presented. Section 6.2 briefly surveys exact solution algorithms for the OCSTP. The formal definition of the OCSTP and two compact formulations are presented in Section 6.3. In Section 6.4, we describe the new extensive formulations to be solved with column-and-row-generation algorithms. Moreover, we formalize the pricing problems and discuss algorithms for their effective solution. Section 6.5 reports computational results. Conclusions are drawn in Section 6.6.

Chapter 2

Dynamic Programming for the Minimum Tour Duration Problem

Christian Tilk, Stefan Irnich

Abstract

The minimum tour duration problem (MTDP) is the variant of the traveling salesman problem with time windows, which consists of finding a time window-feasible Hamiltonian path minimizing the tour duration. We present a new effective dynamic programming (DP)-based approach for the MTDP. When solving the traveling salesman problem with time windows with DP, two independent resources are propagated along partial paths, one for costs and one for earliest arrival times. For dealing with tour duration minimization, we provide a new DP formulation with three resources for which effective dominance and bounding procedures are applicable. This is a non-trivial task because in the MTDP at least two resources depend on each other in a non-additive and non-linear way. In particular, we define consistent resource extension functions (REF) so that dominance is straightforward using componentwise comparison for the respective resource vectors. Moreover, one of the main advantages of the new REF definition is that the DP can be reversed consistently such that the forward DP or any of its relaxations provides bounds for the backward DP, and vice versa. Computational tests confirm the effectiveness of the proposed approach.

2.1 Introduction

In this paper, we consider a variant of the *traveling salesman problem with time windows* (TSPTW), in which the objective is the minimization of the tour duration. There is no consistent naming of this problem in the literature. We will use the name *minimum tour duration problem* (MTDP) in the following and start with its definition. Let $G = (V, A)$ be a digraph with node set V and arc set A. Two distinguished nodes are given, the start node $o \in V$ and the destination node $d \in V$. A *Hamiltonian o-d-path* is a simple path which start at o, ends at d, and visits all nodes exactly once. A travel time $t_{ij} > 0$ is associated with each arc $(i, j) \in A$ and a time window $[a_i, b_i]$ with each node $i \in V$. For any path $P = (i_0, i_1, \ldots, i_p)$, a sequence $T = (T_0, T_1, \ldots, T_p)$ of numbers is called a

schedule. A schedule with $T_k \in [a_{i_k}, b_{i_k}]$ for all $k \in \{0, 1, \ldots, p\}$ and $T_{k-1} + t_{i_{k-1},i_k} \leq T_k$ for all $k \in \{1, \ldots, p\}$ is called *feasible*. A *TSPTW tour* is a Hamiltonian o-d-path P for which a feasible schedule exists. We say that T_k is the *time when service starts* at node i_k. Note that we do not explicitly consider *service times* because they can be included in the travel times t_{ij}. The MTDP is the problem of finding a TSPTW tour P with a feasible schedule $T = (T_o, \ldots, T_d)$ minimizing $T_d - T_o$.

In contrast, the objective in the TSPTW is minimizing the arc-traversal costs for given arc costs c_{ij} for $(i, j) \in A$. There exists a third problem related to minimizing the completion time T_d. We call this problem *minimum completion time problem* (MCTP). It is the special case of the MTDP, in which the starting time T_o is fixed. TSPTW, MTDP, and MCTP only differ in their objectives, and these objectives are generally conflicting (Vidal *et al.*, 2011). Savelsbergh (1992) pointed out that it is important to be able to handle them. Indeed, vehicles are often a scarce resource so that their cost is a function of the time the vehicle is in use. Moreover, depending on the drivers' contracts, they may be paid according to their working time. The tour duration we consider here is, however, only an approximation of the working times because hours of service regulations require to schedule additional break and rest times (see, e.g., Goel and Vidal, 2014).

The contribution of the paper at hand is to present a new effective dynamic-programming (DP) algorithm for the MTDP. It generalizes the approach presented by Baldacci *et al.* (2012a), who solve the TSPTW with a DP-based algorithm. Their algorithm solves all but one instance of the known benchmark sets for the TSPTW and outperforms all exact methods published in the literature so far. When solving the TSPTW, two independent resources are propagated along partial paths, one for costs and one for earliest arrival times. For dealing with tour duration minimization, there exist several possibilities to define and propagate resources. However, all these alternatives use at least three resources. We will follow ideas presented in (Irnich, 2008) in order to apply effective dominance and bounding procedures. This is a non-trivial task because in the MTDP at least two resources depend on each other in a non-additive and non-linear way. In particular, we define consistent resource extension functions (REFs, see Desaulniers *et al.*, 1998) so that dominance is straightforward using componentwise $\leq$ for the respective resource vectors. Moreover, one of the main advantages of the new REF definition is that the DP can be reversed consistently such that the forward DP or any of its relaxations provides bounds for the backward DP, and vice versa.

Using relaxations to obtain lower bounds is common practice in routing problems, e.g., using a state-space relaxation (Christofides *et al.*, 1981). We present two new relaxations for the MTDP with only one and two resource, respectively, which are attractive due to their low computational complexity. These and other relaxations can be combined with the ng-tour and ngL-tour relaxation (Baldacci *et al.*, 2011, 2012a).

To compute tight lower bounds, we use two methods: First, we adapt a penalty method first suggested by Christofides *et al.* (1981). Second, we generate the neighborhoods for the ng-tour and ngL-tour relaxations dynamically. This technique has been successfully applied for solving different routing problems (Roberti and Mingozzi, 2014; Bode and Irnich, 2015). To the best of our knowledge, we present the first exact algorithm for

the MTDP. We provide computational results with optimal solutions for many known benchmark instances, which were originally provided for the TSPTW.

Our proposed DP-algorithm for the MTDP uses several building blocks that were already used in the DP-algorithm of Baldacci *et al.* (2012a) for the TSPTW. However, since the MTDP is more complex and uses three resources, there exist many more reasonable possibilities to define and combine different relaxations, either by relaxing elementarity or resource feasibility. We see our contribution also in the identification of those combinations of relaxations that prove to be effective for the MTDP. As a result, building blocks such as the penalty method and bounding procedures benefit from the insights we gain from the in-depth analysis of the different relaxations.

This paper is structured as follows. Section 2.2 briefly surveys exact solution algorithms to the TSPTW, MTDP, and MCTP. In Section 2.3, we present our new DP formulation for the MTDP. The computation of upper bounds with the help of a heuristic is presented in Section 2.4. Section 2.5 discusses relaxations, namely the adapted ng-tour and ngL-tour relaxations and two new relaxations with one and two resources, respectively. Moreover, a relaxation-based bounding procedure is presented. In order to further improve the lower bounds, we apply a penalty method in Section 2.6. Section 2.7 reports the computational results. Concluding remarks are given in Section 2.8.

2.2 Literature Review

The TSPTW and MCTP are often discussed problems in the literature, but only little attention was dedicated to the MTDP. Christofides *et al.* (1981) proposed a method for solving the MCTP based on state-space relaxation. They reported solutions of instances with up to 50 nodes. Baker (1983) proposed a branch-and-bound method for the MCTP producing exact solutions for instances with up to 50 nodes. Langevin *et al.* (1993) introduced a two-commodity flow formulation for the MCTP and TSPTW and developed a branch-and-bound algorithm that is able to solve instances with up to 60 nodes. Their model can easily be adapted to the MTDP.

Several DP approaches were proposed for the TSPTW: Dumas *et al.* (1995) presented a DP algorithm and advanced preprocessing procedures to reduce the number of states and state transitions. They computed solutions of instances with up to 200 nodes, but relatively tight time windows. The DP algorithm of Mingozzi *et al.* (1997) is based on another state-space relaxation. They solved the TSPTW with precedences for instances with up to 120 nodes. Li (2009) solved the TSPTW with a bi-directional resource-bounded label correcting algorithm. This algorithm is able to solve instances with up to 233 nodes. Recently, Baldacci *et al.* (2012a) presented the ng-tour and ngL-tour relaxations for the TSPTW to compute tight lower bounds. To improve the lower bounds they solve the dual of a problem that seeks a minimum-weight convex combination of non-necessarily elementary tours with a dual-ascent heuristic. Their computational results are impressive: All but one instances from several TSPTW benchmark sets are solved to proven optimality.

Balas and Simonetti (2001) presented a DP model and algorithm for a special case of the TSPTW and the MCTP, which can also be applied as a heuristic for the general case. Ascheuer *et al.* (2001) were the first to develop branch-and-cut algorithms for the TSPTW. They implemented three alternative integer programming formulations of the TSPTW and solved instances with up to 233 nodes. Dash *et al.* (2012) presented an extended formulation of the TSPTW based on partitioning the time windows into buckets. The LP-relaxation of their formulation provides strong lower bounds, which they exploited in a branch-and-cut algorithm.

To the best of our knowledge, (Savelsbergh, 1992) is the first paper dealing with the MTDP. Savelsbergh described edge-exchange improvement methods for the MTDP and the VRPTW with the objective of minimizing the route duration. The only recent papers dealing with the MTDP are an ant-colony approach by Favaretto *et al.* (2006) and a new two-commodity flow formulation by Kara *et al.* (2013). Favaretto *et al.* (2006) call the problem *temporal TSPTW* and present computational results for the MTDP on benchmark instances originally proposed for the TSPTW. Kara *et al.* (2013) solved their model with the MIP solver CPLEX to optimality for instances with up to 40 nodes.

The problem of minimizing the tour duration also occurs as a subproblem in truck driver scheduling and routing when the driving time is limited, e.g., by hours of service regulations as studied by Goel (2009). He presented an exact method for scheduling driving periods, breaks, rest periods and handling activities for a given tour based on a labeling algorithm. Among others, Goel and Vidal (2014) studied similar problems for regulations of different countries. They presented an algorithm that combines population-based metaheuristics with a local search that uses forward labeling procedures for checking compliance with complex hours of service regulations. Prescott-Gagnon *et al.* (2010) solve a vehicle-routing problem with time windows and European driver rules. They used a column-generation approach, which utilizes a tabu search to heuristically solve the subproblem. To check the route feasibility they model all feasibility rules as resource constraints and develop a label-setting algorithm to perform this check.

2.3 Dynamic-Programming Formulations

In this section, we present an exact DP formulation for the MTDP. First, we will introduce the REFs for the forward propagation and the corresponding DP recursion. Second, we will present propagation rules that limit the number of possible extensions and fathoming rules that eliminate labels which cannot lead to a feasible or optimal solution. Last, we define consistent REFs for the backward propagation and show how paths of the forward and the backward formulation can be concatenated. Consistency refers at least to the following five aspects:

1. Using the same set of resources, the MTDP can either be solved using forward or backward propagation. Both resulting DP algorithms have identical worst-case complexity.

2. Let $P = (o, \ldots, i)$ be a path resulting from forward propagation, and let $P^{bw} = (i, \ldots, d)$ be a path resulting from backward propagation. Moreover, let $L = L(P)$ and $L^{bw} = L^{bw}(P^{bw})$ be the corresponding labels that result from forward and backward propagation, respectively. Then, the concatenation $P \oplus P^{bw}$ is feasible w.r.t. resource consumption if and only if $L \leq L^{bw}$ holds.

3. If $P \oplus P^{bw}$ is feasible w.r.t. resource consumption, the labels allow the computation of the minimum tour duration in constant time.

4. The above label comparison enables the bidirectional solution of the MTDP by combined forward and backward propagation.

5. If an upper bound on the MTDP is known, the above label comparison also enables the use of bounding techniques: A backward label provides a valid bound for the forward label, and vice versa. Moreover, any label resulting from a relaxation may be used for bounding instead of a label produced with the exact DP.

2.3.1 Forward Dynamic-Programming Formulation

A *forward path* $P = (o, \ldots, i)$ defines a *forward label* (k, i, S, T), where $T = (T^{time}, T^{dur}, T^{help})$ with the following semantics:

- $i \in V$ is the last visited node in P
- $S \subseteq V$ is the set of all nodes visited by the path
- k is the path length, i.e., $k = |P| = |S| - 1$ (while k can be derived from S here, we will later use relaxations, for which this is not the case)
- T^{time} is the earliest feasible time, at which the last node i can be visited in the path
- T^{dur} is the minimum tour duration of the path starting at node o, visiting the set $S \subset V$ and ending at node i so that every node is visited within its time window
- T^{help} is the negative of the latest possible departure time at node o so that the time window constraints of every node in the path are satisfied and the tour duration is T^{dur}

While k, i, and S are standard attributes that can describe any partial path, the resource vector $T = (T^{time}, T^{dur}, T^{help})$ consists of the three actual resources that are specific for the MTDP. Following the description in (Irnich, 2008, Sect. 2.4.4), we will now define resource windows, the initial label for the path $P = (o)$, and REFs for forward propagation.

The feasible values of the resources $T_i = (T_i^{time}, T_i^{dur}, T_i^{help})$ at node $i \in V$ are given by the following resource windows:

$$T_i^{time} \in [a_i, b_i] \tag{2.1a}$$
$$T_i^{dur} \in [0, UB], \text{where } UB \text{ is any upper bound on the MTDP (tour duration)} \tag{2.1b}$$
$$T_i^{help} \in [-b_o, \infty) \tag{2.1c}$$

The earliest possible time a Hamiltonian o-d-path can start is a_o and the latest is b_o. Therefore, the initial forward label for $P = (o)$ is defined as $(0, o, \{o\}, T_o)$ with $T_o = (T_o^{time}, T_o^{dur}, T_o^{help}) = (a_o, 0, -b_o)$. Forward propagation of a label (k, i, S, T_i) along an arc $(i, j) \in A$, i.e., towards node j produces the new label $(k+1, j, S \cup \{j\}, T_j)$. Herein, T_j results from the following REFs:

$$T_j^{time} = f_{ij}^{time}(T_i) := \max\{T_i^{time} + t_{ij}, a_j\} \tag{2.2a}$$
$$T_j^{dur} = f_{ij}^{dur}(T_i) := \max\{T_i^{dur} + t_{ij}, T_i^{help} + a_j\} \tag{2.2b}$$
$$T_j^{help} = f_{ij}^{help}(T_i) := \max\{T_i^{dur} + t_{ij} - b_j, T_i^{help}\} \tag{2.2c}$$

Note that the third resource T^{help} has been defined to be negative so that a non-decreasing REF results. Desaulniers *et al.* (1998) were among the first who explicitly stressed the high importance of non-decreasing REFs, i.e., functions where $S \leq T$ implies $f(S) \leq f(T)$. If resources are propagated with non-decreasing REFs, standard dominance with componentwise $\leq$-comparison is valid.

The resources T_i^{dur} and T_i^{help} are interdependent. This represents, on the one hand, that when we must wait at a node j, we can shift the start time to avoid an increase of the tour duration, as long as the schedule stays feasible. On the other hand, the possible shift can be limited by the difference of the tour duration and the latest service start b_j associated with node j.

We conclude that $T_i^{time} - T_i^{dur}$ is the earliest feasible departure time at node o that avoids unnecessary waiting. Hence, $-T_i^{help} - T_i^{time} + T_i^{dur}$ is the possible amount of time that we may shift the earliest possible departure time in direction b_o. In addition $-T_i^{help} + T_i^{dur}$ is the latest feasible arrival time at node i when the path duration is T_i^{dur}.

Resources and their update are also important in local search when the feasibility or duration of a tour has to be evaluated for a neighborhood move. Kindervater and Savelsbergh (1997) introduced three attributes for total travel time, earliest departure time at the last node, and latest arrival time at the first node in their search approach. However, the respective update functions just work mono-directionally for the forward propagation and they are not non-decreasing. Recently, Vidal *et al.* (2013) introduced resources for the duration, time warp use, and the earliest and latest service time at the start of the tour in order to compute the tour duration in amortized constant time. These resources are obviously related to our resources. However, both works use resources that do *not* follow the generic definition of non-decreasing REFs as coined in (Irnich, 2008) which allows a direct inversion of the propagation process for bounding or bidirectional labeling.

Example 2.1. We consider a path $P = (0, 1, 2, 3, 4)$ with $o = 0$ and $d = 4$ and with time windows $[a_j, b_j]$ and travel times $t_{j,j+1}$ given in the first three columns of the following table:

Node j	Time window $[a_j, b_j]$	Travel time $t_{j,j+1}$	Resources T_j^{time}	T_j^{dur}	T_j^{help}
0	[0, 6]	1	0	0	-6
1	[2, 5]	2	2	1	-4
2	[5, 6]	3	5	3	-3
3	[11, 12]	4	11	8	-3
4	[14, 18]	–	15	12	-3

The (minimum) tour duration of the path $P_1 = (o, 1)$ coincides with the sum of travel times because waiting can be avoided by starting at any time between 1 and 4. The latest possible departure time is limited due to b_1. The path $P_2 = (o, 1, 2)$ limits the latest possible departure time in the same manner as the path P_1. The tour duration of the path $P_3 = (o, 1, 2, 3)$ consists of six units of travel time and two units of waiting time. The waiting time cannot be avoided because starting later than at time $T_o = 3$ violates the due date b_2 and the earliest time to arrive at node 3 is $a_3 = 11$. Next, the path $P_4 = (o, 1, 2, 3, 4)$ has the tour duration twelve, which comprises the sum of travel times and two units of waiting time at node 3 before a_3. The latest possible departure time of this path is three, which is limited due to b_2.

Before we start the actual DP algorithm, we modify the instance to obtain an equivalent one with fewer arcs, tighter time windows, and a set of precedences. This kind of *preprocessing* was originally suggested by Desrosiers *et al.* (1995). We iterative compute and update two types of values. $EAT(i, j)$ is the earliest feasible arrival time at node j when coming from node i, and $LDT(i, j)$ is the latest feasible departure time from node i when going to node j. Furthermore, the time window constraints impose a partial ordering of the nodes, which we use to identify node precedences: For all $j \in V$ the set $\pi(j)$ is the set of all nodes $i \in V$ that must precede j. The precedences $\pi(j)$, time windows $[a_i, b_i]$, times $EAT(i, j)$ and $LDT(i, j)$, and the (reduced) arc set A mutually affect each other (see Desrosiers *et al.* (1995) for details). Therefore, the computation should be iterated until no more modifications are made. Note that preprocessing generally reduces the possible extensions of labels, that preprocessed instances are sometimes significantly smaller, and have stronger relaxations so that they are in the end easier to solve.

The forward DP is described in Algorithm 2.1. Herein, all labels are grouped according to the length k of the corresponding path, and $\mathcal{L}_k$ denotes this set. In the following, we comment on the components of Algorithm 2.1: The forward propagation (Step 4) always creates feasible partial paths. In particular, only arcs $(i, j) \in A$ from the preprocessed instance are allowed, the partial path must be elementary (ensured by $j \notin S$), and must respect all precedences ($\pi(j) \subseteq S$). In the feasibility check (Step 6), we first compare $T_j = (T_j^{time}, T_j^{dur}, T_j^{help})$ against the upper bounds $(b_j, UB, -a_o)$ given by (2.1).

Algorithm 2.1: Forward Dynamic Programming Labeling Algorithm

```
1  SET L_0 := {(0, o, {o}, (a_o, 0, -b_o))}
2  for k = 0, 1, ..., |V| - 2 do
3      for (k, i, S, T_i) ∈ L_k do
4          for (i, j) ∈ A : j ∉ S, π(j) ⊆ S do
5              SET T_j := f_ij(T_i)
6              if FeasibilityCheck(T_j) then
7                  SET L_j := (k + 1, j, S ∪ {j}, T_j)
8                  if BoundingCheck(L_j) then
9                      ADD L_j to L_{k+1}
10     CALL Dominance algorithm for L_{k+1}
11 FIND a label L*_d = (|V| - 1, d, V, T_d) ∈ L_{|V|-1} with T_d^dur minimal
   Result: The path P* represented by label L*_d
```

Moreover, there might not exist an extension to a feasible TSPTW tour due to resource consumption. We apply the following rule:

Rule 2.1. *(Feasibility) A label (k, i, S, T_i) cannot lead to a feasible solution if there exists a node $j \in V \setminus S$ with $T_i^{time} > LDT(i, j)$. Hence, such a label can be discarded.*

Bounding procedures try to identify those labels, for which any extension to a feasible TSPTW tour will only create a non-optimal tour. In case of the MTDP, non-optimality refers to the tour duration measured with the help of the resource T_i^{dur}. It is obvious that a label (k, i, S, T_i) cannot lead to an optimal solution if the earliest service time a_d at the destination node d minus the label's latest possible departure time at o without unnecessary waiting $-T_i^{help}$, i.e., $a_d + T_i^{help}$ is greater than an upper bound. In the following we assume that an upper bound UB on MTDP is known. Step 8 of Algorithm 2.1 uses the following rule:

Rule 2.2. *(Bounding) Let UB be an upper bound for the MTDP. A label (k, i, S, T_i) cannot lead to an improved solution if $a_d + T_i^{help} \geq UB$. Hence, it can be discarded.*

A *dominance algorithm* eliminates those labels whose final extension to the destination node d produce longer tour durations compared to extensions of another label. The dominance algorithm (Step 10) applies the following rule:

Rule 2.3. *(Domination) Let $L = (k, i, S, T_i)$ and $L' = (k, i, S, T_i')$ be two labels with identical path length k, set S and last node i. Let $P = P(L)$ and $P' = P(L')$ be the respective paths. If $T_i \leq T_i'$ (component-by-component), any feasible extension of P' towards d is also a feasible extension of P with non-smaller tour duration. Hence, L' can be discarded.*

The validity of this dominance rule is a direct consequence of the fact that the REFs are non-decreasing (see Desaulniers *et al.*, 1998; Irnich and Desaulniers, 2005).

2.3.2 Backward Dynamic-Programming Formulation

We now define consistent backward resource extensions that allow the reversal of the DP approach so that a backward DP results. Moreover, the backward DP or any of its relaxations provides bounds for the forward DP.

Irnich (2008) presented a general framework applicable to classical REFs of the form $f_{ij}(T_i) = \max\{p_{ij}, T_i + t_{ij}\}$ and REFs of the form $f_{ij}(T_i, T_i') = (\max\{p_{ij}, T_i + t_{ij}, T_i' + u_{ij}\}, \max\{p_{ij}', T_i + t_{ij}', T_i' + u_{ij}'\})$ (the latter is called REFs with pairwise max-term). Herein, it is assumed that the extension along the arc $(i,j) \in A$ is feasible if $f_{ij}(T)$ and $f_{ij}(T, T')$ does not exceed q_{ij} and (q_{ij}, q_{ij}'), respectively. Then, the inverse REFs for REFs with pairwise max-term are $f_{ij}^{bw}(T_j) = \min\{q_{ij}, T_j - t_{ij}\}$ and $f_{ij}^{bw}(T_j, T_j') = (\min\{q_{ij}, T_j - t_{ij}, T' - t_{ij}'\}, \min\{q_{ij}', T_j - u_{ij}, T_j' - u_{ij}'\})$ (see Theorem 5 in (Irnich, 2008)).

For the MTDP, a *backward path* $(i, .., d)$ defines a *backward label* (k, i, S, T_i^{bw}). Herein, k is the length of the path, i the first visited node (i.e., the last node when propagating backward), S the set of all visited nodes, and a resource vector $T = (T_i^{time}, T_i^{dur}, T_i^{help})$. It is very important to mention that the semantics of the backward resources differs from the semantics of the forward resources: First, T_i^{time}, the latest feasible arrival time at node i. Second, T_i^{dur} is the difference between UB and the minimum tour duration of the path. Note that we assume that an upper bound for the minimal tour duration of a Hamiltonian path is known. The assumption is certainly not restrictive because $UB = b_d - a_o$ is always a valid upper bound. Moreover, the tour duration $T_i^{dur} - UB$ does not take the upper bound b_i of the time window at node i into account. Third, T_i^{help} is the difference of UB and the earliest possible arrival time at node d with tour duration T_i^{dur} that satisfies the time window constraints of every node in the path.

We define the initial state $(0, d, \{d\}, T_d)$ with $T_d = (T_d^{time}, T_d^{dur}, T_d^{help}) := (b_d, UB, UB - a_d)$. When we propagate a state (k, i, S, T) backward from node j to node i, i.e., in reverse direction along an arc $(i,j) \in A$, we add i to S and use the following REFs to update the resource vector T_j:

$$T_i^{time} = f_{ij}^{time,bw}(T_j) := \min\{T_j^{time} - t_{ij}, b_i\} \tag{2.3a}$$

$$T_i^{dur} = f_{ij}^{dur,bw}(T_j) := \min\{T_j^{dur} - t_{ij}, T_j^{help} - t_{ij} + b_j\} \tag{2.3b}$$

$$T_i^{help} = f_{ij}^{help,bw}(T_j) := \min\{T_j^{dur} - a_j, T_j^{help}\} \tag{2.3c}$$

The resource windows are defined as before by equations (2.1).

In general, any backward label T_i^{dur} at the start node i disregards the upper bound b_i of the time window at that node. However, the true resulting tour duration can be computed as $\max\{UB - T_i^{dur}, UB - T_i^{help} - b_i\}$.

Example 2.2. We consider the same example and path $P = (0, 1, 2, 3, 4)$ with time windows $[a_j, b_j]$ as in Example 2.1. As before the time windows and travel times $t_{j,j+1}$ are given in the first three columns of the following table. The forward resources are denoted by S and backward resources by T so that we can distinguish between the two. Note that the computed values are correct for any upper bound $UB \geq 12$ on the tour duration.

Node	Time window	Travel time	Backward Resources			True duration	Forward Resources		
j	$[a_j, b_j]$	$t_{j,j+1}$	T_j^{time}	T_j^{dur}	T_j^{help}	$\max\left\{UB - T_j^{dur}, UB - T_j^{help} - b_j\right\}$	S_j^{time}	S_j^{dur}	S_j^{help}
4	[14,18]	-	18	UB	$UB-14$	0	15	12	-3
3	[11,12]	4	12	$UB-4$	$UB-14$	4	11	8	-3
2	[5, 6]	3	6	$UB-7$	$UB-15$	9	5	3	-3
1	[2, 5]	2	4	$UB-11$	$UB-15$	11	2	1	-4
0	[0, 6]	1	3	$UB-12$	$UB-15$	12	0	0	-6

We discuss the backward labels now: First, the initial backward resource vector (T_j^{time}, T_j^{dur}, T_j^{help}) $= (18, UB, UB-14)$ results from the upper bound values defined in (2.1). In the path $Q_3 = (3, 4)$ the tour duration $UB - T_3^{dur} = 4$ and sum of travel time coincide, and $T_j^{help} = UB - 14$ means that the earliest arrival time at node $d = 4$ is 14 because the bound $a_3 = 11$ remains disregarded (see explanation of the semantics of the backward resources given above). The tour duration of the path $Q_2 = (2, 3, 4)$ is still 7, i.e., identical to the sum of the travel times, because also here the respective bound $b_2 = 6$ is not taken in account. The earliest possible arrival time at node d is $UB - T_3^{help} = 15$. In the path $Q_1 = (1, 2, 3, 4)$, two units of waiting time occur. They cannot be avoided because the earliest possible arrival time at node d is 15 and the latest possible departure time at node 1 is $b_2 - t_{23} = 4$. The path $Q_0 = (0, 1, 2, 3, 4)$ has a tour duration of $UB - T_0^{dur} = 12$, which consists of 10 units of travel time and two units of waiting time before node 3. The earliest possible arrival time at node $d = 4$ is $UB - T_0^{help} = 15$. The table also contains the values for the forward resource vector $S_j = (S_j^{time}, S_j^{dur}, S_j^{help})$ in the last three columns. They are computed using forward partial paths as shown in Example 2.1. At any intermediate node j of the path $(0, 1, 2, 3, 4)$, the forward label S_j for the forward path $(0, \ldots, j)$ and the backward label T_j for the backward path $(j, \ldots, 4)$ fulfill $S_j \leq T_j$. In this sense, the definition of forward and backward REFs are consistent.

Next, we present the fathoming rules for the backward DP. This rules are counterparts of Rules 1-3 for the forward DP. The proofs are straightforward. For the sake of completeness the entire backward dynamic labeling algorithm is given in the Appendix in Section 2.A.

Rule BW 2.1. *(Feasibility) A backward label (k, j, S, T_j) cannot lead to a feasible solution if there exists a node $i \in V \setminus S$ with $T_j^{time} < EAT(i, j)$. Hence, such a label can be discarded.*

Rule BW 2.2. *(Bounding) Let UB be an upper bound for the MTDP. A backward label (k, j, S, T_j) cannot lead to an improved solution if $UB - T_j^{help} - b_o \geq UB$. Hence, it can be discarded.*

Rule BW 2.3. *(Domination) Let $L = (k, j, S, T_j)$ and $L' = (k, j, S, T'_j)$ be two backward labels with identical path length k, set S and first node j. Let $P = P(L)$ and $P' = P(L')$ be the respective backward paths. If $T_i \geq T'_i$ (component-by-component), any feasible extension of P' towards o is also a feasible extension of P with non-smaller tour duration. Hence, L' can be discarded.*

2.3.3 Concatenation of Forward and Backward Paths

The concatenations of forward and backward labels allows the computation of Hamiltonian paths and their attributes. With the above definitions of REFs and labels, feasibility testing and the computation of the objective is straightforward. The corresponding statements are summarized in the following proposition.

Proposition 2.1. *Let a feasible forward path $P^{fw} = (o, \ldots, i)$ with forward label (k, i, S, T_i) and a feasible backward path $P^{bw} = (i, \ldots, d)$ with backward label (k', i, S', T'_i) be given. Then:*

(i) *The concatenation path $P = P^{fw} \oplus P^{bw}$ is a feasible Hamiltonian o-d path if and only if $k + k' = |V| - 1$, $S \cap S' = \{i\}$, and $T \leq T'$ holds.*
(ii) *The tour duration of $P = P^{fw} \oplus P^{bw}$ is given by $z = UB - \max\{T'^{dur}_i - T^{dur}_i, T'^{help}_i - T^{help}_i\}$.*

The formula for the tour duration results from the observations that the duration of the forward path is T^{dur}_i, the duration of the backward path is $UB - T'^{dur}_i$, the latest possible departure time at the start o is $-T^{help}_i$, and the earliest possible arrival time at destination d is $UB - T'^{help}_i$.

Note that if the tour duration is defined by the term $UB - (T'^{help}_i - T^{help}_i)$, there occurs waiting time on the path when going forward from node i to $i + 1$. The amount of waiting time (when going from node i to $i + 1$) is defined by the difference of the right and left term of the maximum. In the Example 2.2, the concatenation of the corresponding forward and backward labels is always feasible and the minimum tour duration is always 12. The time window bounds b_2 and a_3 and the travel time t_{23} imply that there must be waiting when going from node 2 to node 3. Hence, the tour duration at node 2 is defined by the right term of the maximum.

There exist several possible strategies to conduct bidirectional labeling, e.g., discussed by Righini and Salani (2006) and Li (2009). However, we will *not* apply bidirectional labeling to produce Hamiltonian paths, but we use Proposition 2.1 to derive bounding procedures.

2.4 Upper Bounds

For the asymmetric traveling salesman problem (ATSP), Balas (1999) proposed and analyzed a family of large-scale neighborhoods. Although, the number of neighbor solutions is exponential (in the length of the tour), the neighborhoods can be searched efficiently by means of *very large-scale neighborhood search* (VLSNS). In VLSNS, which is a variant of local search, a best neighbor solution is found by solving another optimization problem that can be solved in (pseudo-)polynomial time. We briefly summarize the VLSNS for the so-called Balas-Simonetti neighborhood of the ATSP, before we point out its use in the TSPTW context originally discussed in (Balas and Simonetti, 2001).

Given an ATSP Hamiltonian path $x = (x_1, \ldots, x_n)$ the neighborhood $\mathcal{N}^k_{BS}(x)$, for a given parameter $k \geq 2$, consists of all tours $x' = (x_{\gamma(1)}, \ldots, x_{\gamma(n)})$, where γ is a

permutation of $\{1, \ldots, n\}$ that fulfills the following conditions: For any two indices $i, j \in \{1, \ldots, n\}$ with $i + k \leq j$, the inequality $\gamma(i) \leq \gamma(j)$ holds. It means that if a node x_i precedes a node x_j by at least k positions, then x_i must also precede x_j in the neighbor solution. Moreover, the nodes x_1 and x_n are typically kept fixed at positions 1 and n, respectively.

A best neighbor solution $x' \in \mathcal{N}_{BS}^k(x)$ can be determined by solving a shortest-path problem in an auxiliary graph G_k^*. Figure 2.1 shows an example of G_k^* for $k = 3$ and a tour $x = (x_1, \ldots, x_7)$. The auxiliary graph G_k^* is well-structured and consists of n

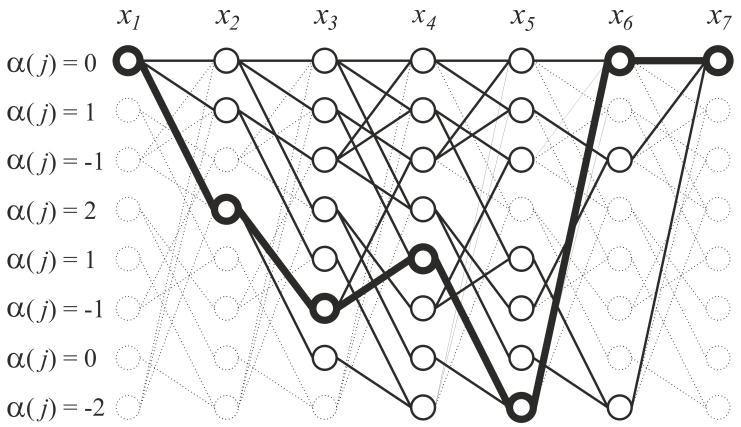

Figure 2.1: Auxiliary Graph G_k^* for $k = 3$ for the Balas-Simonetti Neighborhood $\mathcal{N}_{BS}^3(x)$ of $x = (x_1, \ldots, x_7)$

identical stages for a path x of length $n - 1$. The $(k + 1)2^{k-2}$ states at stage i are denoted by V_i, and $k(k + 1)2^{k-2}$ arcs connect the states of consecutive stages V_i and V_{i+1}. Stage 1 contains the start state o, and stage n the sink state d. Every o-d-path in G_k^* represents a neighbor solution x', and vice versa. This property results from the fact that each state s refers to a (restricted) permutation of the nodes around position i. In particular, for a given tour $x = (x_1, \ldots, x_n)$ the state s in stage i determines the permuted node $x'_i = x_{i+\alpha(s)}$ at position i in the neighbor tour x', where $\alpha(s)$ is an integer number associated with state s.

Since all induced subgraphs $G_k^*[V_i \cup V_{i+1}]$ are identical, only one such copy needs to be constructed beforehand, and only once. Herewith, the auxiliary graph G_k^* is represented implicitly. Every neighborhood search is then a shortest-path computation on this acyclic digraph G_k^* requiring $\mathcal{O}(n \cdot k^2 2^k)$ time and space. Nevertheless, the construction of the subgraphs $G_k^*[V_i \cup V_{i+1}]$ is non-trivial, i.e., the rules that determine the arc set and the values $\alpha(s)$ for states $s \in V_i$; details can be found in the papers (Balas and Simonetti, 2001; Simonetti and Balas, 1996).

The neighboring tour associated with the path depicted in bold in Figure 2.1 is $x' = (x_{1+0}, x_{2+2}, x_{3-1}, x_{4+1}, x_{5-2}, x_{6+0}, x_{7+0}) = (x_1, x_4, x_2, x_5, x_3, x_6, x_7)$. Those states s that refer to positions $i + \alpha(s) < 1$ or $i + \alpha(s) > n = 7$ are states that cannot be reached by any o-d-path; unreachable states are drawn with dotted lines in Figure 2.1. In order to ensure that any o-d-path in G_k^* has a cost identical to the cost of the implied neighbor

solution x', one has to label arc $(s, s') \in V_i \times V_{i+1}$ with cost $c_{x_{i+\alpha(s)}, x_{i+1+\alpha(s')}}$. For instance, the first bold arc in Figure 2.1 is labelled with cost $c_{x_{1+0}, x_{2+2}} = c_{x_1, x_4}$, the second has cost $c_{x_{2+2}, x_{3-1}} = c_{x_4, x_2}$ etc.

The auxiliary graph G_k^* can now be used to improve a given MTDP solution x by solving a shortest-path problem with resource constraints (SPPRC, Irnich and Desaulniers, 2005) on G_k^*. The initial label is $T_o = (a_o, 0, -b_0)$ associated with the initial state o of G_k^*. When propagating resources from a state s at stage i to another state s' at stage $i+1$, we use the REF $f_{i+\alpha(s),i+1+\alpha(s')}$ as defined by (2.2) and check the resource consumption at node $x_{i+1+\alpha(s')}$ using the resource bounds (2.1). Note that it is not necessary to keep track of the last node, the stage, and visited nodes (in the DP recorded by i, k and S) because the auxiliary network G_k^* ensures by construction that o-d paths are elementary whenever the input tour x was elementary. If an improving MTDP tour is found, x is replaced by the new tour and the process is iterated.

This is a local search procedure using the neighborhood $\mathcal{N}_{BS}^k(x)$. One can start the local search even with an elementary tour that is *not* resource feasible. If its neighborhood contains at least one resource feasible tour, the search can be continued.

2.5 Relaxations

We now describe DP relaxations for the MTDP, which can be used to compute lower bounds for the exact DP. We classify the relaxations into two groups: The first group relaxes elementary and the second relaxes resource feasibility. All presented relaxations can be used in the forward and backward DP in the same manner as in the exact DPs. Furthermore, a relaxed backward DP provides bounds (leading to another bounding procedure to be used in Step 8 of Algorithm 2.1) for the forward DP, and vice versa.

The MTDP differs significantly from the TSPTW with respect to relaxations. Since the TSPTW has only the two resources cost and time, resource-based relaxations with only one resource have not been considered (to the best of our knowledge). Such a relaxation would be necessarily one that has a cost objective because otherwise it cannot be used for bounding purposes. In consequence, the are no combinations of relaxations with respect to elementarity and resources for the TSPTW, in contrast to what is discussed next for the MTDP.

2.5.1 Relaxing Elementary

The *ng*-tour relaxations were first introduced by Baldacci *et al.* (2011) for the VRP, and the *ngL*-tour relaxations were presented for the TSPTW by Baldacci *et al.* (2012a). Both are families of relaxations (parameterized) and allow some non-elementary tours. For the MTDP, an *ng-tour* requires the existence of a feasible schedule. We adapt the *ng*-tour relaxation for computing a least-duration *ng*-tour from o to d of length $n+1$. Clearly, if the computed tour is elementary it constitutes an optimal solution to the MTDP.

A specific *ng*-tour relaxation requires the definition of sets $N_i \subseteq V$ with $i \in N_i$, one for each node $i \in V$. A *forward ng-path* $P = (o, ..., i)$ is a non-necessary elementary path starting at node o and ending at node i and all nodes are visited within their time window. P defines a *forward ng-label* (k, i, S_{ng}, T_i), where k is the path length and S_{ng} is a (generally proper) subset of the visited nodes. The vector T contains the same resources as in the exact formulation, and the same resource windows and REFs for the resources are used as in the exact DP. The key point of this relaxations is the update of the set S_{ng}, which differs from the set S in the exact formulation: Forward propagation of a label (i, k, S_{ng}, T_i) along an arc $(i, j) \in A$, i.e., towards node j, produces the new label $(j, k+1, S'_{ng}, T_j)$, where $T_j = f_{ij}(T_i)$ and $S'_{ng} = (S_{ng} \cup \{j\}) \cap N_j$. The interpretation is that the new label forgets the visited nodes that are not in the set N_j so that cycles become possible. For the sake of simplicity, we skip the index *ng* and write S instead of S_{ng} in the following.

The propagation criteria and fathoming rules need to be altered also: The label (k, i, S, T) can be propagated to a node j, if the arc (i, j) exists, $j \notin S$, $T_i^{time} \leq LDT(i, j)$, and $k > |\pi(j)|$. The first two criteria are identical to the exact DP. The third criterion replaces the feasibility Rule 2.1, which is generally invalid for *ng*-tours. The fourth stipulates that at least $|\pi(j)|$ nodes must precede the node j. Note that we cannot require $\pi(j) \subseteq S$ (as in the exact case) because in the *ng*-tour relaxation all nodes $\pi(j)$ may have been visited already even if $\pi(j) \not\subseteq S$.

Next, we consider the fathoming rules for eliminating labels: An *ng*-label dominates another *ng*-label if it consumes less resources and it has more options to be propagated. For the MTDP, it means:

Rule *ng* 2.1. *(ng-Domination)*
Let $L = (k, i, S, T)$ and $L' = (k, i, S', T')$ two ng-labels with identical path length k, $S \subseteq S'$ and the identical last node i. Let $P = P(L)$ and $P' = P(L')$ be the respective paths. If $T_i \leq T'_i$ (component-by-component), any feasible extension of P' towards d is also a feasible extension of P with non-smaller tour duration. Hence, L' can be discarded.

Note that Rule 2.2 for bounding remains applicable as in the exact DP. However, additional bounding possibilities arise when a weaker relaxation provides bounds for the relaxation under consideration. Related aspects are discussed in Section 2.5.4.

The *ngL-tour relaxation* is a restriction of the *ng*-tour relaxation. In addition to the requirements for *ng*-tours, it guarantees that a specific subset of nodes is visited once and only once in a given order. Such a sequence of nodes results from the preprocessing phase, e.g., by the determination of a chain of precedences $(v_0, v_1, v_2, \ldots, v_p)$ with maximum length p. Moreover, we also determine those nodes $V_i \subset V$, which can be visited between each two consecutive nodes v_ℓ and $v_{\ell+1}$ of the chain. Step 4 in Algorithm 2.1 then loops over all $j \in V_\ell, j \notin S$ if the partial path defining the label (k, i, S, T_i) has already visited the node v_ℓ, but not the node $v_{\ell+1}$. For a more detailed description, we refer to (Baldacci *et al.*, 2011).

The quality of the lower bound computed by the *ng*-tour and *ngL*-tour relaxations strongly depends on the choice of the sets N_i. Clearly, larger neighborhoods produce tighter lower bounds, but they also increase the computation time of the relaxed DP.

Therefore, we limit the number of neighbors of a node by a constant Δ. We use two methods to determine *promising* neighborhoods: The first method simply fills the neighborhood N_i of node i with the Δ-nearest nodes which are reachable from i and from which i is reachable. The second method is based on a dynamical augmentation of the ng or ngL neighborhoods, which was applied for the capacitated arc-routing problem by Bode and Irnich (2015) and for the delivery man problem by Roberti and Mingozzi (2014). We start with a small or empty neighborhood $(N_i)_{i \in V}$ and solve the corresponding relaxed DP. If a node i is visited more than once in the optimal ng-tour computed, we add this node to the neighborhood of all nodes, which occurs in the cycle(s) containing node i. We formalize this procedure in Algorithm 2.2.

Algorithm 2.2: Dynamic Augmentation of ng Neighborhoods

```
for i ∈ V do Set N_i := {i}
repeat
    Solve the relaxed DP with neighborhoods (N_i)_{i∈V}; let P be the resulting
    ng-tour
    Detect all cycles C_j for nodes j ∈ V visited more than once in P
    Set 𝒞 := {(j, C_j) : |N_ℓ| < Δ  for all ℓ ∈ C_j}
    Select a vertex disjoint subset of 𝒞' ⊆ 𝒞 of cycles
    for (j, C_j) ∈ 𝒞' and ℓ ∈ C_j do
        Set N_ℓ := N_ℓ ∪ {j}
until 𝒞 = ∅
```

Result: The neighborhoods $(N_i)_{i \in V}$

2.5.2 Relaxing Resource Feasibility

In this section, we present two new relaxations for the MTDP, the $2res$ and the $1res$ relaxation. The $2res$ *relaxation* relaxes the resource feasibility of a tour by omitting the resource T_i^{time}. A path $(o, .., i)$ defines a $2res$*-label* (k, i, S, T_i), where the vector T_i only contains the two interdependent resources T_i^{dur} and T_i^{help}. As before, k is the path length, i the last visited node, and S the set of all visited nodes. We can use the same propagation criteria as in the exact DP (Step 4 of Algorithm 2.1), but the fathoming rules must be altered. The feasibility Rule 2.1 is not applicable because we do not keep track of the resource T_i^{time}. Instead, we use the following weaker feasibility rule:

Rule $2res$ **2.1.** *($2res$-Feasibility)*
A label (k, i, S, T_i) cannot lead to a feasible solution, if there exists a node $j \in V \setminus S$ with $a_o + T_i^{dur} > LDT(i, j)$. Hence, such a label can be discarded.

This rule estimates the tour duration to node j by supposing that its starting time would be the earliest possible time a_o and no time windows would imply waiting. The bounding Rule 2.2 and the dominance Rule 2.3 are applicable as in the exact case.

The $1res$ *relaxation* relaxes the $2res$ relaxation. The resource T_i^{help} is fixed to its smallest possible value $-b_o$. A path $(o, .., i)$ defines a $1res$*-label* (k, i, S, T_i^{dur}), where k is the path length, i the last visited node, S the set of all visited nodes, and T_i^{dur} is a lower bound of the path duration. The label propagation considers only the single resource T_i^{dur}, so that (2.2b) reduces to $T_j^{dur} := f_{ij}^{dur}((k, i, S, T_i)) := \max\{T_i^{dur} + t_{ij}, a_j - b_o\}$. We can apply the same propagation criteria and fathoming rules as for the $2res$ relaxation.

2.5.3 Combined Relaxations

The $1res$ or $2res$ relaxations can be combined with the ng-tour or ngL-tour relaxations. We can use the propagation and dominance rules as in the ng-tour relaxation except for the feasibility criterion that is based on the resource T_i^{time}. Instead, we forbid to propagate a *combined label* (k, i, S, T_i) (with $T_i = T_i^{dur}$ or $T_i = (T_i^{dur}, T_i^{help})$) to a node j if $a_o + T_i^{dur} > LDT(i, j)$. Figure 2.2 shows the hierarchy of the relaxations, in which one relaxation stands below another if the latter is a proper relaxation of the first.

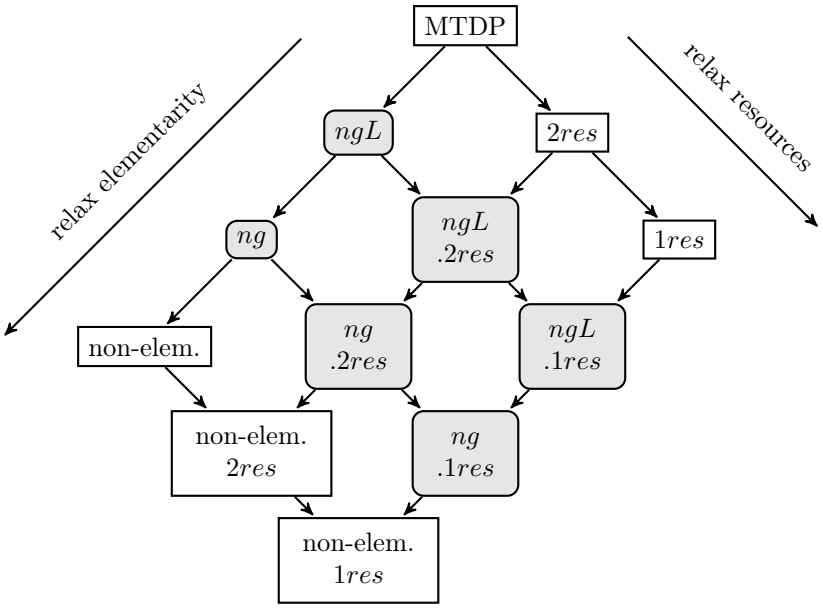

Figure 2.2: Hierarchy of the MTDP Relaxations. Shaded boxes are families of relaxations.

2.5.4 Relaxation-based Bounding

In this section, we adapt the *bounding procedure* introduced by Christofides *et al.* (1981) for the MCTP, and later applied by Baldacci *et al.* (2012a) for the TSPTW. As shown in Proposition 2.1, the concatenation of a forward and a backward path, given by a forward and a backward label, can easily be checked regarding feasibility. The duration of the resulting tour can be computed in constant time.

Instead of concatenating two exact labels, we can concatenate an exact and a label from a relaxation or even two relaxed labels to obtain a non-necessarily feasible TSPTW tour. Let a forward label (k, i, S, T_i) be given. The set of backward labels at stage $k' = |V| - k - 1$, either exact or from a relaxation, is denoted by $\mathcal{L}^{bw}_{k'}$. The following lower bound on the tour duration of the concatenation is a direct consequence of Proposition 2.1:

$$lb^{dur}(k, i, S, T_i) := \min_{\substack{(k', i, S', T'_i) \in \mathcal{L}^{bw}_{k'}: \\ k' = |V| - k - 1, S \cap S' = \{i\}, T_i \leq T'_i}} \max \left\{ T_i^{dur} - T_i'^{dur}, T_i^{help} - T_i'^{help} \right\} \tag{2.4}$$

Note that the part (i) of Proposition 2.1 provides the preconditions $k + k' = |V| - 1, S \cap S' = \{i\}$, and $T_i \leq T'_i$ for forming a feasible TSPTW tour, while part (ii) identifies the term $\max\{T_i^{dur} - T_i'^{dur}, T_i^{help} - T_i'^{help}\}$ providing the tour duration of the concatenation. The following modifications have to be made if the forward label (k, i, S, T_i) or the backward labels (k', i, S', T'_i) result from a relaxation:

- **ng, ngL:** S and/or S' have to be replaced by S_{ng} and/or S'_{ng}.
- **2res:** $T_i \leq T'_i$ has to be replaced by $(T_i^{dur}, T_i^{help}) \leq (T_i'^{dur}, T_i'^{help})$.
- **1res:** $T_i \leq T'_i$ has to be replaced by $T_i^{dur} \leq T_i'^{dur}$ and there is no second term in the maximum.

For combined relaxations (see Section 2.5.3), all of the associated modifications apply. Now we can define a bounding rule:

Rule 2.4. *(Bounding)*
Let UB be an upper bound for the MTDP and $lb^{dur}(k, i, S, T_i)$ be defined as in equation (2.4). Any label (k, i, S, T_i) with $lb^{dur}(k, i, S, T_i) \geq UB$ cannot lead to an improved solution and can be discarded.

The role of forward and backward labels can be swapped, i.e., a single backward label (k', i, S', T'_i) is given and all compatible forward labels from a relaxation provide the lower bound $lb^{dur}(k', i, S', T'_i)$. We leave the obvious formulation of the corresponding equation and bounding rule to the reader.

A final remark concerns Algorithm 2.2 for the dynamic neighborhood augmentation, i.e., for finding effective neighborhoods $(N_i)_{i \in V}$ for the ng- and ngL-tour relaxations. If we alternate between forward and backward DP, every time Step 2 of Algorithm 2.2 is executed, we can use the labels from the last iteration for bounding in the current iteration. Indeed, if the last iteration was a backward (forward) DP, its labels refer to a proper relaxation of the current forward (backward) DP. This trick drastically reduces the computation time for the iterated solution of the DPs, see Section 2.7.

2.6 Improving Bounds

Also the *penalty method* was first applied by Christofides *et al.* (1981) for solving the MCTP and was later used by Baldacci *et al.* (2012a) for the TSPTW. Its purpose is to

further improve lower bounds resulting from relaxations that relax elementarity. We will briefly point out specifics of the penalty method when applied to (combined) ng-tour or ngL-tour relaxation for the MTDP.

Let $V' = V \setminus \{o, d\}$ and $\mathcal{H}$ be the set of all tours generated by a given relaxation. By d_k we denote the minimum duration of the tour $k \in \mathcal{H}$ and by δ_{ik} the number of times an ng-tour $k \in \mathcal{H}$ visits node $i \in V'$.

The penalty method uses penalties λ_i associated with each node $i \in V'$ in order to modify the objective value of non-elementary tours. For ease of notation, we define $\lambda_o = \lambda_d := 0$ and $\Lambda := \sum_{i \in V} \lambda_i$. The objective of the MTDP is tour duration minimization so that we modify the REFs f_{ij}^{dur} and f_{ij}^{help} by subtracting the penalty λ_j. To be consistent, the resource windows defined by (2.1) need to be redefined as $T_o^{dur} \in [0, \infty)$, $T_i^{dur} \in (-\infty, \infty)$ for $i \in V'$, $T_d^{dur} \in (-\infty, UB - \Lambda]$ and $T_i^{help} \in (-\infty, \infty)$ for $i \in V \setminus \{o\}$. The resource bounds $T_o^{help} \in [-b_o, \infty)$ remain unchanged. The altered tour duration of any tour $k \in \mathcal{H}$ is then $d_k - \sum_{i \in V} \delta_{ik} \lambda_i$, which is a modification by Λ for all elementary tours. For a given penalty vector $\lambda \in \mathbb{R}^{|V|}$, the value

$$lb(\lambda) := \min_{k \in \mathcal{H}} \{c_k - \sum_{i \in V} \delta_{ik} \lambda_i + \Lambda\}$$

is a valid lower bound for the MTDP. The following Lagrangian dual problem can be solved to find a tight lower bound:

$$(LD) \qquad z_{LD} := \max\{lb(\lambda) : \lambda \in \mathbb{R}^{|V|}\}$$

We use subgradient optimization and column generation to solve problem (LD): The standard subgradient algorithm, presented as Algorithm 2.5 in Section 2.B of the Appendix is run for $\max_{iter}$ iterations.

The column-generation algorithm for the TSPTW was suggested by Baldacci *et al.* (2012a). The linear relaxation of the master program is the following problem:

$$\min \quad \sum_{k \in \mathcal{H}} d_k y_k \tag{2.5a}$$

$$\text{s.t.} \quad \sum_{k \in \mathcal{H}} \delta_{ik} y_k = 1 \quad \text{for all } i \in V' \tag{2.5b}$$

$$y_k \geq 0 \quad \text{for all } k \in \mathcal{H} \tag{2.5c}$$

Problem (2.5) provides an identical bound as (LD) and can be solved by column generation (see Desaulniers *et al.*, 2005; Lübbecke and Desrosiers, 2005) using a *restricted master program* (RMP), which is the restricted version of (2.5) containing only the tours found in the subgradient optimization and the tour producing the upper bound UB, see Section 2.4. Let $\lambda'_i, i \in V'$ be the dual prices of the constraints (2.5b) of the RMP. Moreover, we define $\lambda'_o := \lambda'_d :=0$. The column-generation algorithm alternates between the LP re-optimization of the RMP and the column-generation pricing problem that adds additional variables (=columns) to the RMP. The pricing problem asks for a tour

with negative reduced cost (=tour duration) $d_k - \sum_{i \in V} \delta_{ik} \lambda'_i$. This requires the solution of the same (relaxed) DP as in the subgradient optimization, but with λ replaced by λ', and Λ replaced by $\Lambda' := \sum_{i \in V} \lambda'_i$. The same bounding possibilities as discussed in Section 2.5.4 remain applicable. This includes the use of the lower bounds $lb(k, i, S, T_i)$ (see equation (2.4)), to which Λ' has to be added at the end. The only crucial point is that REFs and resource bounds need to be altered as described at the beginning of this section.

2.7 Computational Results

This section presents the computational results of the DP-based solution approaches for the MTDP. All computations were performed on a standard PC with an Intel(R) Core(TM) i7-2600 at 3.4 GHz processor with 16 GB of main memory. Algorithms were coded in C++ and compiled in release mode with MS-Visual Studio 2010. The callable library of CPLEX 12.5 was used to iteratively re-optimize the RMP in the column-generation algorithm of Section 2.7.4. We tested our algorithm on the instances of Potvin and Bengio (1996), Gendreau *et al.* (1998), Ohlmann and Thomas (2007), and Ascheuer (1995). The first three sets can be obtained from `http://myweb.uiowa.edu/bthoa/TSPTWBenchmarkDataSets.htm`. These three classes feature instances with x-y coordinates. Travel times are first computed as truncated integer Euclidean distances and then modified to satisfy the triangle inequality. The `Potvin+Bengio` benchmark set contains 30 instances with up to 46 nodes. The instances of the `Gendreau` benchmark consist of 120 instances divided in 24 subgroups of five instances each. The five instances within the same group have an identical number of nodes (between 21 and 101) and comparable time window widths (ranging from 100 to 200). Since the `Gendreau` instances have large variations in size, we divided them into two groups. `Gendreau large` includes 45 instances with more than 80 nodes, and `Gendreau small` the 75 smaller instances. The `Ohlmann+Thomas` benchmark extend the `Gendreau` benchmark regarding an increasing number of nodes (between 150 or 200) and time windows widths (from 120 to 160).

The `Ascheuer` benchmark set is available at `http://ftp.zib.de/pub/mp-testdata/tsp/atsptw/index.html` and consists of 50 asymmetric TSPTW instances with up to 233 nodes. Travel times are integer and satisfy the triangle inequality. As in Dash *et al.* (2012), we divide this class into 32 easy instances and 18 hard instances.

Before we apply our DP algorithms, we preprocess the instances according to Section 2.3. Note that we set $a_d = 0$ (earliest service at the destination) before preprocessing so that the tour duration can vary due to different arrival times at the destination. Furthermore, the `Ascheuer` instances are originally constrained by the origin time window $[a_o, b_o] = [0, 0]$, which we enlarge to $[0, 1000]$.

2.7.1 Computation of Upper Bounds

Since local search times become longer with increasing values of k for the Balas-Simonetti neighborhoods $\mathcal{N}_{BS}^k$, we iterate the local search in a *variable neighborhood descent* (VND) heuristic, where we use the values with $k = 7$, 9, 11, and 13. As an initial tour we use the known optimal tours to the TSPTW or the MCTP, which can be obtained at `http://iridia.ulb.ac.be/~manuel/tsptw-instances`. For these tours we compute an optimal MTDP schedule giving us an initial solution value. Table 2.1 shows the aggregated results over each group of instances. The column *improved* reports the number of times the initial solution value has been improved, the column $\#OPT$ shows the number of times an optimal solution was computed (if an optimum is known). The maximum and average GAP is reported in the next two columns. The last three columns give the minimum, maximum, and average solution time.

The VND improves the objective value of the input tour for nearly all instances except for the benchmark by `Ascheuer`, in which the VND improves approximately half of the known solutions. Furthermore, for instances with a known optimal solution value (computed by us, see Section 2.7.4), the VND returns an optimal solution in approximately 80 % of the cases. Note that the GAP is calculated only for those instances for which an optimal solution is known.

			GAP [%]		Time [s]		
Instances	improved	#OPT	max	ø	min	max	ø
`Potvin+Bengio`	27/30	17/28	7.4	1.0	0.1	7.7	1.1
`Ascheuer easy`	13/32	27/32	0.1	0.1	0.1	2.4	0.5
`Ascheuer hard`	11/18	17/18	1.1	0.1	0.9	8.3	3.2
`Gendreau small`	73/75	58/72	12.3	0.5	0.2	6.1	1.8
`Gendreau large`	44/45	30/32	3.5	0.1	1.3	16.5	6.6
`Ohlmann+Thomas`	25/25	1/1	0.0	0.0	10.8	105.5	35.9
Overall	193/225	151/183					

Table 2.1: Aggregated Results for Upper Bounds computed with the Balas-Simonetti Neighborhood

2.7.2 Comparison of the Relaxations

Now we compare the different relaxations introduced in Section 2.5 regarding their computation times and lower bounds. In pre-tests we found that the ngL-tour relaxations almost always outperform the corresponding ng-tour relaxations. Therefore, we limit our study to the $2res$, $1res$, ngL, $ngL.2res$, and $ngL.1res$ relaxations. Recall that the last three relaxations are parameterized with neighborhoods $(N_i)_{i \in V}$ and a sequence of nodes in precedence. For the comparison, we use a priori generated neighborhoods N_i with the $\Delta = 10$ closest neighbors to node i. The next section analyzes the impact of varying sizes and generation methods of the neighborhoods. The node sequence in precedence is arbitrarily chosen as a longest path in the precedence graph.

We set a hard time limit of 600 seconds for the computation of the lower bound as well as for the computation of the exact DP. The relaxation uses a forward DP, while the exact DP is computed backwards so that the relaxation provides valid bounds according to (2.4). Table 2.2 shows the aggregated results for all relaxations and the different benchmark sets. Column *#LB* displays the number of times a lower bound is computed, i.e., the relaxation is solved within the time limit. The next columns show the average and maximum *GAP* between the computed lower bound and the best known solution, followed by the average and maximum time. Column *#SOL* gives the number of times the exact DP was solved, and the following two columns report the average and maximum time for this step. The last two columns show the overall solution time for each instance on average and the maximum. In order to provide a fair comparison, the maximum and average gaps and times are taken only over those instances for which bounds were computed by all relaxations. Similarly, the times for solving the exact DP and the total time include only those instances solved by all five variants. We briefly

		#LB	GAP [%]		Time LB [s]		#SOL	Time exact [s]		Time all [s]	
Instances	Relaxation		ø	max	ø	max		ø	max	ø	max
	$2res$	18	0.9	5.9	77.2	556.0	18	0.1	1.3	77.2	556.0
`Potvin`	$1res$	19	3.6	9.7	23.6	289.7	19	0.1	1.4	23.7	289.7
`+Bengio`	ngL	21	1.7	9.4	13.5	186.2	21	0.5	4.5	14.0	188.7
($n = 30$)	$ngL.2res$	30	2.4	13.4	0.2	1.3	21	1.2	15.2	1.4	16.4
	$ngL.1res$	30	6.0	19.1	0.1	0.3	21	2.0	27.4	2.1	27.7
	$2res$	26	0.0	0.4	1.2	22.8	26	0.0	0.0	1.2	22.8
`Ascheuer`	$1res$	27	0.9	6.9	1.0	18.3	27	0.3	6.1	1.1	18.3
`easy`	ngL	32	3.1	30.3	0.2	2.4	28	1.9	27.7	2.1	27.8
($n = 32$)	$ngL.2res$	32	3.1	30.3	0.1	0.7	28	2.0	28.0	2.1	28.0
	$ngL.1res$	32	4.1	30.3	0.1	0.9	27	7.9	111.6	8.5	111.7
	$2res$	10	0.0	0.1	43.5	248.9	10	0.0	0.0	43.5	248.9
`Ascheuer`	$1res$	10	1.1	6.7	34.4	233.6	10	4.2	13.9	38.6	245.0
`hard`	ngL	18	1.0	5.9	7.5	41.0	14	3.1	10.4	10.7	41.4
($n = 18$)	$ngL.2res$	18	1.0	6.0	1.3	4.0	13	3.4	12.0	4.8	15.7
	$ngL.1res$	18	1.7	12.5	1.5	4.0	13	3.9	13.1	5.4	15.0
	$2res$	37	0.2	3.4	26.7	335.7	37	0.0	0.2	26.7	335.7
`Gendreau`	$1res$	42	2.5	8.6	12.5	239.7	42	0.1	2.4	12.6	242.1
`small`	ngL	67	7.9	21.8	13.0	95.4	52	2.8	95.1	15.7	138.3
($n = 75$)	$ngL.2res$	75	8.8	24.9	0.6	2.7	54	2.5	86.7	3.1	89.5
	$ngL.1res$	75	12.8	27.9	0.3	0.9	52	4.2	147.9	4.5	148.9
	$2res$	2	0.0	0.0	317.2	483.1	2	0.0	0.0	317.2	483.1
`Gendreau`	$1res$	2	0.0	0.0	243.6	359.7	2	21.8	35.4	265.4	395.1
`large`	ngL	19	0.0	0.0	6.6	12.5	8	66.7	88.3	73.3	89.0
($n = 45$)	$ngL.2res$	45	0.0	0.0	1.2	1.8	9	70.1	95.4	71.3	95.9
	$ngL.1res$	45	0.0	0.0	1.4	1.5	7	161.3	223.4	162.7	224.8
`Ohlmann`	$ngL.2res$	25	4.3	14.7	94.1	369.0	1	0.1	0.1	52.5	52.5
`+Thomas`	$ngL.1res$	25	4.3	14.7	12.5	23.7	1	0.1	0.1	13.7	13.7
($n = 25$)											
	$2res$	93	0.3	5.9	37.4	556.0	93	0.0	1.3	37.8	556.0
Overall	$1res$	100	2.1	9.7	18.7	359.7	100	1.1	35.4	20.0	395.1
($n = 225$)	ngL	157	4.4	30.3	8.8	186.2	123	3.5	95.1	12.4	188.7
	$ngL.2res$	225	4.9	30.3	0.5	4.0	125	3.7	95.4	4.2	95.9
	$ngL.1res$	225	7.6	30.3	0.4	4.0	120	8.2	223.4	8.7	224.8

Table 2.2: Aggregated Results for Different Relaxations

summarize the results: The $2res$ and $1res$ relaxations need much computation time and, therefore, they solve the fewest relaxations. If they are able to compute lower bounds,

however, the exact DP is always solved. In comparison, these two relaxations are too slow and therefore not beneficial.

The $ngL.2res$ and $ngL.1res$ relaxations are able to solve the relaxations on all problem instances and need comparably small time to terminate. When using the ngL-tour relaxation, its time is on average approximately 20 times higher compared to the $ngL.2res$ and $ngL.1res$ relaxations. Moreover, none of the three relaxations $2res$, $1res$, and ngL is able to compute a lower bound for the `Ohlmann+Thomas` instances. Consequently, rows for these relaxations are omitted in Table 2.2.

Comparing ngL with $ngL.2res$, gaps are slightly in favor of the ngL-tour relaxation. However, the exact DP in combination with the $ngL.2res$ relaxation is able to solve the largest number of instances to optimality, 126 out of 225, which is three more than with the ngL-tour relaxation. Also the overall computation times (relaxation plus exact DP) are in favor of the $ngL.2res$ relaxation. Therefore, the $ngL.2res$ relaxation seems to be the best compromise.

2.7.3 Comparison of ngL Neighborhood Sizes

Next, we analyze the impact of different neighborhoods $(N_i)_{i \in V}$ for the $ngL.2res$ relaxations. Recall from Section 2.5.1 that the neighborhoods can either be generated a priori (static version) or be generated using Algorithm 2.2 (dynamic version). We compare the different maximal sizes of the neighborhoods given by $\Delta = 8, 10, 12$, and 14. Pre-tests have shown that smaller or larger sizes are not beneficial (in a stand-alone algorithm not using a penalty method) because either the lower bounds are too weak or the computation times are too large. As in the previous analysis, we set a hard time limit of 600 seconds for the computation of the relaxation as well as for the exact DP.

Table 2.3 shows the aggregated results for the different parametrization. Herein, S denotes the static generation of the neighborhood and D the dynamic augmentation. The first column $\#LB$ denotes the number of times the respective relaxation was solved, followed by the number $\#SOL$ of times the exact DP was solved. The next columns report the average and maximal GAP, the average and maximum time to compute lower bounds and the average and maximum time to solve the exact DP. As before, to ensure a fair comparison, the average and maximum gaps refer to those instances that were solved by all methods. Similarly, the reported times for the exact DP refer to the instances solved to optimality by all variants.

Clearly, a larger Δ requires more computation time for the relaxation leading to generally smaller gaps and, in turn, smaller times for the exact DP. The static neighborhood generation needs less time than the dynamic version because in the latter case more relaxed DPs have to be solved. On the other hand, the dynamic neighborhood augmentation provides smaller gaps, more solutions and better solution times for the exact DP. For a few instances, the static approach solves the exact DP faster than the dynamic one. In these cases, the time windows impose a minimum for the lower bound of the MTDP. Since the dynamic neighborhood augmentation does not necessarily fill all neighborhoods $(N_i)_{i \in V}$ to the maximum size Δ opposed to the static augmentation, the latter may provide better lower bounds.

$ngL.2res$ with	#LB S	#LB D	#SOL S	#SOL D	GAP [%] ø S	GAP [%] ø D	GAP [%] max S	GAP [%] max D	Time LB [s] ø S	Time LB [s] ø D	Time LB [s] max S	Time LB [s] max D	Time Exact [s] ø S	Time Exact [s] ø D	Time Exact [s] max S	Time Exact [s] max D
`Potvin+Bengio` instances (n=30)																
$\Delta = 8$	30	30	22	22	6.1	5.2	21.9	20.5	0.7	1.9	9.4	10.5	5.7	5.3	96.8	94.5
$\Delta = 10$	30	30	22	22	5.3	4.3	20.0	17.2	1.8	5.9	17.7	42.9	4.7	4.4	81.3	80.9
$\Delta = 12$	30	30	22	22	4.9	3.7	19.6	16.0	4.7	20.9	39.2	127.1	3.9	3.6	66.6	65.3
$\Delta = 14$	30	28	22	22	4.7	3.3	19.6	15.5	26.6	69.4	482.7	496.1	3.1	2.7	50.8	46.8
easy `Ascheuer` instances (n=32)																
$\Delta = 8$	32	32	27	27	3.1	1.6	34.2	15.0	0.1	1.1	0.1	10.6	2.6	0.4	35.4	3.9
$\Delta = 10$	32	32	27	27	2.8	2.5	30.3	25.9	0.1	1.1	0.4	1.7	1.9	0.2	28.2	2.0
$\Delta = 12$	32	32	27	28	2.6	1.9	29.5	21.8	0.2	1.1	1.6	5.0	0.5	0.2	6.3	1.8
$\Delta = 14$	32	32	27	28	1.6	1.1	15.0	9.3	1.1	2.9	10.6	18.1	0.4	0.1	3.9	1.8
hard `Ascheuer` instances (n=18)																
$\Delta = 8$	18	18	13	14	1.4	1.2	7.0	4.9	0.3	2.5	0.7	9.3	2.8	1.6	12.8	8.4
$\Delta = 10$	18	18	13	14	1.3	1.1	6.0	4.9	0.8	7.2	2.0	25.2	2.8	1.5	12.8	8.3
$\Delta = 12$	18	18	13	14	1.2	1.0	5.6	4.9	2.7	21.4	8.3	92.4	2.8	1.5	12.7	8.2
$\Delta = 14$	18	18	13	14	1.1	1.0	4.9	4.9	9.7	91.5	33.0	351.8	2.7	1.4	12.7	8.1
`Gendreau small` instances (n=75)																
$\Delta = 8$	75	75	51	54	10.5	9.2	29.4	27.7	0.7	3.0	4.5	16.6	21.3	13.9	149.8	113.3
$\Delta = 10$	75	75	53	54	9.7	8.1	28.3	27.1	1.6	9.5	8.3	78.0	14.3	12.6	112.4	106.7
$\Delta = 12$	75	75	54	55	8.8	7.2	28.3	27.1	4.0	34.0	24.2	273.5	13.1	10.8	102.2	90.2
$\Delta = 14$	75	74	55	55	8.2	6.5	27.1	25.9	11.9	96.6	88.9	598.0	12.7	10.4	101.9	89.4
`Gendreau large` instances (n=45)																
$\Delta = 8$	45	45	8	9	4.6	4.6	21.8	21.8	2.1	11.6	8.8	42.8	14.4	2.8	58.6	22.7
$\Delta = 10$	45	45	9	9	4.6	4.6	21.8	21.8	5.0	35.2	16.7	136.8	13.0	2.8	54.8	22.6
$\Delta = 12$	45	43	10	10	4.6	4.6	21.8	21.8	14.0	100.5	52.7	327.0	12.5	2.7	54.5	21.8
$\Delta = 14$	45	37	10	10	4.6	4.6	21.8	21.8	38.9	241.8	148.4	596.3	12.3	2.7	54.3	21.3
`Ohlmann+Thomas` instances (n=25)																
$\Delta = 8$	25	23	1	1	0.3	0.3	0.5	0.5	17.4	19.4	21.1	34.6	0.1	0.1	0.1	0.1
$\Delta = 10$	25	16	1	1	0.3	0.3	0.5	0.5	40.7	35.7	52.5	67.1	0.1	0.1	0.1	0.1
$\Delta = 12$	24	8	1	1	0.3	0.3	0.5	0.5	106.5	237.4	125.4	470.1	0.1	0.1	0.1	0.1
$\Delta = 14$	16	2	1	1	0.3	0.3	0.5	0.5	336.1	237.4	345.0	470.1	0.1	0.1	0.1	0.1
`Overall` (n=225)																
$\Delta = 8$	200	200	122	127	4.8	4.1	0.6	27.7	0.6	2.7	9.4	42.8	5.9	3.5	149.8	113.3
$\Delta = 10$	200	200	125	128	4.3	3.5	1.4	27.1	1.4	8.1	17.7	136.8	4.5	3.1	112.4	106.7
$\Delta = 12$	200	198	127	130	4.0	3.0	3.8	27.1	3.8	25.3	52.7	326.9	3.8	2.7	102.2	90.2
$\Delta = 14$	200	189	128	130	3.6	2.7	14.7	25.9	14.7	78.2	482.8	598.0	3.4	2.3	101.9	89.4

Table 2.3: Aggregated Results comparing Different ngL Neighborhoods

2.7.4 Results of the Column-Generation Algorithm and Exact Dynamic Program

In this section, we report results of the column-generation algorithm (see Section 2.6) and the successive application of the exact DP. Recall that the column-generation method provides optimal penalties $(\lambda_i^*)_{i \in V}$. Both the lower bound LB and the solution help solving the exact DP faster: On the one hand, the solution of each pricing problem provides a lower bound LB for the MTDP, and this bound may suffice to close the gap to the upper bound computed with the VND (see Section 2.4). On the other hand, the solution itself, i.e., the labels produced by the $ngL.2res$ relaxation can be exploited for bounding using the bounds (2.4). We describe the overall approach in Algorithm 2.3.

The results of the last two subsections suggest the $ngL.2res$ relaxation to be used in the subgradient and the column-generation algorithms as well as the dynamic augmentation of the neighborhoods (Steps 4, 5, and 6). Pre-tests have shown that with small-sized neighborhoods N_i a significantly higher number of column-generation iterations is possi-

Algorithm 2.3: Overall Algorithm

1 CALL preprocessing // → $(A, \pi(i), \sigma(j), EAT(i,j), LDT(i,j), [a_i, b_i])$
2 CALL VND with Balas-Simonetti neighborhoods // → (UB, P^*)
3 COMPUTE static $ngL.2res$ neighborhoods $(N_i)_{i \in V}$ of size $\Delta_1 = 3$ // → (N_i)
4 CALL subgradient method with $\max_{iter} = 10$ // → $(LB, \lambda_i^*, ngL$ `tours`$)$
5 CALL column-generation algorithm // → (LB, λ_i^*)
6 CALL dynamic augmentation of neighborhoods with penalties $(\lambda_i^*)_{i \in V}$ and $\Delta_2 = 14$ // → (LB, N_i)
7 **while** $LB < UB$ **do**
8 SET $B = \min\{\lceil 1.01 \cdot LB \rceil, UB\}$
9 CALL exact DP with upper bound B; bounding with labels of Step 6 // → $(UB, P^*$ `or failed`$)$
10 **if** *DP failed* **then** SET $LB = B + 1$

Result: optimal MTDP tour P^* with minimum tour duration UB

ble. Herewith, lower bounds generally improve more due to good penalties compared to larger-sized neighborhoods. We have obtained the best results with a neighborhood size of $\Delta_1 = 3$ (Step 3). To improve the bounds of this relaxed DP, we augment the neighborhood dynamically (Algorithm 2.2 called in Step 6) after good penalties are found up to a neighborhood size of $\Delta_2 = 14$.

Furthermore, preliminary tests revealed that the running time of the exact DP is strongly impacted by the quality of the upper bound UB: A relatively weak upper bound causes that bounding procedures almost always fail so that the DP algorithm will not terminate (or terminate with an 'out of memory' message) due to an enormous set of labels that has to be generated and stored. Therefore, it is computationally advantageous to try tentative upper bounds B whenever the gap $UB - LB$ is relatively large. The loop (Steps 8–10), iteratively tries to increase the tentative upper bounds B, where the gap between B and LB never exceeds 1 % (plus 1). When the exact DP is called with a tight upper bound, more labels can be bounded and the computation time is reduced. If a tentative bound is too small, the DP will fail in the sense that no $ngL.2res$ tour is computed. In this case, however, one knows that the tentative upper bound is in fact a lower bound (Step 10). We set a hard time limit of one hour for the computation of the lower bounds by the column-generation algorithm as well as for the dynamic neighborhood augmentation and the computation of the exact DP.

For the ease of presentation, Algorithm 2.3 does not detail all possible termination points. Indeed, if any of the relaxed DPs terminates with a solution that is an elementary tour with a feasible schedule, this tour constitutes an optimal solution to the MTDP. Therefore, we perform this kind of check in Steps 4, 5, and 6. Moreover, if the rounded up value of any lower bound equals UB (computed by the subgradient or column-generation algorithm or Algorithm 2.2), the tour that has produced UB is optimal. Hence, Algorithm 2.3 is stopped whenever this happens for a valid upper bound UB (not for tentative upper bounds B).

Tables 2.4–2.6 show the results of Algorithm 2.3 aggregated over each instance group. Results for the `Ohlmann+Thomas` instances are left out, since Algorithm 2.3 did not solve additional instances compared to Section 2.7.3. In each table, n denotes the number of remaining instances to be solved. In turn, $\#SOL$ denotes the number of instances solved to optimality. The average and maximum gap as well as the average and maximum solution time are reported for each of the Steps 5, 6, and 9 of Algorithm 2.3.

Table 2.4 shows the results of Algorithm 2.3 up to Step 5. The column-generation penalty method is able to solve 108 of the 200 instances. Results for the remaining 92 instances are then presented in Table 2.5, where additionally the dynamic neighborhood augmentation procedure (Algorithm 2.2) is applied making use of the best penalties $(\lambda_i^*)_{i \in V}$ computed before. Another 25 instances are now solved so that for the remaining 67 instances the exact DP algorithm is invoked (Steps 8–10). The results of these final computations are summarized in Table 2.6. Another 46 instances are solved by the exact DP. In summary, the overall algorithm is able to solve 179 out of 200 instances.

			GAP [%]		Time [s]	
Instances	n	#SOL	ø	max	ø	max
`Potvin+Bengio`	30	8	2.9	15.1	120.4	1943.3
`Ascheuer easy`	32	20	0.1	0.4	3.6	70.6
`Ascheuer hard`	18	16	0.1	0.4	137.4	625.5
`Gendreau small`	75	41	0.8	7.1	283.4	3308.1
`Gendreau large`	45	23	0.5	4.6	1921.0	3600.0
Overall	200	108	0.9	15.1	582.5	3600.0

Table 2.4: Aggregated Results of the Overall Algorithm for all Instances

We now highlight some of the results visible in the tables: For the column-generation algorithm, the average GAP over all instances is smaller than one percent and the average solution time is smaller than ten minutes. The neighborhood augmentation procedure further reduces the average GAP of the remaining instances by 0.4 percent. Notably, computation times of this step are generally smaller than those for the column-generation method. Recall that alternating between forward and backward DP using bounds from the preceding iteration is possible for the neighborhood augmentation, but not for the column-generation algorithm due to the iterative modification of penalties in this case.

Moreover, the computation times of the neighborhood augmentation algorithm strongly depend on the size of the instance: For the large instances from the `Gendreau large` group (comprising 81 and 101 nodes) much more computation time is needed than for the other groups.

Finally, detailed results for every instance showing bounds, gaps, and computation times of the different solution procedures included in Algorithm 2.3 are listed in Section 2.C of the Appendix. An interesting detail in these results is that for only three instances the solution time of the exact DP exceeds four minutes. We can conclude that either an instance can be solved fast by the exact DP, or the solution time may easily exceed one hour.

		Column Generation				Dyn. *ng* neighb. augment.				
		GAP [%]		Time [s]			GAP [%]		Time [s]	
Instances	n	ø	max	ø	max	#SOL	ø	max	ø	max
`Potvin+Bengio`	22	3.9	15.1	136.8	1943.3	4	3.5	15.1	17.9	100.7
`Ascheuer easy`	12	0.1	0.4	1.2	3.5	2	0.1	0.4	0.2	0.7
`Ascheuer hard`	2	0.2	0.4	313.8	598.6	1	0.1	0.4	68.9	135.8
`Gendreau small`	34	1.6	7.1	310.3	3308.1	16	0.9	7.1	60.5	573.9
`Gendreau large`	22	0.9	4.6	2553.3	3600.0	2	0.8	4.6	872.8	2934.9
Overall	92	1.8	15.1	764.9	3600.0	25	1.4	15.1	236.9	2934.9

Table 2.5: Aggregated Results of the Overall Algorithm for Instances not solved by Step 5

		Column Generation				Dyn. *ng* neighb. augment.				Exact DP		
		GAP [%]		Time [s]		GAP [%]		Time [s]			Time [s]	
Instances	n	ø	max	ø	max	ø	max	ø	max	#SOL	ø	max
`Potvin+Bengio`	18	4.7	15.1	166.0	1943.3	4.3	14.0	21.8	100.7	15	110.1	1283.6
`Ascheuer easy`	10	0.1	0.3	1.2	3.5	0.1	0.4	0.2	0.7	10	0.1	0.1
`Ascheuer hard`	1	0.4	0.4	29.0	29.0	0.1	0.1	1.9	1.9	1	0.3	0.3
`Gendreau small`	18	2.0	7.1	512.9	3308.1	1.7	5.5	111.1	573.9	15	221.8	3108.7
`Gendreau large`	20	1.0	4.6	2448.4	3600.0	0.8	4.3	904.1	2934.9	5	92.0	458.2
Overall	67	2.1	15.1	913.9	3600.0	1.9	14.0	305.6	2934.9	46	118.3	3108.7

Table 2.6: Aggregated Results of the Overall Algorithm for Instances not solved by Steps 5 and 6

During the time we designed the overall algorithm, we tested various other setups and parameters. We used different initial tours in the Balas-Simonetti neighborhood, for instance an ordering of the nodes by increasing center of their time windows. The resulting upper bounds differ from those in Section 2.7.1, but they are in general neither better nor worse. Furthermore, we also did some longer computational tests. As a result, we were able to compute 13 more optimal solutions to the MTDP instances. The tables in Section 2.C of the Appendix indicate these optimal solutions with the symbol †.

2.8 Conclusions

In this paper, we have addressed the *minimum tour duration problem* (MTDP), which is a variant of the TSPTW with the objective of minimizing the time between the departure at the start and the arrival at the destination. The MTDP is a fundamental problem when routing vehicles whose movements are constrained by time windows.

We have presented algorithmic components for a DP-based approach tailored to the MTDP such as relaxed and exact forward and backward DPs, a VND, a dynamic *ng* neighborhood augmentation procedure, and a penalty method based on subgradient and column-generation algorithms. There exists a plethora of possible algorithmic designs to combine and parameterize these components. Our findings are the following:

First, for bounding purposes, a combined $ngL.2res$ relaxation provides an excellent tradeoff between strength of the resulting bounds and the computational effort. While ngL-based relaxations gradually allow some non-elementary tours, the $2res$ relaxation disregards the time window constraints, but keeps the computation of the tour duration exact. No pure relaxation exclusively relaxing elementarity constraints or resource constraints was found competitive with $ngL.2res$.

Second, a penalty algorithm which penalizes those routes that are non-elementary (and therefore also not Hamiltonian) is often more effective than enlarging the ng neighborhoods which specify the actual ngL-tour relaxation. For this reason, we decided for our approach that the column-generation method precedes the neighborhood augmentation procedure.

Third, with optimized penalties and carefully dynamically augmented ng neighborhoods, excellent lower bounds on the MTDP can be achieved. Our computational analysis shows that these lower bounds often suffice to either close the gap or to make the exact DP solution relatively easy.

Fourth, upper bounds produced with the Balas-Simonetti neighborhood-based VND are generally tight (below 1 % on average). However, since the exact DP is very sensible regarding the quality of the presented upper bound, it seems computationally advantageous to provide very tight tentative upper bounds for the exact DP. In case that the bound was wrongly chosen, i.e., too small, a repeated solution of exact DPs with increased tight upper bounds is most of the time faster than solving a single exact DP with a low-quality upper bound.

Summarizing the presented computational results, the proposed DP-based approach can solve above 90 % of the instances with up to 125 nodes from the standard benchmark sets to proven optimality. Of course, time window constraints certainly have a significant impact on the practical hardness of an instance. Therefore, we cannot make this observation a general statement even if some benchmark instances have relative wide time windows. One must keep in mind that the TSPTW is already a (practically) very hard combinatorial problem. For example, the `Ascheuer` benchmark set contained several open instances for more than a decade before it was completely solved by the work of Baldacci *et al.* (2012a). Due to the more involved REFs and resource constraints, the MTDP is certainly an even harder combinatorial problem.

We see one main contribution of the paper at hand in the consistent way that resources and REFs are defined for both forward and backward DP including relaxations. This consistency (see Section 2.3) is the theoretic background for the later algorithm design. As a result, the overall algorithm solves 189 out of 200 instances from the three benchmark sets `Potvin+Bengio`, `Ascheuer`, and `Gendreau` to optimality.

References

Ascheuer, N. (1995). *Hamiltonian Path Problems in the On-Line Optimization of Flexible Manufacturing Systems*. Ph.D. thesis, Technische Universität Berlin.

Ascheuer, N., Fischetti, M., and Grötschel, M. (2001). Solving the asymmetric traveling salesmann problem with time windows by branch-and-cut. *Mathematical Programming, Series A*, **90**(3), 475–506.

Baker, E. (1983). An exact algorithm for the time-constrained traveling salesman problem. *Operations Research*, **31**, 938–945.

Balas, E. (1999). New classes of efficiently solvable generalized traveling salesman problems. *Annals of Operations Research*, **86**, 529–558.

Balas, E. and Simonetti, N. (2001). Linear time dynamic-programming algorithms for new classes of restricted TSPs: A computational study. *INFORMS Journal on Computing*, **13**(1), 56–75.

Baldacci, R., Mingozzi, A., and Roberti, R. (2011). New route relaxation and pricing strategies for the vehicle routing problem. *Operations Research*, **59**(5), 1269–1283.

Baldacci, R., Mingozzi, A., and Roberti, R. (2012). New state-space relaxations for solving the traveling salesman problem with time windows. *INFORMS Journal on Computing*, **24**(3), 356–371.

Bode, C. and Irnich, S. (2015). In-depth analysis of pricing problem relaxations for the capacitated arc-routing problem. *Transportation Science*, **49**(2), 369–383.

Christofides, N., Mingozzi, A., and Toth, P. (1981). State-space relaxation procedures for the computation of bounds to routing problems. *Networks*, **11**, 145–164.

Dash, S., Günlük, O., Lodi, A., and Tramontani, A. (2012). A time bucket formulation for the traveling salesman problem with time windows. *INFORMS Journal on Computing*, **24**(1), 132–147.

Desaulniers, G., Desrosiers, J., Ioachim, I., Solomon, M., Soumis, F., and Villeneuve, D. (1998). A unified framework for deterministic time constrained vehicle routing and crew scheduling problems. In T. Crainic and G. Laporte, editors, *Fleet Management and Logistics*, chapter 3, pages 57–93. Kluwer Academic Publisher, Boston, Dordrecht, London.

Desaulniers, G., Desrosiers, J., and Solomon, M., editors (2005). *Column Generation*. Springer, New York, NY.

Desrosiers, J., Dumas, Y., Solomon, M., and Soumis, F. (1995). Time constrained routing and scheduling. In M. Ball, T. Magnanti, C. Monma, and G. Nemhauser, editors, *Network Routing*, volume 8 of *Handbooks in Operations Research and Management Science*, chapter 2, pages 35–139. Elsevier, Amsterdam.

Dumas, Y., Desrosiers, J., Gelinas, E., and Solomon, M. (1995). An optimal algorithm for the traveling salesman problem with time windows. *Operations Research*, **43**(2), 367–371.

Favaretto, D., Moretti, E., and Pellegrini, P. (2006). An ant colony system approach for variants of the traveling salesman problem with time windows. *Journal of information and optimization sciences*, **27**(1), 35.

Gendreau, M., Hertz, A., Laporte, G., and Stan, M. (1998). A generalized insertion heuristics for the traveling salesman problem with time windows. *Operations Research*, **43**(3), 330–335.

Goel, A. (2009). Vehicle scheduling and routing with drivers' working hours. *Transportation Science*, **43**(1), 17–26.

Goel, A. and Vidal, T. (2014). Hours of service regulations in road freight transport: An optimization-based international assessment. *Transportation Science*, **48**, 391–412.

Irnich, S. (2008). Resource extension functions: Properties, inversion, and generalization to segments. *OR Spectrum*, **30**(1), 113–148.

Irnich, S. and Desaulniers, G. (2005). Shortest path problems with resource constraints. In Desaulniers *et al.* (2005), chapter 2, pages 33–65.

Kara, I., Koc, O. N., Altıparmak, F., and Dengiz, B. (2013). New integer linear programming formulation for the traveling salesman problem with time windows: minimizing tour duration with waiting times. *Optimization*, **62**(10), 1309–1319.

Kindervater, G. and Savelsbergh, M. (1997). Vehicle routing: Handling edge exchanges. In E. Aarts and J. Lenstra, editors, *Local Search in Combinatorial Optimization*, chapter 10, pages 337–360. Wiley, Chichester.

Langevin, A., Desrochers, M., Desrosiers, J., Gélinas, S., and Soumis, F. (1993). A two-commodity flow formulation for the traveling salesman and the makespan problems with time windows. *Networks*, **23**, 631–640.

Li, J.-Q. (2009). A computational study of bi-directional dynamic programming for the traveling salesman problem with time windows. Technical report, Working paper, University of California, Berkeley, Berkeley.

Lübbecke, M. and Desrosiers, J. (2005). Selected topics in column generation. *Operations Research*, **53**(6), 1007–1023.

Mingozzi, A., Bianco, L., and Ricciardelli, S. (1997). Dynamic programming strategies for the traveling salesman problem with time window and precedence constraints. *Operations Research*, **45**(3), 365–377.

Ohlmann, J. and Thomas, B. (2007). A Compressed-Annealing heuristic for the traveling salesman problem with time windows. *INFORMS Journal on Computing*, **19**(1), 80–90.

Potvin, J.-Y. and Bengio, S. (1996). The vehicle routing problem with time windows Part II: Genetic search. *INFORMS Journal on Computing*, **8**(2), 165–172.

Prescott-Gagnon, E., Desaulniers, G., Drexl, M., and Rousseau, L.-M. (2010). European driver rules in vehicle routing with time windows. *Transportation Science*, **44**(4), 455–473.

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

Roberti, R. and Mingozzi, A. (2014). Dynamic ng-path relaxation for the delivery man problem. *Transportation Science*, **48**(3), 413–424.

Savelsbergh, M. (1992). The vehicle routing problem with time windows: Minimizing route duration. *Operations Research Society of America*, **4**(2), 146–154.

Simonetti, N. and Balas, E. (1996). Implementation of a linear time algorithm for certain generalized traveling salesman problems. In W. Cunningham, S. McCormick, and M. Queyranne, editors, *Integer Programming and Combinatorial Optimization*, volume 1084 of *Lecture Notes in Computer Science*, pages 316–329. Springer Berlin Heidelberg.

Vidal, T., Crainic, T. G., Gendreau, M., and Prins, C. (2011). A unifying view on timing problems and algorithms. Technical Report 2011-43, CIRRELT, Montréal, Canada.

Vidal, T., Crainic, T. G., Gendreau, M., and Prins, C. (2013). A hybrid genetic algorithm with adaptive diversity management for a large class of vehicle routing problems with time-windows. *Computers & Operations Research*, **40**(1), 475–489.

Appendix

2.A Backward Dynamic Programming Algorithm

Algorithm 2.4 is the backward DP labeling algorithm. Herein, the set $\sigma(i)$ is the set of all nodes $j \in V$ that must succeed i.

Algorithm 2.4: Backward Dynamic Programming Labeling Algorithm

SET $\mathcal{L}_0^{bw} := \{(0, d, \{d\}, (b_d, UB, UB - a_d))\}$
for $k = 0, 1, \ldots, |V| - 2$ **do**
  for $(k, j, S, T_j) \in \mathcal{L}_k^{bw}$ **do**
    for $(i, j) \in A : i \notin S, \sigma(i) \subseteq S$ **do**
      SET $T_i := f_{ij}^{bw}(T_j)$
      if *FeasibilityCheck(T_i)* **then**
        SET $L_i := (k + 1, i, S \cup \{i\}, T_i)$
        if *BoundingCheck(L_i)* **then**
          ADD L_i to $\mathcal{L}_{k+1}^{bw}$
  CALL Dominance algorithm for $\mathcal{L}_{k+1}^{bw}$
FIND a label $L_o^* = (|V| - 1, o, V, T_o) \in \mathcal{L}_{|V|-1}^{bw}$ with T_o^{dur} maximal
Result: The path P^* represented by label L_o^*

2.B Subgradient Optimization Algorithm

Algorithm 2.5 is the (standard) subgradient algorithm for the resolution of the Lagrangian-dual problem (LD) (see Section 2.6).

Algorithm 2.5: Standard Subgradient Optimization Algorithm

SET $t := 0$, $LB := 0$, $\lambda_l^* = \lambda_l^0 := 0 \ \forall l \in V$
for $t < \max_{iter}$ **do**
  CALL DP Algorithm 2.1 with selected relaxation, modified REFs and penalties $(\lambda_l^t)_{l \in V}$
  FIND a label $L_d^* = (|V| - 1, d, V, T_d) \in \mathcal{L}_{|V|-1}$ with T_d^{dur} minimal
  if $T_d^{dur} + \sum_{j \in V} \lambda_j^t > LB$ **then**
    SET $LB := T_d^{dur} + \sum_{j \in V} \lambda_j^t$, $\lambda^* := \lambda$
  COMPUTE ng-tour k and coefficients $(\delta_{lk})_{l \in V}$ corresponding to label L_d^*
  for $j \in V$ **do**
    SET
    $$\lambda_j^{t+1} := \left(\lambda_j^t - \left(LB - 1.2\left(LB + \sum_{l \in V} \delta_{lk} \lambda_l^t\right)\right)\right) \cdot (2 - 2\delta_{jk}) \,/\, \left(\sum_{l \in V} (2 - 2\delta_{lk})^2\right)$$
Result: Lower Bound LB and best computed penalties $(\lambda_l^*)_{l \in V}$

2.C Detailed Computational Results

This section reports, for all instances individually, the results produced with Algorithm 2.3. For each instance group a separate table is presented resulting in nine different tables (Tables 2.7–2.15). Herein, $|V|$ denotes the number of nodes of the instance and *UB BS* the upper bound computed with the VND (see Section 2.4). These are followed by columns LB, GAP and computation $Time$ for the column-generation algorithm (Step 5). The same information is shown for the dynamic augmentation of the neighborhoods (Step 6). The last two columns denote the computation time of the exact DP (Step 9) and the optimal solution value, if it is known. Otherwise, an interval $[LB, UB]$ for the optimum is given. (Note that the lower bound LB shown for the open instances may differ from the lower bound resulting from the neighborhood augmentation, since the exact DP improves lower bounds every time Step 10 is reached.) The symbol '†' indicates that an optimal solution or a better lower or upper bound, respectively, was computed by another setup during pre-tests. The entry '–' in a column means that invoking this part of the algorithm was not necessary, while 'TL' denotes that the time limit of 3600 seconds in this part of the algorithm was reached.

			Column Generation			Dyn. *ng* neighb. augment.			Exact DP	
Instance	$\|V\|$	UB BS	LB	GAP [%]	Time [s]	LB	GAP [%]	Time [s]	Time [s]	OPT
rc_201.1.txt	20	50352	50352.0	0.0	0.1	–	–	–	–	50352
rc_201.2.txt	26	75633	75633.0	0.0	0.0	–	–	–	–	75633
rc_201.3.txt	32	81605	81605.0	0.0	0.3	–	–	–	–	81605
rc_201.4.txt	26	81207	80999.0	0.3	0.1	81207.0	0.0	0.1	–	81207
rc_202.1.txt	33	78820	75720.2	1.9	15.8	76324.0	1.2	1.3	0.2	77215
rc_202.2.txt	14	31504	31504.0	0.0	0.2	–	–	–	–	31504
rc_202.3.txt	29	89203	87057.5	0.0	1.1	87069.0	0.0	0.3	–	87069
rc_202.4.txt	28	79446	77899.0	1.9	9.2	77899.0	1.9	0.1	0.2	79446
rc_203.1.txt	19	45340	45066.3	0.6	0.3	45340.0	0.0	0.1	–	45340
rc_203.2.txt	33	80942	80637.5	0.2	96.8	80637.5	0.2	1.8	0.2	80798
rc_203.3.txt	37	88733	84702.1	3.1	273.0	84883.7	2.9	100.7	6.7	87403
rc_203.4.txt	15	31841	31841.0	0.0	0.0	–	–	–	–	31841
rc_204.1.txt	46	89889	87687.1	0.4	1943.3	87687.1	0.4	64.3	126.8	88069
rc_204.2.txt	33	67462	65038.4	3.1	196.3	65060.3	3.1	8.6	TL	67120†
rc_204.3.txt	24	45495	43919.0	3.5	59.3	43919.0	3.5	1.1	0.2	45495
rc_205.1.txt	14	37549	37549.0	0.0	0.0	–	–	–	–	37549
rc_205.2.txt	27	79495	74189.0	5.9	0.1	74189.0	5.9	0.2	0.1	78869
rc_205.3.txt	35	82764	82738.1	0.0	20.2	82764.0	0.0	0.3	–	82764
rc_205.4.txt	28	78937	77051.0	2.4	0.2	77051.0	2.4	0.1	0.1	78937
rc_206.1.txt	4	11784	11784.0	0.0	0.0	–	–	–	–	11784
rc_206.2.txt	37	84349	79950.7	5.2	10.4	80949.0	4.0	1.2	0.4	84349
rc_206.3.txt	25	57723	57035.2	1.2	1.8	57565.0	0.3	0.2	0.1	57723
rc_206.4.txt	38	84770	80242.3	4.3	8.5	81334.5	3.0	12.2	0.2	83830
rc_207.1.txt	34	77056	72627.0	0.9	36.2	72627.0	0.9	0.2	0.1	73260
rc_207.2.txt	31	70347	62736.4	10.5	6.3	63298.0	9.7	8.7	1283.6	70116
rc_207.3.txt	33	73286	63113.3	7.5	22.3	63372.4	7.1	64.9	225.2	68230
rc_207.4.txt	6	11961	11961.0	0.0	0.0	–	–	–	–	11961
rc_208.1.txt	38	79904	68981.0	13.7	247.4	68981.0	13.7	0.3	TL	[73432,79904]
rc_208.2.txt	29	55478	51350.2	3.8	7.9	51656.0	3.2	41.9	8.1	53369
rc_208.3.txt	36	67902	57625.1	15.1	54.1	58407.4	14.0	84.2	TL	[61302,67902]

Table 2.7: Detailed Results of the Column-Generation Method for the Instances of `Potvin+Bengio`

			Column Generation			Dyn. *ng* neighb. augment.			Exact DP	
Instance	$\|V\|$	UB BS	LB	GAP [%]	Time [s]	LB	GAP [%]	Time [s]	Time [s]	OPT
rbg010a.tw	11	2975	2975.0	0.0	0.0	–	–	–	–	2975
rbg016a.tw	17	2465	2457.1	0.3	0.0	2465.0	0.0	0.1	–	2465
rbg016b.tw	17	1304	1299.0	0.4	0.2	1299.0	0.4	0.0	0.1	1304
rbg017.2.tw	16	1351	1351.0	0.0	0.1	–	–	–	–	1351
rbg017.tw	16	1756	1756.0	0.0	0.0	–	–	–	–	1756
rbg017a.tw	18	4296	4296.0	0.0	0.1	–	–	–	–	4296
rbg019a.tw	20	2448	2448.0	0.0	0.0	–	–	–	–	2448
rbg019b.tw	20	2975	2975.0	0.0	0.1	–	–	–	–	2975
rbg019c.tw	20	4536	4526.0	0.2	0.4	4526.0	0.2	0.1	0.1	4536
rbg019d.tw	20	2917	2917.0	0.0	0.0	–	–	–	–	2917
rbg020a.tw	21	4689	4689.0	0.0	0.0	–	–	–	–	4689
rbg021.2.tw	20	4528	4526.0	0.0	0.4	4526.0	0.0	0.2	0.1	4528
rbg021.3.tw	20	4528	4519.6	0.2	0.5	4520.0	0.2	0.2	0.1	4528
rbg021.4.tw	20	4525	4516.0	0.2	0.8	4516.0	0.2	0.0	0.1	4525
rbg021.5.tw	20	4516	4510.0	0.1	0.8	4510.0	0.1	0.1	0.1	4516
rbg021.6.tw	20	4489	4483.5	0.0	2.2	–	–	–	–	4484
rbg021.7.tw	20	4481	4479.0	0.0	2.8	4479.0	0.0	0.3	0.1	4479
rbg021.8.tw	20	4481	4478.0	0.0	2.2	4478.0	0.0	0.6	–	4478
rbg021.9.tw	20	4481	4478.0	0.0	2.2	4478.0	0.0	0.7	0.1	4478
rbg021.tw	20	4536	4526.0	0.2	0.4	4526.0	0.2	0.1	0.1	4536
rbg027a.tw	28	5093	5088.7	0.1	3.5	5090.0	0.1	0.5	0.1	5093
rbg031a.tw	32	2953	2953.0	0.0	1.0	–	–	–	–	2953
rbg033a.tw	34	3157	3157.0	0.0	1.3	–	–	–	–	3157
rbg034a.tw	35	2714	2714.0	0.0	0.5	–	–	–	–	2714
rbg035a.2.tw	36	2715	2715.0	0.0	5.0	–	–	–	–	2715
rbg035a.tw	36	2874	2874.0	0.0	2.6	–	–	–	–	2874
rbg038a.tw	39	5115	5115.0	0.0	1.4	–	–	–	–	5115
rbg040a.tw	41	5079	5079.0	0.0	0.1	–	–	–	–	5079
rbg050a.tw	51	11450	11450.0	0.0	10.5	–	–	–	–	11450
rbg055a.tw	56	6367	6367.0	0.0	4.5	–	–	–	–	6367
rbg067a.tw	68	9736	9736.0	0.0	0.3	–	–	–	–	9736
rbg125a.tw	126	13652	13652.0	0.0	70.6	–	–	–	–	13652

Table 2.8: Detailed Results of the Column-Generation Method for the `Ascheuer easy` Instances

			Column Generation			Dyn. *ng* neighb. augment.			Exact DP	
Instance	$\|V\|$	UB BS	LB	GAP [%]	Time [s]	LB	GAP [%]	Time [s]	Time [s]	OPT
rbg041a.tw	42	3245	3245.0	0.0	4.7	–	–	–	–	3245
rbg042a.tw	43	2962	2949.8	0.4	29.0	2959.0	0.1	1.9	0.3	2962
rbg048a.tw	49	9793	9793.0	0.0	2.6	–	–	–	–	9793
rbg049a.tw	50	12657	12657.0	0.0	1.0	–	–	–	–	12657
rbg050b.tw	51	11357	11357.0	0.0	53.7	–	–	–	–	11357
rbg050c.tw	51	10431	10431.0	0.0	76.9	–	–	–	–	10431
rbg086a.tw	87	16299	16299.0	0.0	0.2	–	–	–	–	16299
rbg092a.tw	93	11924	11924.0	0.0	1.0	–	–	–	–	11924
rbg132.2.tw	131	17524	17524.0	0.0	9.0	–	–	–	–	17524
rbg132.tw	131	17929	17929.0	0.0	0.8	–	–	–	–	17929
rbg152.3.tw	151	16455	16455.0	0.0	74.8	–	–	–	–	16455
rbg152.tw	151	17019	17019.0	0.0	72.5	–	–	–	–	17019
rbg172a.tw	173	17221	17213.5	0.0	598.6	17220.1	0.0	135.8	–	17221
rbg193.2.tw	192	20401	20401.0	0.0	44.2	–	–	–	–	20401
rbg193.tw	192	20869	20868.8	0.0	625.5	–	–	–	–	20869
rbg201a.tw	202	20818	20818.0	0.0	306.7	–	–	–	–	20818
rbg233.2.tw	232	25143	25143.0	0.0	39.5	–	–	–	–	25143
rbg233.tw	232	25691	25691.0	0.0	531.9	–	–	–	–	25691

Table 2.9: Detailed Results of the Column-Generation Method for the `Ascheuer hard` Instances

			Column Generation			Dyn. *ng* neighb. augment.			Exact DP	
Instance	$\|V\|$	UB BS	LB	GAP [%]	Time [s]	LB	GAP [%]	Time [s]	Time [s]	OPT
n20w120.001.txt	21	296	295.3	0.2	0.1	–	–	–	–	296
n20w120.002.txt	21	220	217.3	1.2	0.4	220.0	0.0	0.1	–	220
n20w120.003.txt	21	303	303.0	0.0	0.0	–	–	–	–	303
n20w120.004.txt	21	312	308.0	1.3	0.5	308.0	1.3	0.1	0.1	312
n20w120.005.txt	21	277	274.3	1.0	0.8	276.6	0.1	0.1	–	277
n20w140.001.txt	21	188	187.3	0.4	0.2	–	–	–	–	188
n20w140.002.txt	21	280	277.0	1.1	0.1	–	–	–	–	280
n20w140.003.txt	21	252	251.3	0.3	0.8	–	–	–	–	252
n20w140.004.txt	21	280	275.5	1.6	0.4	280.0	0.0	0.1	–	280
n20w140.005.txt	21	231	230.3	0.3	0.5	–	–	–	–	231
n20w160.001.txt	21	284	283.0	0.4	0.7	284.0	0.0	0.1	–	284
n20w160.002.txt	21	205	204.7	0.1	0.1	–	–	–	–	205
n20w160.003.txt	21	277	275.7	0.5	0.2	277.0	0.0	0.0	–	277
n20w160.004.txt	21	222	222.0	0.0	0.3	–	–	–	–	222
n20w160.005.txt	21	284	283.2	0.3	0.8	–	–	–	–	284
n20w180.001.txt	21	303	285.0	5.9	0.9	302.5	0.2	0.2	–	303
n20w180.002.txt	21	319	302.6	1.4	0.7	307.0	0.0	0.2	–	307
n20w180.003.txt	21	273	270.0	1.1	0.3	270.0	1.1	0.1	0.1	273
n20w180.004.txt	21	234	232.0	0.9	0.9	234.0	0.0	0.1	–	234
n20w180.005.txt	21	207	199.6	0.7	1.3	201.0	0.0	0.1	–	201
n20w200.001.txt	21	233	230.5	1.1	0.7	232.0	0.4	0.1	0.1	233
n20w200.002.txt	21	211	209.0	0.9	1.6	209.0	0.9	0.0	0.1	211
n20w200.003.txt	21	271	254.8	2.8	0.7	262.0	0.0	0.2	–	262
n20w200.004.txt	21	320	279.7	7.1	0.7	285.0	5.3	0.8	0.1	301
n20w200.005.txt	21	229	226.0	0.4	0.5	–	–	–	–	227

Table 2.10: Detailed Results of the Column-Generation Method for the `Gendreau 20` Instances

			Column Generation			Dyn. *ng* neighb. augment.			Exact DP	
Instance	$\|V\|$	UB BS	LB	GAP [%]	Time [s]	LB	GAP [%]	Time [s]	Time [s]	OPT
n40w120.001.txt	41	446	431.0	3.4	63.1	431.0	3.4	0.2	0.2	446
n40w120.002.txt	41	514	509.4	0.9	42.3	513.1	0.2	6.1	–	514
n40w120.003.txt	41	420	412.3	1.1	51.8	416.0	0.2	9.8	0.3	417
n40w120.004.txt	41	347	343.8	0.9	16.9	346.8	0.0	1.6	–	347
n40w120.005.txt	41	418	417.3	0.2	9.1	–	–	–	–	418
n40w140.001.txt	41	402	401.0	0.2	18.5	–	–	–	–	402
n40w140.002.txt	41	401	401.0	0.0	2.6	–	–	–	–	401
n40w140.003.txt	41	429	428.0	0.2	42.8	–	–	–	–	429
n40w140.004.txt	41	400	399.1	0.2	76.8	–	–	–	–	400
n40w140.005.txt	41	390	389.1	0.2	41.6	–	–	–	–	390
n40w160.001.txt	41	418	418.0	0.0	18.0	–	–	–	–	418
n40w160.002.txt	41	388	383.7	1.1	63.5	388.0	0.0	2.9	–	388
n40w160.003.txt	41	368	367.2	0.2	15.8	–	–	–	–	368
n40w160.004.txt	41	361	356.3	0.2	252.5	357.0	0.0	14.3	–	357
n40w160.005.txt	41	316	315.1	0.3	213.5	–	–	–	–	316
n40w180.001.txt	41	399	388.8	1.3	125.8	389.8	1.1	138.2	11.6	394
n40w180.002.txt	41	379	378.1	0.2	38.4	–	–	–	–	379
n40w180.003.txt	41	346	345.5	0.1	45.6	–	–	–	–	346
n40w180.004.txt	41	378	369.0	0.3	71.0	–	–	–	–	370
n40w180.005.txt	41	363	362.0	0.3	301.3	–	–	–	–	363
n40w200.001.txt	41	345	339.0	0.0	537.4	339.0	0.0	11.9	–	339
n40w200.002.txt	41	343	337.2	1.4	222.3	337.2	1.4	49.7	0.2	342
n40w200.003.txt	41	391	342.2	1.7	77.4	344.8	0.9	135.9	5.0	348
n40w200.004.txt	41	377	368.3	0.5	33.8	369.3	0.2	25.4	120.9	370
n40w200.005.txt	41	350	347.0	0.9	77.1	347.0	0.9	269.7	TL	$347^{\dagger}$

Table 2.11: Detailed Results of the Column-Generation Method for the `Gendreau 40` Instances

			Column Generation			Dyn. *ng* neighb. augment.			Exact DP	
Instance	$\|V\|$	UB BS	LB	GAP [%]	Time [s]	LB	GAP [%]	Time [s]	Time [s]	OPT
n60w120.001.txt	61	484	483.0	0.2	268.9	483.0	0.2	36.9	55.0	484
n60w120.002.txt	61	553	552.1	0.2	265.2	–	–	–	–	553
n60w120.003.txt	61	488	488.0	0.0	2.1	–	–	–	–	488
n60w120.004.txt	61	556	556.0	0.0	186.8	–	–	–	–	556
n60w120.005.txt	61	549	548.2	0.1	526.4	–	–	–	–	549
n60w140.001.txt	61	560	560.0	0.0	175.5	–	–	–	–	560
n60w140.002.txt	61	597	594.1	0.5	230.1	595.3	0.3	26.4	8.7	597
n60w140.003.txt	61	567	567.0	0.0	266.1	–	–	–	–	567
n60w140.004.txt	61	567	566.0	0.2	510.3	–	–	–	–	567
n60w140.005.txt	61	501	497.0	0.0	310.0	497.0	0.0	42.7	16.1	497
n60w160.001.txt	61	614	614.0	0.0	671.5	–	–	–	–	614
n60w160.002.txt	61	614	614.0	0.0	13.7	–	–	–	–	614
n60w160.003.txt	61	507	507.0	0.0	851.4	–	–	–	–	507
n60w160.004.txt	61	505	504.0	0.2	633.2	–	–	–	–	505
n60w160.005.txt	61	561	560.3	0.1	698.9	–	–	–	–	561
n60w180.001.txt	61	488	487.7	0.1	172.5	–	–	–	–	488
n60w180.002.txt	61	503	501.5	0.3	398.5	502.0	0.2	19.1	–	503
n60w180.003.txt	61	526	525.1	0.2	282.1	–	–	–	–	526
n60w180.004.txt	61	577	577.0	0.0	207.1	–	–	–	–	577
n60w180.005.txt	61	486	466.0	4.1	1818.0	466.0	4.1	425.1	TL	466†
n60w200.001.txt	61	465	446.5	3.8	2642.4	451.1	2.8	264.8	3108.7	464
n60w200.002.txt	61	504	503.1	0.2	545.4	–	–	–	–	504
n60w200.003.txt	61	525	494.5	5.8	3308.1	496.2	5.5	573.9	TL	497†
n60w200.004.txt	61	512	511.1	0.2	1740.1	–	–	–	–	512
n60w200.005.txt	61	545	545.0	0.0	1771.4	–	–	–	–	545

Table 2.12: Detailed Results of the Column-Generation Method for the `Gendreau 60` Instances

			Column Generation			Dyn. *ng* neighb. augment.			Exact DP	
Instance	$\|V\|$	UB BS	LB	GAP [%]	Time [s]	LB	GAP [%]	Time [s]	Time [s]	OPT
n80w100.001.txt	81	664	660.2	0.4	919.5	661.5	0.2	183.7	458.2	663
n80w100.002.txt	81	707	683.0	0.0	415.6	683.0	0.0	113.5	0.9	683
n80w100.003.txt	81	717	717.0	0.0	11.0	–	–	–	–	717
n80w100.004.txt	81	753	753.0	0.0	3.5	–	–	–	–	753
n80w100.005.txt	81	664	661.5	0.4	181.6	661.5	0.4	167.7	0.3	664
n80w120.001.txt	81	620	620.0	0.0	910.5	–	–	–	–	620
n80w120.002.txt	81	695	695.0	0.0	16.1	–	–	–	–	695
n80w120.003.txt	81	622	621.0	0.2	2999.2	–	–	–	–	622
n80w120.004.txt	81	591	590.1	0.2	2331.0	–	–	–	–	591
n80w120.005.txt	81	690	689.0	0.1	2332.5	–	–	–	–	690
n80w140.001.txt	81	635	635.0	0.0	838.8	–	–	–	–	635
n80w140.002.txt	81	591	588.0	0.5	588.5	588.0	0.5	525.3	TL	589†
n80w140.003.txt	81	617	612.7	0.7	TL	614.4	0.4	857.2	TL	[615,617]
n80w140.004.txt	81	561	549.0	2.1	TL	549.0	2.1	81.8	TL	550†
n80w140.005.txt	81	687	685.0	0.3	638.0	685.0	0.3	80.6	0.3	687
n80w160.001.txt	81	564	561.7	0.4	TL	563.1	0.2	1090.0	–	564
n80w160.002.txt	81	609	601.5	1.2	TL	602.0	1.1	2400.0	TL	[603,609]
n80w160.003.txt	81	638	628.2	1.5	TL	630.0	1.2	1380.0	TL	[633,638]
n80w160.004.txt	81	596	596.0	0.0	2033.0	–	–	–	–	596
n80w160.005.txt	81	584	573.7	1.8	1689.5	573.7	1.8	295.5	TL	[583,584]†
n80w180.001.txt	81	617	610.2	0.5	TL	610.7	0.4	631.9	TL	613†
n80w180.002.txt	81	570	555.7	2.5	TL	559.5	1.8	2529.1	TL	[564,570]
n80w180.003.txt	81	623	623.0	0.0	3553.7	–	–	–	–	623
n80w180.004.txt	81	592	592.0	0.0	2641.0	–	–	–	–	592
n80w180.005.txt	81	571	568.3	0.5	TL	569.4	0.3	1115.5	TL	[570,571]
n80w200.001.txt	81	584	557.3	4.6	TL	558.8	4.3	2934.9	TL	[563,567]†
n80w200.002.txt	81	550	545.4	0.8	TL	548.5	0.3	1541.3	TL	[549,550]
n80w200.003.txt	81	617	617.0	0.0	74.6	–	–	–	–	617
n80w200.004.txt	81	611	606.1	0.8	TL	608.3	0.4	1625.8	TL	611†
n80w200.005.txt	81	564	561.0	0.5	1707.6	561.0	0.5	522.6	0.3	564

Table 2.13: Detailed Results of the Column-Generation Method for the `Gendreau 80` Instances

			Column Generation			Dyn. *ng* neighb. augment.			Exact DP	
Instance	$\|V\|$	UB BS	LB	GAP [%]	Time [s]	LB	GAP [%]	Time [s]	Time [s]	OPT
n100w120.001.txt	101	825	825.0	0.0	783.9	–	–	–	–	825
n100w120.002.txt	101	846	843.0	0.4	853.2	843.0	0.4	386.6	TL	843†
n100w120.003.txt	101	852	852.0	0.0	41.9	–	–	–	–	852
n100w120.004.txt	101	868	868.0	0.0	923.0	–	–	–	–	868
n100w120.005.txt	101	806	806.0	0.0	545.1	–	–	–	–	806
n100w140.001.txt	101	956	956.0	0.0	42.9	–	–	–	–	956
n100w140.002.txt	101	949	948.0	0.1	2306.6	948.0	0.1	415.0	TL	[948,949]
n100w140.003.txt	101	783	783.0	0.0	3114.7	–	–	–	–	783
n100w140.004.txt	101	791	791.0	0.0	39.6	–	–	–	–	791
n100w140.005.txt	101	733	731.0	0.3	TL	731.0	0.3	293.3	TL	733†
n100w160.001.txt	101	802	802.0	0.0	2928.1	–	–	–	–	802
n100w160.002.txt	101	731	729.8	0.2	TL	730.1	0.1	29.7	–	731
n100w160.003.txt	101	882	882.0	0.0	52.7	–	–	–	–	882
n100w160.004.txt	101	751	751.0	0.0	2711.1	–	–	–	–	751
n100w160.005.txt	101	855	855.0	0.0	1344.2	–	–	–	–	855

Table 2.14: Detailed Results of the Column-Generation Method for the `Gendreau 100` Instances

			Column Generation			Dyn. *ng* neighb. augment.			Exact DP	
Instance	$\|V\|$	UB BS	LB	GAP [%]	Time [s]	LB	GAP [%]	Time [s]	Time [s]	OPT
n150w120.001.txt	151	9200	9098.0	1.1	TL	9098.0	1.1	860.5	TL	[9198,9200]†
n150w120.002.txt	151	8692	8670.0	0.3	TL	8670.0	0.3	1381.3	TL	[8686,8692]†
n150w120.003.txt	151	8611	8477.0	1.6	TL	8477.0	1.6	TL	TL	[8598,8611]
n150w120.004.txt	151	8759	8648.0	1.3	TL	8648.0	1.3	1379.9	TL	[8751,8759]
n150w120.005.txt	151	8555	8509.0	0.5	TL	8509.0	0.5	991.4	TL	8555†
n150w140.001.txt	151	9560	9560.0	0.0	729.1	9560.0	0.0	3.3	–	9.560
n150w140.002.txt	151	9811	9596.0	2.2	TL	9596.0	2.2	TL	TL	9722†
n150w140.003.txt	151	7915	7860.0	0.7	TL	7860.0	0.7	TL	TL	[7896,7904]†
n150w140.004.txt	151	8494	8210.0	3.3	TL	8210.0	3.3	TL	TL	[8300,8494]†
n150w140.005.txt	151	8712	8448.0	3.0	TL	8448.0	3.0	2834.2	TL	[8707,8712]†
n150w160.001.txt	151	9062	8953.0	1.2	TL	8953.0	1.2	TL	TL	[8953,9032]†
n150w160.002.txt	151	8905	8023.0	9.9	TL	8023.0	9.9	TL	TL	[8023,8410]†
n150w160.003.txt	151	8886	8827.0	0.7	TL	8827.0	0.7	1283.5	TL	[8827,8830]†
n150w160.004.txt	151	8632	8404.0	2.6	TL	8404.0	2.6	TL	TL	[8539,8620]†
n150w160.005.txt	151	8669	8530.0	1.6	TL	8530.0	1.6	TL	TL	[8530,8643]†
n200w120.001.txt	201	10454	10360.0	0.9	TL	10360.0	0.9	TL	TL	[10410,10454]
n200w120.002.txt	201	10225	10150.0	0.7	TL	10150.0	0.7	1224.9	TL	10225†
n200w120.003.txt	201	10810	10734.0	0.7	TL	10734.0	0.7	1720.9	TL	[10777,10810]
n200w120.004.txt	201	10177	10146.0	0.3	TL	10146.0	0.3	917.5	TL	[10146,10177]
n200w120.005.txt	201	10385	10195.0	1.8	TL	10195.0	1.8	1344.1	TL	[10197,10252]†
n200w140.001.txt	201	10893	10813.0	0.7	TL	10813.0	0.7	TL	TL	[10813,10893]
n200w140.002.txt	201	10398	10078.0	3.1	TL	10078.0	3.1	2986.3	TL	[10284,10398]
n200w140.003.txt	201	10404	10290.0	1.1	TL	10290.0	1.1	TL	TL	[10290,10404]
n200w140.004.txt	201	10518	10416.0	1.0	TL	10416.0	1.0	1051.3	TL	[10416,10518]
n200w140.005.txt	201	10743	10652.0	0.8	TL	10652.0	0.8	TL	TL	[10692,10700]†

Table 2.15: Detailed Results of the Column-Generation Method for the `Ohlmann+ Thomas` Instances

Chapter 3

Branch-and-Price-and-Cut for the Active-Passive Vehicle-Routing Problem

Christian Tilk, Nicola Bianchessi, Michael Drexl,
Stefan Irnich, Frank Meisel

Abstract

This paper presents a branch-and-price-and-cut algorithm for the exact solution of the active-passive vehicle-routing problem (APVRP). The APVRP covers a range of logistics applications where pickup-and-delivery requests necessitate a joint operation of active vehicles (e.g., trucks) and passive vehicles (e.g., loading devices such as containers or swap bodies). The objective is to minimize a weighted sum of the total distance traveled, the total completion time of the routes, and the number of unserved requests. To this end, the problem supports a flexible coupling and decoupling of active and passive vehicles at customer locations. Accordingly, the operations of the vehicles have to be synchronized carefully in the planning. The contribution of the paper is twofold: Firstly, we present an exact branch-and-price-and-cut algorithm for this class of routing problems with synchronization constraints. To our knowledge, this algorithm is the first such approach that considers explicitly the temporal interdependencies between active and passive vehicles. The algorithm is based on a non-trivial network representation that models the logical relationships between the different transport tasks necessary to fulfill a request as well as the synchronization of the movements of active and passive vehicles. Secondly, we contribute to the development of branch-and-price methods in general, in that we solve, for the first time, an ng-path relaxation of a pricing problem with linear vertex costs by means of a bidirectional labeling algorithm. Computational experiments show that the proposed algorithm delivers improved bounds and solutions for a number of APVRP benchmark instances. It is able to solve instances with up to 76 tasks, 4 active, and 8 passive vehicles to optimality within two hours of CPU time.

3.1 Introduction

Many applications in the area of transport logistics involve problems in which the execution of transport requests calls for a joint operation of different resources such as trucks,

tractors, drivers, trailers, semi-trailers, swap bodies, containers, or accompanying staff. A typical example is found in the transportation of containerized goods, where not just a manned truck but also an empty container is required for executing a request. Further examples are found in the health care sector or in the security industry, where the transportation of patients and valuable items must be accompanied by medics and security guards respectively. In general, we can distinguish two classes of transport resources. The first class is constituted by means of transport that can move on their own from one location to another such as, for example, manned trucks. We refer to these resources as *active vehicles*. The second class consists of resources that cannot move autonomously but require an active vehicle for being repositioned. This class comprises trailers, semi-trailers, all kinds of loading devices, equipment, and accompanying staff. For simplicity, we refer to all these resources jointly as *passive vehicles*.

In the literature on vehicle-routing problems (VRPs, see Irnich *et al.*, 2014, for an overview), the distinction of active and passive vehicles is usually ignored, and operations are planned for active vehicles only (cf. the recent survey by Lahyani *et al.*, 2015). This restricts the applicability of the developed models and algorithms to real-world problems in which either passive vehicles do not play a role or where active and passive vehicles are paired to fixed units. Although the latter eases the solution of the VRP, it may hinder an effective utilization of the resources. If, for example, a manned truck (active vehicle) and an empty container (passive vehicle) are considered a unit in operations planning, the truck and its driver have to wait at a customer location while the container is being stuffed. Therefore, in order to support a more flexible use of such resources, Meisel and Kopfer (2014) introduced what we denote here as the active-passive vehicle-routing problem (APVRP), in which an explicit distinction between active and passive vehicles is made. This distinction enables the modeling of complex transport operations in which an active vehicle carries a passive vehicle (e.g., an empty container) to some pickup location, drops it off there, and leaves this location for performing transports of other passive vehicles elsewhere. Later, when the container has been stuffed, the same or some other active vehicle returns to the customer, picks up the container, and carries it to the delivery location. The problem introduces multiple interdependencies between vehicles and raises the need to synchronize the operations and the movements of active and passive vehicles in time and space. According to the survey by Drexl (2012b), such a combination of synchronization requirements is rarely addressed in the VRP literature. This fact is in marked contrast to the abovementioned practical relevance of the APVRP. It therefore seems appropriate to devote further studies to this generic and complex problem.

Our paper addresses this research gap and provides a twofold contribution: Firstly, we present an exact algorithm for the APVRP which, to the best of our knowledge, is the first branch-and-price-and-cut approach that considers explicitly the temporal interdependencies between active and passive vehicles and the resulting synchronization requirements. Because of these synchronization requirements, the adaptation of the branch-and-price-and-cut concept to the APVRP is not straightforward. Our algorithm is based on an extended set-partitioning formulation which, in turn, uses a non-trivial network representation that models the logical relationships between the different trans-

port tasks necessary to fulfill a request as well as the synchronization of the movements of active and passive vehicles. Secondly, we contribute to the development of branch-and-price methods for routing with synchronization in general: We provide solutions to the pricing subproblem, which is an elementary shortest-path problem with time windows and with linear vertex costs, by solving, for the first time, its ng-path relaxation (Baldacci *et al.*, 2011) by means of a bidirectional labeling algorithm. We actually apply a refined ng-path relaxation, taking into account partial requests (henceforth called *tasks*) and precedences between these tasks instead of individual vertices or complete requests. The pricing problem structure and the use of the ng-path relaxation, moreover, require a sophisticated merge step in the labeling algorithm. Computational experiments show that the proposed algorithm delivers improved bounds and solutions for the APVRP benchmark suite of Meisel and Kopfer (2014), solving instances with up to 76 tasks, 4 active, and 8 passive vehicles to optimality within two hours of CPU time.

The paper is organized as follows. Related literature is reviewed in Section 3.2. In Section 3.3, we formally describe the APVRP. A corresponding extended set-partitioning formulation is provided in Section 3.4. The branch-and-price-and-cut algorithm is presented in Section 3.5, and the method is computationally evaluated in Section 3.6. Finally, Section 3.7 summarizes the paper and discusses potential avenues for further research.

3.2 Literature

The manifold real-world logistics applications for vehicle-routing problems with synchronization requirements have motivated several studies. Most papers consider applications in which active vehicles have to be synchronized. One example is the operations planning for cross-docks where different trucks deliver less-than-truckload shipments that are then merged to full-truckload shipments before being sent out to customers (see, e.g., Buijs *et al.*, 2014; Morais *et al.*, 2014). Another active research field is found in the management of home care operations. Here, service operations have to be synchronized if a person requires the help of two caregivers at the same time (see Bredström and Rönnqvist, 2008; Mankowska *et al.*, 2013; Labadie *et al.*, 2014; Afifi *et al.*, 2015). Further research addresses forestry applications where trucks have to be served by forest loaders (Hachemi *et al.*, 2013), intermodal transportation via ships, trains, and trucks for the supply of automotive factories (Mues and Pickl, 2005), scheduling of cooperating technician teams at customer locations (Dohn *et al.*, 2009), and ship routing where cargoes from different origins have to be delivered to one and the same client simultaneously (Andersson *et al.*, 2011). Eventually, there are also arc routing problems that include synchronization of active vehicles, e.g., when planning snow plowing operations (see Laporte, 2016) and road marking (see Salazar-Aguilar *et al.*, 2013).

The mentioned applications all involve active vehicles that move autonomously to those locations where synchronized services are required. By contrast, in the APVRP, an active vehicle and a passive vehicle must both traverse route segments synchronously in order to perform a service. Typical applications are found in routing problems where

trucks pull trailers or swap bodies (see Smilowitz, 2006; Cheung *et al.*, 2008; Drexl, 2013). Obviously, trailers and swap bodies are passive vehicles that are immobile without a truck. Usually, they play the role of optional capacity extensions of trucks rather than being mandatory for the execution of transport operations. An exceptional case is when an individual customer has such a large demand that it must be served jointly by a truck with a swap body (see Huber and Geiger, 2014). A further application of such a synchronization requirement is found in the drayage operations of container terminals, where empty and loaded containers are moved between customer locations and transshipment points. In this field of logistics, the containers constitute the passive vehicles that are mandatory for the transport of goods (see Cheung *et al.*, 2008; Xue *et al.*, 2014; Zhang *et al.*, 2010, 2013, 2014). Movement synchronization en route is also found in VRPs where driver crews can be assigned flexibly to trucks (see Hollis *et al.*, 2006; Drexl *et al.*, 2013). In such problems, the exchange of crews enables a better utilization of the trucks in compliance with work regulations for truck drivers. Further applications of a synchronization of vehicles and crews are considered in the papers by Kim *et al.* (2010), where technicians have to be carried to customer locations, and by Kergosien *et al.* (2011, 2013), where ambulances carry patients and accompanying physicians from one care unit to another.

Drexl (2007, 2014) studies the VRP with trailers and transshipments (VRPTT), a problem which also requires the synchronization of operations and movements of active and passive vehicles. Two branch-and-cut algorithms are presented, but only very small instances can be solved. For a deeper investigation of VRPs with synchronization of operations and vehicle movements, Meisel and Kopfer (2014) provide mixed-integer programming (MIP) formulations, a branch-and-cut algorithm, an adaptive large neighborhood search (ALNS) metaheuristic, and benchmark instances for the APVRP. To our knowledge, the branch-and-price algorithm by Smilowitz (2006) is the only other exact method for the APVRP so far. To facilitate understanding, the differences between the algorithm of Smilowitz (2006) and ours as well as the extensions and improvements our approach provides will be discussed in Section 3.5.2, when the difficulties arising from the synchronization requirements will have been thoroughly explained.

A few other papers provide exact methods (typically based on column generation techniques) for VRPs with synchronization requirements, but merely for problems that involve active vehicles only (see Mues and Pickl, 2005; Dohn *et al.*, 2009, 2011; Andersson *et al.*, 2011). However, the features of the VRPTT and the APVRP, which include simultaneous operations planning of active and passive vehicles together with the possibility to couple and decouple them flexibly on their routes, are not supported by any of these approaches.

3.3 Problem Description and Modeling

The problem considered in this paper can be formally described as follows. We are given a set of pickup-and-delivery requests R, a set A of classes of active vehicles and a set P of passive vehicles. For each class $a \in A$ of active vehicles, K_a denotes the number of

vehicles in the class. Each request $r \in R$ consists of transporting a loading unit from a pickup location ℓ_r^+ to a delivery location ℓ_r^-. To fulfill a request r, an active vehicle must carry a passive vehicle to ℓ_r^+ for loading. Each passive vehicle can load only one request at a time, and each active vehicle can transport only one passive vehicle at a time. Hence, the loaded passive vehicle must then be transported directly to ℓ_r^- for unloading. Afterwards, the empty passive vehicle must be carried away from ℓ_r^-. From the point of view of an active vehicle, the fulfillment of a request comprises three transport tasks:

i) providing an empty passive vehicle at the pickup location of a request,

ii) direct transport of a loaded passive vehicle from the pickup to the delivery location, and

iii) carrying away the emptied passive vehicle from the delivery location.

One or two or three different active vehicles may perform these tasks for a request.

There are compatibility relationships between the requests and the active and the passive vehicles. P^r denotes the set of passive vehicles that can be used to perform request r. Likewise, R^p denotes the set of requests that can be performed with passive vehicle p. P^a indicates the set of passive vehicles that can be carried by an active vehicle from class a, and A^p is the set of classes of active vehicles compatible with passive vehicle p.

All active vehicles are initially based at the same start depot o and end their routes at the same end depot d. Each passive vehicle p has its own start and end positions: It is initially located at o_p and must be brought to d_p at the end, whether or not it is used to fulfill a request. Hence, there are two tasks associated with each passive vehicle p, namely, to pickup and to deliver the vehicle at o_p and d_p respectively. (Aggregating identical active vehicles into classes while at the same time considering individual passive vehicles is convenient when setting up the network on which our problem formulation is based, ensures that each network arc is traversed by at most one active and/or at most one passive vehicle in any feasible solution, and makes branching easier.)

Let s_r^+ indicate the time necessary to load an empty passive vehicle with request r and s_r^- indicate the time to unload r. Times for coupling and uncoupling of passive vehicles to or from active vehicles are assumed to be zero (but could easily be incorporated into the model). Picking up a loaded request r at its pickup location can be finished no earlier than at time e_r. Waiting is allowed. Unloading a request r at its delivery location must be finished no later than l_r. The overall planning horizon is $[0, t^{max}]$. Requests that cannot be fulfilled imply a penalty.

The objective is to minimize a weighted sum of the total distance traveled, the total completion time of the routes, and the number of unserved requests. The respective weights are $\alpha, \beta, \gamma \in \mathbb{R}_+$.

The APVRP as described above can be modeled as an optimization problem over a set of graphs $G^a = (V^a, E^a)$, one for each class a of active vehicles. Each set V^a of vertices contains o and d, i.e., the initial and the final location of all active vehicles, o_p and d_p, i.e., the initial and the final locations of all passive vehicles $p \in P^a$, and the

set N^a containing the following four vertices for each passive vehicle $p \in P^a$ and each compatible request $r \in R^p$:

v^-_{rp}	delivery of an empty passive vehicle p into which request r is loaded;
w^+_{rp}	pickup of request r loaded in passive vehicle p;
w^-_{rp}	delivery of request r loaded in passive vehicle p;
v^+_{rp}	pickup of the empty passive vehicle p in which request r was transported.

Thus $N^a = \cup_{p\in P^a, r\in R^p}\{v^-_{rp}, w^+_{rp}, w^-_{rp}, v^+_{rp}\}$. For each request $r \in R$, we define $V^-_r = \{v^-_{rp} : p \in P^r\}$, $W^+_r = \{w^+_{rp} : p \in P^r\}$, $W^-_r = \{w^-_{rp} : p \in P^r\}$, and $V^+_r = \{v^+_{rp} : p \in P^r\}$. The vertices in $N^R = \bigcup_{r\in R}(V^-_r \cup W^+_r \cup W^-_r \cup V^+_r)$ are henceforth referred to as *request vertices*.

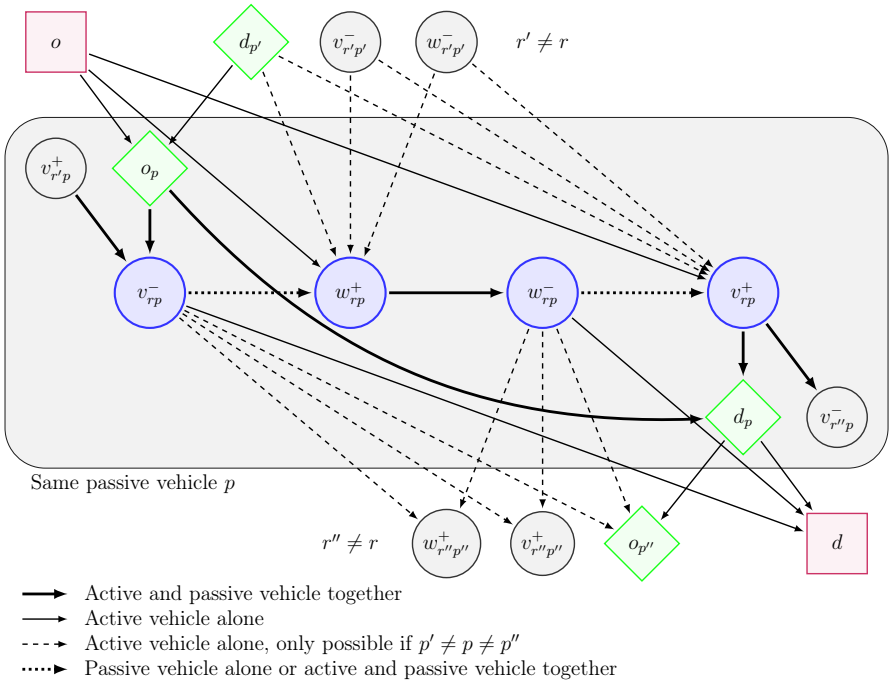

Figure 3.1: Arc Set of Graph $G^a = (V^a, E^a)$

Arcs in E^a between each pair of vertex types are represented in Figure 3.1. In the figure, solid bold arcs (⟶) can be traversed by an active vehicle while traveling together with a passive vehicle attached. Dotted arcs (⋯▸) are service arcs and are traversed by the same passive vehicle p alone or by p together with an active vehicle, while solid and dashed arcs (⟶ and - -▸) are traversed by the active vehicle alone. In particular, all dashed arcs (- -▸) exist only if $p' \neq p \neq p''$. For the four request vertices in the shaded rectangle, all types of ingoing and outgoing arcs are depicted. It is easy to see that the graphs constructed in this way allow an active vehicle to perform any of the three transport tasks of any compatible request. For example, assume that active vehicle a_1 performs the first task of request r, and that active vehicle a_2 performs the second and

the third task of r, i.e., vehicle a_2 transports some passive vehicle p loaded with r from w^+_{rp} to w^-_{rp} and afterwards moves away the empty passive vehicle. In this case, a_2 reaches w^+_{rp} without a passive vehicle, coming from either o or from a vertex $v^-_{r'p'}$, $w^-_{r'p'}$, or $d_{p'}$, with $r' \neq r$ and $p' \neq p$. (If $p' = p$, $d_{p'}$ is the final location of passive vehicle p, and an arc from $d_{p'}$ to w^+_{rp} for some $r \in R$ is impossible because of the uniqueness of the passive vehicles.) After that, a_2 visits vertices w^-_{rp} and v^+_{rp} and leaves v^+_{rp} heading towards either d_p or towards a vertex $v^-_{r''p}$ with $r \neq r' \neq r'' \neq r$. The travel distances between any pair $(i, j) \in E^a$ of vertices are denoted by c_{ij} and are given by the distance between the associated locations. The same holds for the travel and service time t_{ij} for $(i, j) \in E^a$ with t_{ij} equal to s^+_r and s^-_r for arcs (v^-_{rp}, w^+_{rp}) and (w^-_{rp}, v^+_{rp}) respectively. Following the specifications given by Meisel and Kopfer (2014), the time windows at the vertices are given in Table 3.1. The complete notation used throughout the paper is also summarized in the Appendix.

Vertex i	Earliest time e_i	Latest time l_i
v^-_{rp}	$t_{oo_p} + t_{o_p v^-_{rp}}$	$l_{w^+_{rp}} - (s^+_r + t_{w^+_{rp} w^-_{rp}} + s^-_r)$
w^+_{rp}	$\max\{e_{v^-_{rp}} + s^+_r, e_r\}$	$l_{w^-_{rp}} - (t_{w^+_{rp} w^-_{rp}} + s^-_r)$
w^-_{rp}	$e_{w^+_{rp}} + t_{w^+_{rp} w^-_{rp}}$	$\min\{l_{v^+_{rp}}, l_r\} - s^-_r$
v^+_{rp}	$e_{w^-_{rp}} + s^-_r$	$t^{max} - (t_{v^+_{rp} d_p} + t_{d_p d})$
Other	0	t^{max}

Table 3.1: Vertex Time Windows

The request vertices take up an idea described by Drexl (2007, Sect. 4.3.3). They essentially correspond to pairs of operations and passive vehicles. This ensures that the itineraries of the passive vehicles are implicit, i.e., need not be determined explicitly, but can be unequivocally reconstructed from given routes for the active vehicles. Hence, a feasible solution to the APVRP is a set of scheduled routes for the active vehicles that fulfills:

- Each route starts at vertex o and terminates at vertex d.
- All visited vertices are visited within their prescribed time windows.
- For each $r \in R$, at most one vertex per set is visited in sets V^-_r, W^+_r, and V^+_r. This ensures the synchronization of task fulfillment of requests within and between active vehicles. Note that it is not necessary to impose that at most one vertex is visited in set W^-_r, because vertices in this set can be reached only via an arc coming from a vertex in W^+_r.
- For each $r \in R$, $p \in P^r$, vertex w^+_{rp} is visited if and only if vertex v^-_{rp} has been visited, and vertex v^+_{rp} is visited if and only if vertex w^+_{rp} has been visited. This requirement is necessary to preserve consistency with respect to the passive vehicle used to satisfy a request in its different stages. Moreover, given that the final locations d_p of any passive

vehicle $p \in P$ can only be reached from the corresponding initial location o_p or from vertices in $\bigcup_{r \in R} V_r^+$, this requirement preserves the flow of the unique passive vehicle used to satisfy a request through vertices v_{rp}^-, w_{rp}^+, w_{rp}^-, and v_{rp}^+.

- $T_{v_r^-} + s_r^+ \leq T_{w_r^+}$ and $T_{w_r^-} + s_r^- \leq T_{v_r^+}$ for each $r \in R$, where $T_{v_r^-}$, $T_{w_r^+}$, $T_{w_r^-}$, and $T_{v_r^+}$ are the service start times at the vertices possibly selected in sets V_r^-, W_r^+, W_r^-, and V_r^+ respectively. This ensures temporal synchronization of tasks within and between active vehicles.
- Each passive vehicle is picked up at its initial location and placed at its final location.
- Each active vehicle performs at most one feasible route (the trivial route from o to d does not exist), so that at most K_a routes are performed for each class a of active vehicles.

The first two requirements are intra-route constraints, the subsequent four represent both intra-route and inter-route constraints, and the last one is an inter-route constraint.

Note that, along a route, an active vehicle of class $a \in A$ can be associated with different passive vehicles $p \in P^a$. This is due to the structure of the graphs $G^a = (V^a, E^a)$. Moreover, a passive vehicle $p \in P$ can be associated with different routes traveled by different active vehicles.

3.4 An Extended Set-Partitioning Formulation

In order to solve the APVRP with branch-and-price-and-cut, we use an extended set-partitioning formulation. This extensive formulation can be derived from a compact APVRP model, e.g., the one by Meisel and Kopfer (2014), using a Dantzig-Wolfe reformulation for integer programs (see Desaulniers *et al.*, 1998; Lübbecke and Desrosiers, 2005). For the sake of brevity, though, we omit the formal derivation.

For each class $a \in A$ of active vehicles, let Ω^a be the set of all feasibly scheduled routes. In contrast to many other column-generation algorithms for vehicle routing described in the literature, a column in the APVRP does not only represent a path, i.e., the sequence of visited vertices. A column in the APVRP provides a path (in the graph $G^a = (V^a, E^a)$) and a feasible schedule for this path. For simplicity, however, we will refer to *routes* in the following, but use the terms *path* and *schedule* to describe the routing and the scheduling components. The following attributes characterize a route q:

X_{ij}^q	number of times arc (i, j) is traversed by route q;
T_i^q	service start time at a vertex $i \in N^R \cup \{d\}$;
b_i^q	number of times vertex i is visited on route q;
c^q	total costs of route q.

The route costs are defined as by Meisel and Kopfer (2014) as $c^q = \alpha \sum_{(i,j) \in E^a} c_{ij} X_{ij}^q + \beta T_d^q$, i.e., they are a weighted sum of the length of the route and the arrival time at the

destination. Note that service start times are well-defined because a feasible route is elementary.

The formulation uses the following types of variables: continuous variables λ^{aq} measuring the flow of active vehicles of class $a \in A$ along route $q \in \Omega^a$, binary variables x^a_{ij} indicating whether or not arc $(i,j) \in E^a$ is traversed by an active vehicle of class $a \in A$, and binary variables u_r indicating whether or not request $r \in R$ remains unfulfilled. Recall that the penalty for not performing a request is γ. Now, the extended set-partitioning formulation for the APVRP is as follows:

$$\min \quad \sum_{a\in A}\sum_{q\in\Omega^a} c^q\lambda^{aq} + \gamma\sum_{r\in R} u_r \tag{3.1a}$$

$$\text{s.t.} \quad \sum_{a\in A}\sum_{q\in\Omega^a}\sum_{p\in P^r\cap P^a} b^q_{v^-_{rp}}\lambda^{aq} + u_r = 1 \qquad r\in R \tag{3.1b}$$

$$\sum_{a\in A^p}\sum_{q\in\Omega^a}\left(b^q_{v^-_{rp}} - b^q_{w^+_{rp}}\right)\lambda^{aq} = 0 \qquad r\in R,\ p\in P^r \tag{3.1c}$$

$$\sum_{a\in A^p}\sum_{q\in\Omega^a}\left(b^q_{w^-_{rp}} - b^q_{v^+_{rp}}\right)\lambda^{aq} = 0 \qquad r\in R,\ p\in P^r \tag{3.1d}$$

$$\sum_{a\in A}\sum_{q\in\Omega^a}\sum_{p\in P^r\cap P^a}\left(T^q_{w^+_{rp}} - T^q_{v^-_{rp}}\right)\lambda^{aq} + s^+_r u_r \geq s^+_r \qquad r\in R \tag{3.1e}$$

$$\sum_{a\in A}\sum_{q\in\Omega^a}\sum_{p\in P^r\cap P^a}\left(T^q_{v^+_{rp}} - T^q_{w^-_{rp}}\right)\lambda^{aq} + s^-_r u_r \geq s^-_r \qquad r\in R \tag{3.1f}$$

$$\sum_{a\in A^p}\sum_{q\in\Omega^a} b^q_{o_p}\lambda^{aq} = 1 \qquad p\in P \tag{3.1g}$$

$$\sum_{q\in\Omega^a}\lambda^{aq} \leq K_a \qquad a\in A \tag{3.1h}$$

$$x^a_{ij} = \sum_{q\in\Omega^a} X^q_{ij}\lambda^{aq} \qquad a\in A,(i,j)\in E^a \tag{3.1i}$$

$$\lambda^{aq} \geq 0 \qquad a\in A, q\in\Omega^a \tag{3.1j}$$

$$x^a_{ij} \in \{0,1\} \qquad a\in A,(i,j)\in E^a \tag{3.1k}$$

$$u_r \in \{0,1\} \qquad r\in R \tag{3.1l}$$

(3.1a) is the objective function. (3.1b) are the set partitioning constraints, which ensure that each request is either performed exactly once or the penalty γ is paid for leaving the request unfulfilled. (3.1c) and (3.1d) preserve consistency with respect to the passive vehicle used to satisfy a request in its different stages. (3.1e) and (3.1f) are time synchronization constraints ensuring that the different vertices corresponding to a request are visited at correct points in time, taking into account their precedence relationships. (3.1g) and (3.1h) are quantity constraints for passive vehicles and for classes of active vehicles respectively. Constraints (3.1g) are valid only if it is not allowed to temporarily

park a passive vehicle at its initial or final location. (3.1i) link the route and the arc variables. (3.1j)–(3.1l) indicate the domains of the variables.

Note that there are no binary restrictions on the λ^{aq} variables. The reason for this is that a solution may be fractional in terms of λ^{aq} variables for a class of active vehicles. The λ^{aq} variables may correspond to the same path with different schedules, i.e., to the same sequence of vertices visited at different points in time, so that the arc variables are integral and no branching is necessary. This issue is discussed in detail by Desaulniers *et al.* (1998) and Jans (2010).

In the following, the linear relaxation of formulation (3.1) is denoted as the *master program*. It does not contain the coupling constraints (3.1i) and the integer constraints (3.1k)–(3.1l). For solving the master program, a column generation algorithm (Desaulniers *et al.*, 2005) is employed. Branching is required to finally ensure integer solutions of formulation (3.1).

3.5 A Branch-and-Price-and-Cut Algorithm

In this section, we present the branch-price-and-cut algorithm we devised for solving the APVRP. First, in Subsection 3.5.1, we give a MIP model of the pricing problems for generating new columns. In Subsection 3.5.2, we describe a labeling algorithm for actually solving these pricing problems. Then, in Subsection 3.5.3, we describe our branching strategy. Finally, valid cutting planes for the master problem are described in Subsection 3.5.4.

3.5.1 Pricing Problems

As is common for column-generation algorithms for vehicle-routing problems, the pricing problems for the APVRP are shortest-path problems with resource constraints (SPPRCs) on graphs with negative cost cycles (Irnich and Desaulniers, 2005). For each class $a \in A$ of active vehicles, there is one such pricing problem. Its goal is to find at least one route with negative reduced costs or to prove that no such route exists. The dual variables of those constraints in which the λ^{aq} variables occur in the restricted master program, the resulting linear vertex costs, and the resulting reduced costs of the arcs can be read from Table 3.2.

We use binary variables x_{ij} indicating whether or not arc $(i,j) \in E^a$ is traversed and continuous T_i variables indicating the point in time when the service at vertex $i \in V^a$ begins. The symbols $\delta^+(i)$ and $\delta^-(i)$ denote the forward and backward star of vertex i respectively. The pricing problem for a class a of active vehicles can be specified as:

$$\min \quad \sum_{(i,j)\in E^a} \tilde{c}_{ij} x_{ij} + \sum_{i \in N^a} \tilde{c}_i T_i + \beta T_d \tag{3.2a}$$

$$\text{s.t.} \quad \sum_{(o,j)\in\delta^+(o)} x_{oj} = 1 \tag{3.2b}$$

$$\sum_{(j,i)\in\delta^-(i)} x_{ji} - \sum_{(i,j)\in\delta^+(i)} x_{ij} = 0 \qquad i \in V^a \setminus \{o,d\} \tag{3.2c}$$

$$\sum_{(i,d)\in\delta^-(d)} x_{id} = 1 \tag{3.2d}$$

$$T_i + t_{ij} \leq T_j + t^{max}(1 - x_{ij}) \qquad (i,j) \in E^a \tag{3.2e}$$

$$e_i \leq T_i \leq l_i \qquad i \in \{o,d\} \tag{3.2f}$$

$$e_i \sum_{(i,j)\in\delta^+(i)} x_{ij} \leq T_i \leq l_i \sum_{(i,j)\in\delta^+(i)} x_{ij} \qquad i \in V^a \setminus \{o,d\} \tag{3.2g}$$

$$x_{ij} \in \{0,1\} \qquad (i,j) \in E^a \tag{3.2h}$$

(3.2a) is the objective of minimizing the reduced costs of the route. (3.2b)–(3.2d) are path-flow constraints and (3.2e)–(3.2g) ensure time-window feasibility. All arc variables are binary, as imposed by (3.2h).

Constraint	Dual variable	Range
(3.1b)	π_r	$\mathbb{R}$
(3.1c)	ϕ^+_{rp}	$\mathbb{R}$
(3.1d)	ϕ^-_{rp}	$\mathbb{R}$
(3.1e)	τ^+_r	≥ 0
(3.1f)	τ^-_r	≥ 0
(3.1g)	μ^P_p	$\mathbb{R}$
(3.1h)	μ^A_a	≤ 0

(a)

Vertex i	Linear vertex costs $\tilde{c}_i$
v^-_{rp}	τ^+_r
w^+_{rp}	$-\tau^+_r$
w^-_{rp}	τ^-_r
v^+_{rp}	$-\tau^-_r$
d	β
otherwise	0

(b)

Arc (i,j)	Reduced costs $\tilde{c}_{ij}$
(i,o_p)	$\alpha c_{io_p} - \mu^P_p$
(i,d_p)	αc_{id_p}
(i,v^-_{rp})	$\alpha c_{iv^-_{rp}} - \pi_r - \phi^+_{rp}$
(i,w^+_{rp})	$\alpha c_{iw^+_{rp}} + \phi^+_{rp}$
(i,w^-_{rp})	$\alpha c_{iw^-_{rp}} - \phi^-_{rp}$
(i,v^+_{rp})	$\alpha c_{iv^+_{rp}} + \phi^-_{rp}$
(i,d)	$\alpha c_{id} - \mu^A_a$

(c)

Table 3.2: (a) Dual Variables and their Ranges; (b) Linear Vertex Costs; (c) Reduced Costs of Arcs

3.5.2 Dynamic Programming Labeling Algorithms

The usual solution approach for SPPRC pricing problems is a dynamic-programming based labeling algorithm. As has been shown by Salani (2005), solving such pricing problems by bidirectional dynamic programming, i.e., by propagating labels forward from the start depot vertex and backward from the end depot vertex, allows considerable speedups compared to a unidirectional procedure. We therefore adopt this approach and adapt it to the APVRP pricing problem.

In what follows, we first point out the specific characteristics resulting from the linear vertex costs in the APVRP pricing problem and briefly review similar problems and solution approaches encountered in the literature. We then discuss issues related to (non-)elementarity of SPPRC solutions and present our adaptation of the ng-path relaxation. Afterwards, we describe the forward label extension step and the dominance

procedure we apply. Then, we elaborate on the backward label extension and the method for merging forward and backward labels, and, finally, we present some techniques for accelerating the pricing process.

Labeling Algorithms for SPPRCs with Linear Vertex Costs

Due to the linear vertex costs $\tilde{c}_i$ in model (3.2), which are not present in pricing problems for standard VRPs, there is a tradeoff in the APVRP pricing problem: In a path in which the linear vertex costs are non-negative at all vertices, it is best to visit all vertices as early as possible. Similarly, in a path in which the linear vertex costs are non-positive at all vertices, it is best to visit all vertices as late as possible. However, paths in the APVRP may visit vertices with positive as well as vertices with negative vertex costs. Hence, determining a cost-optimal schedule even for a *given* path is a non-trivial optimization problem in itself. For a labeling algorithm, this means that the linear vertex costs and the resulting tradeoff between costs and time create an infinite number of possible states that do not dominate one another. Such a situation was first studied by Ioachim *et al.* (1998) on an acyclic time-space network in the context of aircraft fleet routing and scheduling. Their approach of storing cost functions (with time as the independent variable) as labels and propagating these functions instead of scalar values forms the basis of our method, which is described in detail in Section 3.5.2.

Other authors that consider shortest-path pricing problems with linear vertex costs in column-generation algorithms for routing problems are Christiansen and Nygreen (1998) (for a ship routing and scheduling problem with inventory constraints), Dohn *et al.* (2009, 2011) (for VRPs with precedences and temporal dependencies between visits to customers), Liberatore *et al.* (2011) (for a VRP with soft time windows) and Spliet and Gabor (2014) (for a VRP with time windows that have to be assigned before customer demand is known). The pricing problem of Christiansen and Nygreen (1998) is solved with the Ioachim *et al.* (1998) algorithm. Dohn *et al.* (2009, 2011) describe two alternative approaches for dealing with the linear vertex costs: They consider a so-called time-indexed formulation with an implicit representation of precedence constraints and the possibility of branching on time windows to handle temporal dependencies. Liberatore *et al.* (2011) and Spliet and Gabor (2014) also base their procedures on the Ioachim *et al.* (1998) approach.

The main differences between the problems and solution procedures presented by Ioachim *et al.* (1998), Liberatore *et al.* (2011), Spliet and Gabor (2014), and ours are as follows: Ioachim *et al.* (1998) consider an acyclic network, all others have networks with cycles. Due to the soft time windows, the Liberatore *et al.* (2011) problem is, on the one hand, simpler with respect to time windows, as the complete planning horizon is feasible. On the other hand, it is more difficult with respect to the cost (penalty) function *at each single vertex*: This penalty function is decreasing until the beginning of the desired time window, then constant until the end of the desired time window, and then increasing. By contrast, the cost functions in the papers by Ioachim *et al.* (1998), Spliet and Gabor (2014) as well as in our paper are either constant or only increasing or only decreasing at each vertex.

Both the Liberatore *et al.* (2011) and the Spliet and Gabor (2014) pricing problem feature a capacity constraint, which is not present in the Ioachim *et al.* (1998) problem or in the APVRP. Liberatore *et al.* (2011) handle the capacity constraint in the dominance procedure by allowing a label L to dominate another label L' only if the load of L is less than or equal to that of L'. Spliet and Gabor (2014) construct an auxiliary acyclic graph in which the capacity constraint is taken into account implicitly at the cost of a weaker dominance.

The branch-and-price algorithm by Smilowitz (2006) mentioned in the literature review section, although considering an APVRP use case, does not take into account the time synchronization aspect in an exact manner, contrary to our approach. Instead, the interdependencies between tasks are modeled away by assuming that a passive vehicle which is used to perform a task τ is available for performing other tasks at the earliest after the end of τ's time window plus the service time for τ. In this way, the underlying network does not contain arcs between tasks with overlapping time windows, no linear vertex costs occur, and standard labeling approaches can be used for solving the pricing problem.

Elementarity and Precedences

In an optimal solution to the set-partitioning formulation (3.1), no task will be performed more than once. This implies that all columns in an optimal solution correspond to elementary paths in the network. It is well known that the elementary shortest-path problem with resource constraints is NP-hard in the strong sense (Dror, 1994). Hence, in column-generation algorithms for VRPs, many authors solve as pricing problems non-elementary SPPRCs (see the survey by Desaulniers *et al.*, 2014). This can be done in pseudo-polynomial time (Irnich and Desaulniers, 2005). Although this yields weaker lower bounds, and although routes with cycles must be removed in the branching process, solving only a relaxed pricing problem often pays off with respect to overall computation time. One such approach that has been very successfully used for different types of VRPs is the ng-path relaxation introduced by Baldacci *et al.* (2011).

We adapt this approach to the APVRP as follows. Our ng-path relaxation is based on tasks instead of vertices, thus leading to a stronger relaxation on the network we use. For each vertex $i \in N^a \cup \{o_p, d_p : p \in P^a\}$, we use an ng-neighborhood $\mathcal{N}_i$ containing the task associated to i (see Table 3.3) and the ν closest tasks associated to vertices j for which a cycle $(j, \ldots, i, \ldots, j)$ would be feasible with respect to time windows and travel and service times. (We use different values of the parameter ν in our computational experiments.) In an ng-path in the APVRP, it is possible that a task τ is performed more than once if, between two visits to a vertex i associated with the task, at least one other vertex j is visited such that $\tau \notin \mathcal{N}_j$.

In addition, to foster elementarity of partial paths without weakening dominance, we use the following approach: Consider the second line of Table 3.3. As described in Section 3.3, we associate three tasks with each request, and two tasks with each passive vehicle, for pickup and delivery at the initial and final location. Obviously, these tasks must be performed in the chronological sequences τ_r^1, τ_r^2, τ_r^3 and τ_p^o, τ_p^d (but not

necessarily by the same active vehicle). Now, at each request vertex i, we consider two subsets of these tasks, $\mathcal{T}_i^{\text{test}}$ and $\mathcal{T}_i^{\text{set}}$, as indicated in the third and fourth line in the table. Before extending a label L at vertex i to vertex j, we test the tasks in the associated set $\mathcal{T}_j^{\text{test}}$. If any of these tasks has already been fulfilled along the partial path represented by L, then L is not extended to j, because such a path is not feasible for the APVRP. Similarly, upon extending a label L at vertex i to vertex j, we mark the tasks in $\mathcal{T}_j^{\text{set}}$ as fulfilled, because visiting them would lead to an infeasible path regarding task precedences. Note that it is unnecessary to include the tasks in the sets $\mathcal{T}^{\text{test}}$ and $\mathcal{T}^{\text{set}}$ in the ng-neighborhood. This is because these tasks are irrelevant for dominance, as dominance checks occur only between labels resident at the same vertex, and as all tasks to test and all tasks to set are the same for all labels at the same vertex. With respect to the ng-path relaxation, on the one hand, the tasks to set allow making labels more comparable from the dominance point of view, and, on the other hand, the tasks to test prevent the construction of infeasible paths. This improvement can always be applied whenever a subset of tasks associated with some vertices of the graph must be executed according to a given precedence relation.

Vertex i	o_p	v_{rp}^-	w_{rp}^+	w_{rp}^-	v_{rp}^+	d_p
Associated task	τ_p^o	τ_r^1	τ_r^2	τ_r^2	τ_r^3	τ_p^d
Tasks to test $\mathcal{T}_i^{\text{test}}$	τ_p^o, τ_p^d	$\tau_r^1, \tau_r^2, \tau_r^3, \tau_p^d$	$\tau_r^2, \tau_r^3, \tau_p^d$	τ_r^3, τ_p^d	τ_r^3, τ_p^d	τ_p^d
Tasks to set $\mathcal{T}_i^{\text{set}}$	τ_p^o	τ_p^o, τ_r^1	τ_p^o, τ_r^1	$\tau_p^o, \tau_r^1, \tau_r^2$	$\tau_p^o, \tau_r^1, \tau_r^2, \tau_r^3$	τ_p^o, τ_p^d

Table 3.3: Chain of Precedences for each Pair $(p, r) \in P \times R^p$

Forward Label Extension

As mentioned, our approach is based on the one by Ioachim *et al.* (1998), who store cost functions as labels instead of scalars. Such a cost function c provides the minimal costs $c(T)$ incurred by a path when the service at the last vertex of the path starts at time T. Ioachim *et al.* (1998) prove that this function is piecewise linear, convex, and contains at most as many linear pieces as there are vertices in the path. Moreover, they show that pieces with positive slope can be replaced by a single piece with slope zero. Hence, for the APVRP, a label comprises the following components:

n	number of time-slope pairs (in the following called pieces);
$(t^p, s^p)_{p=1}^n$	the n pieces; $s^p < 0$ for $p = 1, \ldots, n-1$, and $s^n \leq 0$
c^1	(reduced) costs at start time of piece 1;
t^{n+1}	end time of last piece n;
S	tasks that have already been performed or are unreachable along the route and which are taken into account for checking elementarity in the label extension step according to the ng-path relaxation;
k	number of intervals on which the label is dominated;
$(I^d)_{d=1}^k$	the k intervals on which the label is dominated.

The following information can then be derived:

c^* $= c^1 + \sum_{p=1}^{n} s^p(t^{p+1} - t^p)$; the optimal (reduced) costs;

t^* $= t^n$ for $s^n = 0$, and t^{n+1} otherwise; the earliest time to obtain costs c^*;

$c(T)$ $= c^1 + \sum_{p=1}^{q-1} s^p(t^{p+1} - t^p) + s^q(T - t^q)$ for $t^q \leq T \leq t^{q+1}$; the tradeoff curve;

s^{n+1} $= 0$ defined so for convenience.

Figure 3.2 depicts two typical situations. In Figure 3.2(a), the slope of the function is strictly negative over its complete range, so that the minimal (reduced) costs are achieved when starting the service at the vertex as late as possible. In Figure 3.2(b), the slope of the function is zero on the positive-length interval from t^2 to t^3, and any time value in this interval yields the minimal (reduced) costs c^*. Note that the figures imply that the paths whose cost functions are depicted in 3.2(a) and (b) contain at least three and two vertices respectively.

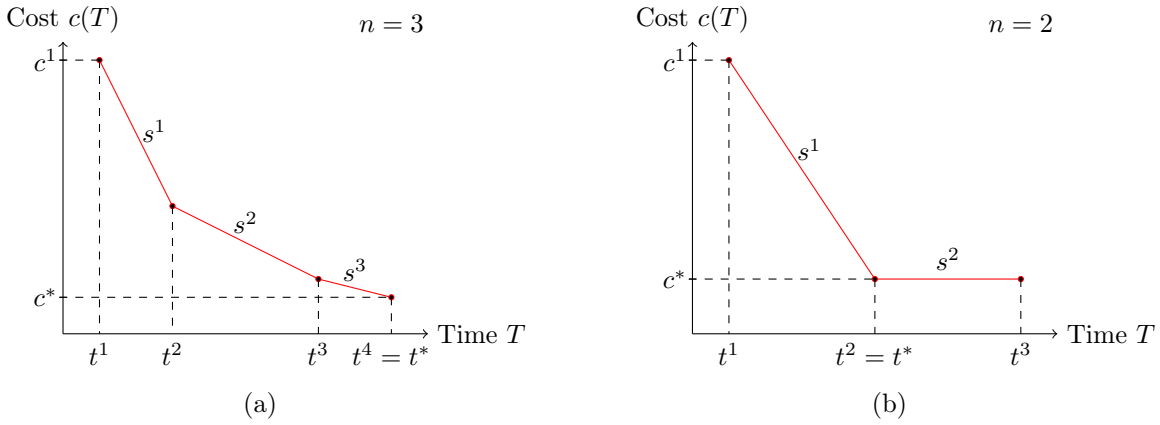

Figure 3.2: Examples of Labels; (a) A label with $n = 3$ linear pieces and final slope $s^n < 0$; (b) A label with $n = 2$ linear pieces and final slope $s^n = 0$.

The initial label at vertex o has a single piece $(t_o^1, s_o^1) = (e_o, 0)$, initial reduced costs $c_o^1 = 0$, task set $S_o = \varnothing$, and it is undominated, i.e., $k_o = 0$. Note that by initializing the first piece as $(t_o^1, s_o^1) = (e_o, -\beta)$ would allow for changing the time-related part of the objective to route duration minimization, see also (Tilk and Irnich, 2016).

Now we describe the label extension step. (We present the extension step here because the paper by Ioachim *et al.* (1998) omits several details.) Let $L_i = ((t_i^p, s_i^p)_{p=1}^{n_i+1}, c_i^1, S_i, (I_i^d)_{d=1}^{k_i})$ be a label at vertex i. The extension of L_i along the arc (i, j) is feasible if $t_i^1 + t_{ij} \leq l_j$ and $S_i \cap \mathcal{T}_j^{\text{test}} = \varnothing$. In this case, a new label L_j at vertex j is created. The attributes of L_j are computed in the following way:

First, extending some of the existing pieces may be obsolete (cf. Ioachim *et al.*, 1998, p. 200), either because the extended pieces arrive too early or too late at j or because several resulting new slopes are zero. We compute the indices f and g of the first and

last new piece to be kept:

$$f := \max\left\{1;\ \max\{p \in \{1,\dots,n_i+1\} : t_i^p + t_{ij} \le e_j\}\right\}$$

and

$$g := \min\left\{n_i;\ \min\{p \in \{0,\dots,n_i\} : s_i^p + \tilde{c}_j \ge 0 \text{ or } t_i^{p+1} + t_{ij} \ge l_j\}\right\}$$

(defining $s_i^0 = -\infty$ for the case $p = 0$). It may also happen that a new piece must be created (cf. Ioachim *et al.*, 1998, p. 200). There are three cases: (i) The last piece of L_i has negative slope, will be extended to j and when starting from i at the optimal time t_i^* vertex j can be reached in its time window, (ii) j is reached before the start of its time window for all $T \in [t_i^1, l_i]$, and (iii) only a single point in time is propagated to j and this point is a breakpoint of $c_i(T)$.

Thus, we define the new-piece indicator

$$\delta := \begin{cases} 1 & \text{if } (g = n_i, t_i^* = t_i^{n+1}, \text{ and } t_i^* + t_{ij} < l_j) \text{ or } (f = n_i + 1) \text{ or } (l_j = t_i^f + t_{ij}) \\ 0 & \text{otherwise} \end{cases}, \tag{3.3}$$

so that the new label L_j comprises the

$$n_j := \max\{0, g - f + 1\} + \delta \tag{3.4}$$

new pieces.

Second, non-obsolete pieces (t, s) are extended using the function $f_{ij}(t, s) := (\max\{e_j, t + t_{ij}\}, \min\{0, s + \tilde{c}_j\})$, so that the new pieces are

$$(t_j^p, s_j^p) := f_{ij}(t_i^{f+p-1}, s_i^{f+p-1}) \qquad \text{for all } p = 1, \dots, n_j, \tag{3.5a}$$

and the new end time of the pieces associated with slope 0 is

$$(t_j^{n_j+1}, s_j^{n_j+1}) := (l_j, 0). \tag{3.5b}$$

Third, with the help of the tradeoff curve $c_i(T)$ of L_i and the already computed attributes, the costs at the start time t_j^1 of the new pieces can be expressed as

$$c_j^1 := c_i(\min\{t_i^*, t_j^1 - t_{ij}\}) + \tilde{c}_{ij} + \tilde{c}_j t_j^1. \tag{3.5c}$$

Note that the formula for the cost update given by Ioachim *et al.* (1998, p. 202) is not always correct.

Fourth, the remaining attributes are

$$S_j := (S_i \cap \mathcal{N}_j) \cup \mathcal{T}_j^{\text{set}} \qquad \text{and} \qquad k_j := 0. \tag{3.5d}$$

The latter definition means that at the time of its creation, the new label is undominated. This completes the description of the new label $L_j = ((t_j^p, s_j^p)_{p=1}^{n_j+1}, c_j^1, S_j, (I_j^d)_{d=1}^{k_j})$.

From a label L_d at the end depot d, the corresponding o-d-path is reconstructed, as usual, by iterating backward through the predecessor labels. Let $(d = i+1, i, \dots, 0 = o)$

be the backtracked path from a label L_d. The optimal schedule $(T_{i+1}, \dots, T_0)$ is then given by the following recursion:

$$T_{i+1} = t_d^*, \tag{3.6a}$$

$$T_i = \min\{t_i^*, T_{i+1} - t_{i,i+1}\}. \tag{3.6b}$$

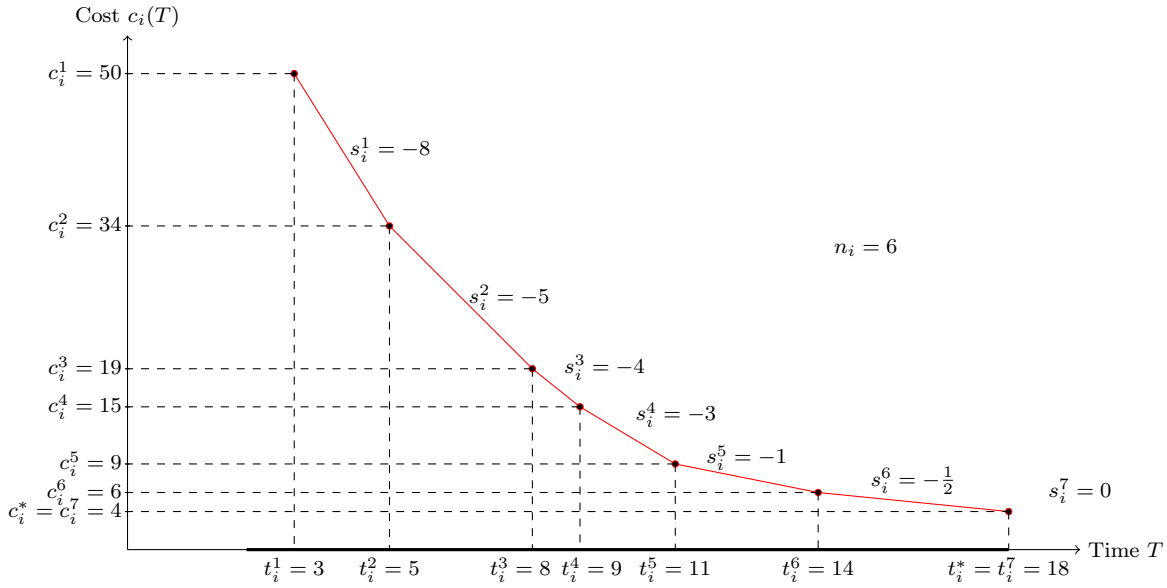

Figure 3.3: Detailed Numerical Example of Extension Step

Example 3.1. *We give a concrete numerical example for the extension step: A label L_i at vertex i has $n_i = 6$ pieces defined as in Figure 3.3. The time window at vertex i is $[e_i, l_i] = [2, 18]$, while the domain of $c_i(T)$ is $[3, 18]$. As the last piece has negative slope $s_i^6 = -\frac{1}{2}$, we have $t_i^* = l_i = 18$ with minimum cost $c_i^* = 4$. We extend L_i along the arc (i, j) having reduced cost $\tilde{c}_{ij} = 15$ and travel time $t_{ij} = 3$. The linear vertex cost of the head vertex j is assumed to be $\tilde{c}_j = -1$.*

We will show how the new tradeoff curve $c_j(T)$ at vertex j is computed. For the sake of explanation, we vary the time window $[e_j, l_j]$ of vertex j. The following table shows, depending on e_j and l_j (assuming $e_j \le l_j$), the resulting auxiliary values (f, g, δ) and the number n_j of pieces of the new tradeoff curve $c_j(T)$:

$e_j \le l_j$ *requ.*	$l_j = 6$	$6 < l_j \le 8$	$8 < l_j \le 11$	$11 < l_j \le 12$	$12 < l_j \le 14$	$14 < l_j \le 17$	$17 < l_j \le 21$	$l_j > 21$
$e_j < 8$	$(1,0,1)$ $n_j = 1$	$(1,1,0)$ $n_j = 1$	$(1,2,0)$ $n_j = 2$	$(1,3,0)$ $n_j = 3$	$(1,4,0)$ $n_j = 4$	$(1,5,0)$ $n_j = 5$	$(1,6,0)$ $n_j = 6$	$(1,6,1)$ $n_j = 7$
$8 \le e_j < 11$		$(2,1,1)$ $n_j = 1$	$(2,2,0)$ $n_j = 1$	$(2,3,0)$ $n_j = 2$	$(2,4,0)$ $n_j = 3$	$(2,5,0)$ $n_j = 4$	$(2,6,0)$ $n_j = 5$	$(2,6,1)$ $n_j = 6$
$11 \le e_j < 12$			$(3,2,1)$ $n_j = 1$	$(3,3,0)$ $n_j = 1$	$(3,4,0)$ $n_j = 2$	$(3,5,0)$ $n_j = 3$	$(3,6,0)$ $n_j = 4$	$(3,6,1)$ $n_j = 5$
$12 \le e_j < 14$				$(4,3,1)$ $n_j = 1$	$(4,4,0)$ $n_j = 1$	$(4,5,0)$ $n_j = 2$	$(4,6,0)$ $n_j = 3$	$(4,6,1)$ $n_j = 4$
$14 \le e_j < 17$					$(5,4,1)$ $n_j = 1$	$(5,5,0)$ $n_j = 1$	$(5,6,0)$ $n_j = 2$	$(5,6,1)$ $n_j = 3$
$17 \le e_j < 21$						$(6,5,1)$ $n_j = 1$	$(6,6,0)$ $n_j = 1$	$(6,6,1)$ $n_j = 2$
$e_j \ge 21$							$(7,6,1)$ $n_j = 1$	$(7,6,1)$ $n_j = 1$

In a first example, we choose $[e_j, l_j]$ *so wide that all six pieces of the original curve* $c_i(T)$ *are transferred into the inner part of the time window, and an additional seventh piece must be created. This happens, for example if* $[e_j, l_j] = [0, 25]$. *The corresponding values are* $(f, g, \delta) = (1, 6, 1)$ *giving us* $n_j = 7$. *Using Eq. (3.5a) with* $f_{ij}(t, s) = (\max\{e_j, t + t_{ij}\}, \min\{0, s - \tilde{c}_j\}) = (\max\{0, t + 3\}, \min\{0, s - 1\})$, *we get* $(t_j^1, s_j^1) = f_{ij}(t_i^1, s_i^1) = f_{ij}(3, -8) = (6, -9)$, $(t_j^2, s_j^2) = f_{ij}(t_i^2, s_i^2) = f_{ij}(5, -5) = (8, -6)$, $(t_j^3, s_j^3) = f_{ij}(t_i^3, s_i^3) = f_{ij}(8, -4) = (11, -5)$, $(t_j^4, s_j^4) = f_{ij}(t_i^4, s_i^4) = f_{ij}(9, -3) = (12, -4)$, $(t_j^5, s_j^5) = f_{ij}(t_i^5, s_i^5) = f_{ij}(11, -1) = (14, -2)$, *and* $(t_j^6, s_j^6) = f_{ij}(t_i^6, s_i^6) = f_{ij}(14, -\frac{1}{2}) = (17, -1.5)$. *The seventh piece results from the zero slope piece* $(t_i^7, s_i^7) = (18, 0)$ *at vertex* i *and is* $(t_j^7, s_j^7) = f_{ij}(18, 0) = (21, -1)$. *The reduced cost at the start time* $t_j^1 = 6$ *is then computed with the help of Eq. (3.5c). We obtain* $c_i(\min\{t_i^*, t_j^1 - t_{ij}\}) = c_i(\min\{18, 6 - 3\}) = c_i(3) = 50$ *resulting in* $c_j^1 = 50 + \tilde{c}_{ij} + \tilde{c}_j t_j^1 = 50 + 15 + (-1)6 = 59$.

As a second set of examples, we show the extreme case that $c_j(T)$ *has a single piece only, i.e.,* $n_j = 1$. *This results if the time window* $[e_j, l_j]$ *is so small that only one of the pieces of* $c_i(T)$ *is relevant and no new piece is created as a last piece. This is true for* $f = g$ *and* $\delta = 0$ *(values above the diagonal in the table). For example, choosing* $[e_j, l_j] = [13, 14]$ *gives* $(f, g, \delta) = (4, 4, 0)$. *With* $f_{ij}(t, s) = (\max\{e_j, t + t_{ij}\}, \min\{0, s + \tilde{c}_j\}) = (\max\{13, t + 3\}, \min\{0, s - 1\})$, *we get* $(t_j^1, s_j^1) = f_{ij}(t_i^4, s_i^4) = f_{ij}(9, -3) = (13, -4)$. *A single piece can also result if the domain of* $c_j(T)$ *is a single point in time* ($t_j^1 = l_j$) *and a breakpoint of* $c_i(T)$ *is mapped into this degenerated domain. The latter condition is true if* $l_j = t_i^f + t_{ij}$, *which imposes* $\delta = 1$ *due to the last term in Eq. (3.3). For example, the time window* $[e_j, l_j] = [11, 11]$ *gives* $(f, g, \delta) = (3, 2, 1)$ *and thus* $n_j = 1$ *(values on diagonal of above table). For the new extra piece, we get* $(t_j^1, s_j^1) = f_{ij}(t_i^3, s_i^3) = f_{ij}(8, -4) = (11, -5)$ *with* $f_{ij}(t, s) = (\max\{e_j, t + t_{ij}\}, \min\{0, s + \tilde{c}_j\}) = (\max\{11, t + 3\}, \min\{0, s - 1\})$ *according to Eq. (3.5a).*

Finally, we give a last example of a resulting curve with $n_j = 3$ *pieces. We choose* $[e_j, l_j] = [9, 14]$ *resulting in* $(f, g, \delta) = (2, 4, 0)$. *Again, the three resulting pieces of* $c_j(T)$ *are computed using Eq. (3.5a) with* $f_{ij}(t, s) = (\max\{9, t + 3\}, \min\{0, s - 1\})$. *We get* $(t_j^1, s_j^1) = f_{ij}(t_i^2, s_i^2) = f_{ij}(5, -5) = (9, -6)$, $(t_j^2, s_j^2) = f_{ij}(t_i^3, s_i^3) = f_{ij}(8, -4) = (11, -5)$, *and* $(t_j^3, s_j^3) = f_{ij}(t_i^4, s_i^4) = f_{ij}(9, -3) = (12, -4)$. *The initial cost is* $c_j^1 = 29 + 15 + (-1)9 = 35$, *where* $29 = c_1(\min\{t_i^*, t_j^1 - t_{ij}\}) = c_1(\min\{18, 9 - 3\}) = c_1(6)$.

We now proceed to the discussion of the dominance procedure and details of its implementation.

Dominance between Labels

When checking a label L for dominance, it is compared one by one with all other undominated labels residing at the same vertex. During each such comparison between L and another label L', first, the sets S_L and $S_{L'}$ are compared. If $S_{L'} \subseteq S_L$, a pointwise dominance is performed with respect to each point on the tradeoff curve of L, and the intervals on which L is dominated, i.e., where $c_{L'}(T) \leq c_L(T)$ holds, are tentatively stored. We compute these domination intervals efficiently by determining the intersection points of $c_L(T)$ and $c_{L'}(T)$. Ioachim *et al.* (1998) have shown that even if the

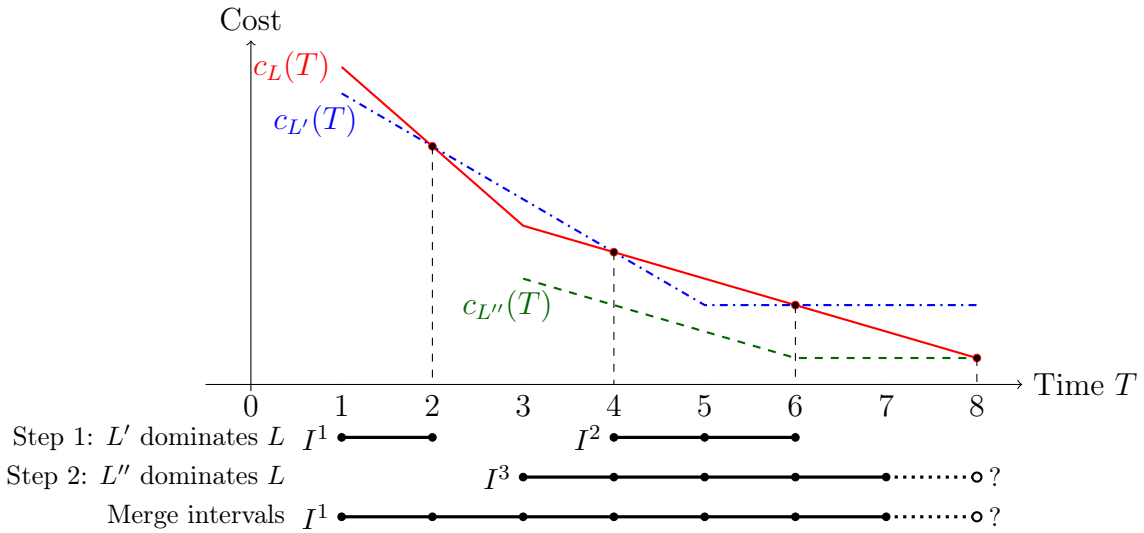

Figure 3.4: Detailed Example of Dominance

intersection points may be non-integer real numbers, it suffices to describe the intervals with integer bounds. Also Liberatore *et al.* (2011) exploit domination intervals, however, they delete the dominated intervals from the tradeoff curves so that they can become discontinuous.

After having compared L with all other labels in this way, these intervals are used to update $(I^d)_{d=1}^k$, the intervals on which the tradeoff curve of L is dominated, and k, the number of intervals on which L is dominated. If the complete tradeoff curve of a label is dominated, the label itself is dominated and can be discarded. Note that this means that the decision on whether or not a label is obsolete will regularly be based on comparisons with more than one other label. Put differently, in most cases, only several other labels together will make one label obsolete. This is in contrast to dominance procedures for classical VRPs, where a pairwise dominance is applicable and one label dominates another one or not.

To avoid that two labels L and L' dominate each other on an interval I^d when $S_L = S_{L'}$ and $c_L(T) = c_{L'}(T)$ for all $T \in I^d$, we use a tie-breaking rule, similar as in standard VRPs where it must be avoided that two identical labels eliminate each other.

Example 3.2. *We explain the dominance mechanism with the help of another concrete example depicted in Figure 3.4. We assume that a label L is given with tradeoff curve $c_L(T)$. Initially, L is not dominated, i.e., $k = 0$. In a first step, another label L' with tradeoff curve $c_{L'}(T)$ is generated at the same vertex. Assuming $S_{L'} \subsetneq S_L$, this second label dominates L on the intervals $[1, 2]$ and $[4, 6]$. This information is stored within L setting $k = 2$ and $(I^1, I^2) = ([1, 2], [4, 6])$.*

In the second step, a third label L'' with tradeoff curve $c_{L''}(T)$ is generated. Now we assume that $S_{L''} = S_L$ holds. Clearly, L'' dominates L on the half-open interval $[3, 8)$ because $c_{L''}(T) < c_L(T)$ for all $T \in [3, 8)$. Since $c_{L''}(8) = c_L(8)$, dominance at time

$T = 8$ depends on the tie breaking rule. Therefore, the dominance interval I^3 is either $[3, 7]$ or $[3, 8]$ and in Figure 3.4 the right end of the interval is dotted.

Merging the dominance intervals I^1, I^2, and I^3 leads to a single interval, even if neither $c_{L'}(T)$ nor $c_{L'}(T)$ is below $c_L(T)$ on the open interval $(2, 3)$. As stressed before, it suffices that dominance reflects the properties of schedules at integer points in time. As a result, the merging results in $k = 1$ and, depending on the tie breaking rule, the new dominance interval is $I^1 = [1, 7]$ or $I^1 = [1, 8]$. Only in the latter case, I^1 comprises the complete domain of $c_L(T)$ so that label L can be discarded.

Refinements of the dominance procedure that speed up the pricing process are the elimination of dominated pieces at the beginning or at the end of the range of the tradeoff curve and the replacement of consecutive dominated pieces by one aggregated piece.

Bidirectional Labeling

We start by briefly describing the backward label extension before we detail the bidirectional labeling approach.

Given the forward label extension process, the backward label extension process is rather simple: It is sufficient to invert all time windows and to invert the linear vertex costs. To be precise, a time window $[e_r, l_r]$ for a request r is replaced by $[t^{max} - l_r, t^{max} - e_r]$, $\tilde{c}$ is replaced by $-\tilde{c}$, and then the same algorithm as in the forward labeling is applied. Note that it suffices to compute $t^{max} - T$ to recalculate the real time for a given point in time T on the tradeoff curve of the backward label. For simplicity, when a point in time on the tradeoff curve of the backward label is mentioned in the following, we refer to the real time. Hence, the pieces are numbered from 1 to n with decreasing, non-negative slopes, and t^1 is the latest feasible time for a backward label. Figure 3.5 summarizes the notation for backward labels.

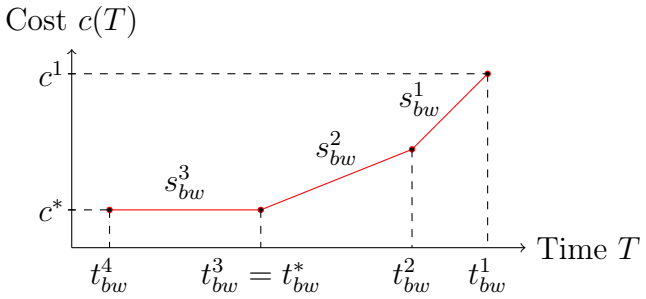

Figure 3.5: Example of a Backward Label with $n = 3$ Pieces

In bidirectional labeling algorithms, forward labels are not necessarily propagated until the end depot, and backward labels are not necessarily propagated until the start depot. Instead, labels are propagated only up to a so-called half-way point, thus limiting the overall number of created labels. Suitable forward and backward labels must then be merged to obtain complete *o*-*d*-paths. As described by Salani (2005, Sect. 4.6.4),

this is done using a half-way point test to avoid creating the same path from different pairs of forward and backward labels. Setting $t^{max}/2$ as half-way point, we propagate forward labels at a vertex i only if $t_i^1 \leq t^{max}/2$, and backward labels at a vertex j only if $t_j^1 > t^{max}/2$.

We then merge on vertices, i.e., we consider forward and backward labels at the same vertex i. A forward label at a vertex i qualifies for merging if $i = d$ or its earliest service start time is $t_{fw}^1 > t^{max}/2$. This condition prevents the creation of identical paths from different pairs of forward and backward labels. We check all backward labels at vertex i for whether or not they can be merged with the chosen forward label. The procedure becomes simpler if instead of the backward label itself, its predecessor label is considered. Let this predecessor be resident at vertex j. Note that this is a kind of merging along the resulting arc (i, j). However, with the merge on vertices we benefit from the fact that both the forward and the backward label are Pareto-optimal. This is in contrast to the situation when merging is directly performed over arcs, where neither the forward extension of the forward label along (i, j) nor the backward extension of the backward label along (i, j) is guaranteed to be Pareto-optimal, leading to considerably more pairs of labels to be checked.

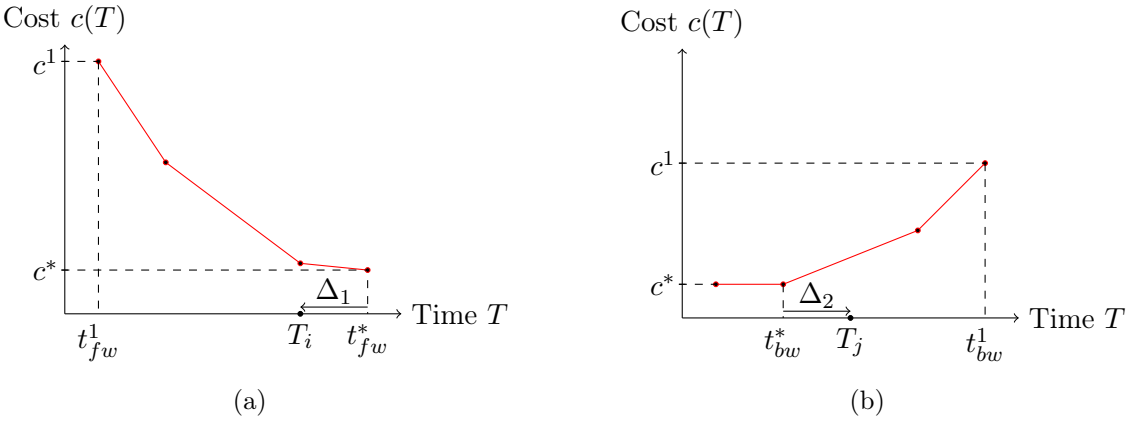

Figure 3.6: Example of a Merge Step, Forward and Backward Label with Optimal Times T_i and T_j for their Concatenation

We consider forward labels at a vertex i for a merge with predecessor backward labels at a vertex j if the following conditions are fulfilled to obtain a feasible path:

$$S_{fw} \cap S_{bw} = \varnothing \quad \text{and} \quad t_{fw}^1 + t_{ij} \leq t_{bw}^1 \tag{3.7}$$

The first condition is necessary to avoid paths violating the partial elementarity required by the ng-path relaxation, the second is needed to ensure overall temporal feasibility. Now assume that these conditions are fulfilled for a forward label at a vertex i and a backward label at a vertex j, and consider Figure 3.6. On the left hand side of Figure 3.6, an exemplary forward cost function at i is depicted. The predecessor backward label with an exemplary backward linear cost function at j is depicted on the right hand side. We have to compute the optimal service start times T_i and T_j at the vertices i and

j, respectively, so that all other service start times at the vertices in the path can be derived from these values, see Eqs. (3.6). There are two cases:

i) $t^*_{fw} + t_{ij} \leq t^*_{bw}$.
This is unproblematic: $T_i = t^*_{fw}$ and $T_j = t^*_{bw}$ are the optimal service start times for vertices i and j.

ii) $\Delta = t^*_{fw} + t_{ij} - t^*_{bw} > 0$.
In this case, T_i must be set to an earlier point in time than t^*_{fw}, and/or T_j must be set to a later point in time than t^*_{bw}. To ensure optimality, we compute the optimal times T_i and T_j by distributing the total required time shift Δ between the forward and the backward label. To distribute Δ, we initialize $T_i = t^*_{fw}$ and $T_j = t^*_{bw}$ and update them in the following manner. We consider the slopes of the cost function pieces. If, for example, $t^*_{fw} = t^{n+1}_{fw}$, $t^*_{bw} = t^n_{bw}$ and $|s^n_{fw}| \leq |s^{n-1}_{bw}|$, as it is the case in Figure 3.6, setting T_i to an earlier time than t^*_{fw} will increase the costs of the merged path by at most as much as pushing T_j to a later point in time than t^*_{bw}. Hence, we shift T_i backward by $\partial_1 := \min\{T_i - t^n_{fw}, \Delta\}$ units, i.e., $T_i := T_i - \partial_1$. The remaining time shift is $\Delta := \Delta - \partial_1$. Now, if $\Delta = 0$, we are done. Otherwise, we compare the slope of the preceding forward piece, here s^{n-1}_{fw}, with s^{n-1}_{bw} and select the piece with the smaller absolute slope. If, as depicted in the figure, $|s^{n-1}_{bw}| < |s^{n-1}_{fw}|$, we shift T_j forward by $\partial_2 := \min\{t^{n-1}_{bw} - T_j, \Delta\}$ units, i.e., $T_j := T_j + \partial_2$. Again, the remaining time shift is $\Delta := \Delta - \partial_2$. We iterate until $\Delta = 0$, and this is guaranteed to happen because of condition (3.7). The overall algorithm is depicted in Algorithm 3.1. Note that if the next forward piece to consider does not exist, i.e., if the merge point T_i is already equal to the start time t^1_{fw} of the forward tradeoff curve, we take the backward one, and vice versa. This can be achieved by defining $s^0_{fw} := -\infty$ and $s^0_{bw} := +\infty$.

In the merge procedure just described, the cost functions of backward labels are relevant only in the interval $[t^{max}/2, t^{max}]$ from the half-way point to the end of the planning horizon. Therefore, it is sufficient to construct the tradeoff curve of a backward label only up to $t^{max}/2$. This saves some computation time when constructing the linear cost function while maintaining optimality of the procedure.

The reduced cost of a merged path can be obtained by $c^*_m := c_{fw}(T_i) + c_{bw}(T_j) + \tilde{c}_{ij}$. After the merge, we perform a final dominance test as described in Subsection 3.5.2 for all routes resulting from merged labels. Finally, we add all undominated negative reduced-cost routes to the master problem.

Acceleration Techniques

To accelerate the pricing process, we use a heuristic pricing procedure, the so-called *limited discrepancy search* (LDS) introduced by Feillet *et al.* (2007), and a heuristic dominance.

Algorithm 3.1: Computation of Optimal Service Start Times T_i and T_j when Merging

$\Delta := t^*_{fw} + t_{ij} - t^*_{bw}$, $T_i := t^*_{fw}$, $T_j := t^*_{bw}$
if $(s^n_{fw} < 0)$ **then** $p_{fw} := n$ **else** $p_{fw} := n - 1$
if $(s^n_{bw} > 0)$ **then** $p_{bw} := n$ **else** $p_{bw} := n - 1$
while $(\Delta > 0)$ **do**
  if $(|s^{p_{fw}}_{fw}| < |s^{p_{bw}}_{bw})|$ **then**
    $\partial := \min\{T_i - t^{p_{fw}}_{fw}, \Delta\}$
    $T_i := T_i - \partial$, $p_{fw} := p_{fw} - 1$
  else
    $\partial := \min\{t^{p_{bw}}_{bw} - T_j, \Delta\}$
    $T_j := T_j + \partial$, $p_{bw} := p_{bw} - 1$
  $\Delta := \Delta - \partial$

Result: Optimal service start times T_i and T_j

LDS works as follows. In each pricing iteration, the outgoing arcs of each vertex are partitioned into "good" and "bad" arcs. The arcs with the lowest current modified arc costs and all arcs incident to the start or the end depot are considered good arcs, the others are regarded as bad arcs. The labels have an additional attribute storing the number of bad arcs used in the associated path. The value of this attribute is incremented by one each time a label is extended along a bad arc. Only those labels are considered feasible for which the number of bad arcs used is below a specified upper bound, the discrepancy limit. This limit is set to a fixed value during the complete solution process.

As for heuristic dominance, we use a pairwise comparison of two labels L and L', and if $c^*_L < c^*_{L'}$ and $t^1_L < t^1_{L'}$, we discard L'. The limited discrepancy search as well as the heuristic dominance are applied in each column generation iteration. Only if they fail to produce a negative reduced cost column, the exact pricing and dominance algorithms are applied.

3.5.3 Branching Strategy

Let $\tilde{\lambda}^{aq}$, $\tilde{u}_r$, and $\tilde{x}^a_{ij} = \sum_{q\in\Omega^a} X^q_{ij}\tilde{\lambda}^{aq}$ be the values of the corresponding decision variables λ^{aq}, u_r, and x^a_{ij}. We apply the following five-stage hierarchical branching scheme: First, if the overall number of unserved requests, i.e., $u_\Sigma = \sum_{r\in R}\tilde{u}_r$, is fractional, we create the two branches $\sum_{r\in R} u_r \le \lfloor u_\Sigma \rfloor$ and $\sum_{r\in R} u_r \ge \lceil u_\Sigma \rceil$. If the objective (3.1a) prioritizes request fulfillment, i.e., $\gamma \gg \alpha, \beta$, the latter branch is obsolete. Second, we branch on the individual u_r variables, where one request r^* with $\tilde{u}_{r^*}$ closest to $1/2$ is selected, and the two branches $u_{r^*} = 0$ and $u_{r^*} = 1$ are created. Third, we branch on the overall number of active vehicles in use: If $a_\Sigma = \sum_{a\in A}\sum_{q\in\Omega^a}\tilde{\lambda}^{aq}$ is fractional, the resulting branches are given by $\sum_{a\in A}\sum_{q\in\Omega^a}\lambda^{aq} \le \lfloor a_\Sigma \rfloor$ and $\sum_{a\in A}\sum_{q\in\Omega^a}\lambda^{aq} \ge \lceil a_\Sigma \rceil$. All these branching

rules are put into effect by adding a constraint to the master program (3.1). In addition, when the second branching rule is applied and u_{r^*} is set to 1, all the subgraphs induced by vertices in $V^-_{r^*} \cup W^+_{r^*} \cup W^-_{r^*} \cup V^+_{r^*}$ can be removed from all networks G^a, $a \in A$.

Fourth, note that for arcs $(i_p, j_{p'}) \in E$ that have both endpoints in N^R, at most one of the arcs $\bigcup_{q,q' \in P}\{(i_q, j_{q'})\}$ of one of the networks G^a, $a \in A$, can be present in a solution. Hence, if $\Sigma_{ij} = \sum_{a \in A} \sum_{q,q' \in P^a} \tilde{x}^a_{i_q,j_{q'}}$ is fractional, we create two branches by setting $\sum_{a \in A} \sum_{q,q' \in P^a} x^a_{i_q,j_{q'}}$ to zero and one respectively. If only one of the endpoints is in N^R, we can apply a similar branching rule. If several Σ_{ij} are fractional, we choose a pair (i, j) with value closest to $1/2$. The zero-branch is implemented by eliminating the associated arcs from all networks G^a, $a \in A$. The one-branch first fixes $u_r = 0$ for the corresponding request(s) r associated with i_p and $j_{p'}$ and eliminates incompatible arcs from all networks G^a, $a \in A$.

The fifth and last rule is branching on individual arcs, which finally ensures integrality of the x^a_{ij} and u_r variables. (Recall that model (3.1) poses no integer requirement on the λ^{aq} variables.) If $\tilde{x}^a_{ij}$ for an arc $(i, j) \in E^a$ is fractional, then one can branch on $x^a_{ij} = 0$ and $x^a_{ij} = 1$, where we select the combination (a, i, j) for which $\tilde{x}^a_{ij}$ is closest to $1/2$. Branching on individual arcs $(i, j) \in E^a$ is implemented by eliminating (i, j) from the underlying network G^a for $x^a_{ij} = 0$, while for the branch $x^a_{ij} = 1$, all ingoing arcs $\delta^-(i)$ are eliminated for networks $G^{a'}, a' \neq a$, and all arcs $\delta^+(i) \setminus \{(i, j)\}$ are eliminated from the network G^a.

As branch-and-bound node-selection rule, we apply a best-bound-first strategy, because our primary goal is to improve the dual bound.

3.5.4 Cutting Planes

To strengthen the formulation, we extend formulation (3.1) with subset-row inequalities (Jepsen *et al.*, 2008). For the APVRP, a valid inequality is defined on a subset of tasks instead of vertices as done for defining ng-neighborhoods, see Subsection 3.5.2. We restrict ourselves to those inequalities defined on three tasks as proposed by Jepsen *et al.* (2008) because they can be separated by straightforward enumeration. The inequality for a task set U_k, in the following denoted by $SR(U_k)$, is given by $\sum_{a \in A} \sum_{q \in \Omega} \lfloor \frac{h^q}{2} \rfloor \lambda^{aq} \leq 1$, where h^q is the number of times route q serves a task in U_k.

The addition of subset-row inequalities in the master problem requires the following adjustments to our pricing problems: Let $\eta_k \leq 0$ be the dual price of the subset-row inequality $SR(U_k)$. The value η_k must be subtracted from the reduced costs for every second service to tasks in U_k. Therefore, an additional binary resource sr^k, one for each inequality $SR(U_k)$, is necessary in the labeling algorithm for indicating the parity of the number of times a task in U_k is served. Note that the same task may be served more than once in the ng-path relaxation.

Jepsen *et al.* (2008) have proposed a tailored dominance rule that avoids a point-wise comparison of all resources sr^k and thereby reduces the number of incomparable labels significantly. In the APVRP case, the dominance rule changes as follows: A precondition for a label L' to dominate another label L is $S_{L'} \subseteq S_L$. Moreover, let $H = \{k : sr^k_{L'} =$

$1, sr_L^k = 0\}$ (the index set of new resources on which L' is inferior compared to L). Then, dominance is given at those times T where $c'_L(T) - \sum_{k \in H} \eta_k \leq c_L(T)$.

In addition, the merge procedure of the bidirectional labeling algorithm slightly changes. If both a forward and a backward path have served an odd number of tasks in U_k, then the dual price η_k of the subset-row inequality $SR(U_k)$ has to be subtracted from the reduced costs.

3.6 Experimental Results

The results reported in this section were obtained using a standard PC with an Intel(R) Core(TM) i7-2600 3.4 GHz processor and 16 GB of main memory. The algorithms were coded in C++ with MS-Visual Studio 2010. The callable library of CPLEX 12.5 was used for solving the linear relaxations of the restricted master program in the column-generation algorithm.

3.6.1 Test Instances

Meisel and Kopfer (2014) created a set of 180 APVRP test instances (henceforth referred to as MK instances) with the following characteristics: The instances range in size from 38-1600 tasks, 2-25 active vehicles, 4-50 passive vehicles, and a planning horizon t^{max} of 2500-5000 time units. Locations are randomly placed in a 100×100 area, distances and travel times are set to the Euclidean distance, time windows are of width 1000 with random start times in $[0, t^{max} - 1000]$, and service times for loading and unloading range between 50 and 100 time units. Each passive vehicle is compatible with 2 active vehicles, each request is compatible with 3 passive vehicles. Due to their huge planning horizon t^{max} where time units can model minutes of a working week, these instances are very hard to solve with an exact approach.

Hence, we additionally created a new set of instances, in which a time unit can model 10 minutes of a working week. It is clear that our new instances are on average easier to solve than the MK instances, however, the chosen time discretization is fine enough for many practical situations. The new set contains 20 instances with 38 tasks, 2 active, and 4 passive vehicles (leading to extended networks with approximately 170 vertices and 3500 arcs), and 20 instances with 76 tasks, 4 active, and 8 passive vehicles (resulting in extended networks with ca. 650 vertices and 4000 arcs). The instances have a planning horizon of 1000 time units, and service times s_r^+ and s_r^- vary between 25 and 50. To each of the 40 instances, we assigned time windows that allow different levels of flexibility: For each request $r \in R$, we first compute the travel time between its pickup and its delivery location. Initially, each request time window $[e_r, l_r]$ (refer to Table 3.1 for the definition of e_r and l_r) is set so that the time window width $l_r - e_r$ is equal to the computed travel time plus the service times s_r^+ and s_r^-. By adding a *time window flexibility* of $F = 25, 50, 100$, and 200, we create 160 new instances. A time window flexibility of F means that e_r and l_r are further shifted apart by exactly F time units (it is ensured that

the new $[e_r, l_r]$ is in the overall planning horizon $[0, t^{max}]$). All other characteristics of the new instances are the same as in the MK instances.

3.6.2 Algorithmic Setup

For all experiments, we used objective function weights of $\alpha = 10$, $\beta = 1$, and $\gamma = 10{,}000$, i.e., a hierarchical objective of first fulfilling as many requests as possible, then minimizing traveled distance, and then minimizing route completion time. Feasibility of the restricted master program is ensured by initializing it with dummy routes each bringing a passive vehicle from its origin to its destination as well as with columns for the u_r variables. We set a CPU time limit of 2 hours. Preliminary tests showed that bidirectional labeling was superior to the unidirectional variant, that the best results were obtained with an ng-neighborhood size of 15, applying limited discrepancy search and heuristic dominance during the pricing as described in Section 11, and stopping each labeling procedure as soon as 100 or more negative reduced cost routes have been found.

3.6.3 Algorithmic Performance

Here, we present the experimental results for different setups. First, we present the result of the branch-and-price algorithm obtained without using subset-row inequalities. The next paragraph contains the results of the full branch-and-price-and-cut algorithm. Finally, we discuss the impact of time window flexibility on the solution quality.

Branch-and-Price Results

In a first series of experiments, we applied our algorithm without adding cutting planes. As for the MK instances, Meisel and Kopfer (2014) could not solve any of these to optimality with the branch-and-cut algorithm presented in their paper. Our algorithm was able to solve to optimality one 38-task MK instance within the 2-hour time limit. For this solved instance the previous upper bound was improved by 10.05 %. The average gap between the upper and the lower bound after 2 hours was 6.79 % for the MK instances, with a minimum non-zero gap of 3.47 % and a maximum gap of 13.90 %. The average increase in the lower bound at the root node of the branch-and-bound tree compared to the one of the branch-and-cut by Meisel and Kopfer (2014) was a significant 18.35 %.

Tables 3.4 and 3.5 report the results on the test instances we created. Both tables have the same columns. The first column indicates the time window flexibility F of the instances considered in each row, the second specifies the number of instances solved to optimality. The next columns indicate minimal, average, and maximal values of (i) the CPU times, (ii) the gap between the value of the LP relaxation at the root node and the best known upper bound (also using upper bounds for instances with smaller F as bounds for instances with a bigger F), (iii) the percentage of the root gap closed at the end of the optimization (100.0 % for instances that were solved to optimality), and (iv) the number of branch-and-bound nodes solved. It can be seen from the tables that

we are able to consistently solve instances with 38 tasks and a time window flexibility of up to 100. In general, the instances become more difficult with increasing time window flexibility, and timeout is reached for most 76-task instances, with significant gaps remaining. For the 76-task instances with a time window flexibility of 100 and 200, the number of solved branch-and-bound nodes even decreases, on average, compared to the instances with shorter time windows. This means that the time needed to solve a node increases considerably. Analyses showed that this is mostly caused by the slow convergence of the master program as well as the increasing solution times for the pricing problems, which, in turn, are due to the increasing number of labels created because of the increasing flexibility offered by longer time windows.

TW flex.	# Solved	Time [sec]			Gap at root [%]			Gap closed [%]			# Nodes solved		
		min	avg	max	min	avg	max	min	avg	max	min	avg	max
25	20/20	36	191	931	0.00	0.43	1.65	100.00	100.00	100.00	1	5.40	23
50	20/20	43	362	2485	0.00	0.55	2.11	100.00	100.00	100.00	1	8.30	47
100	19/20	38	1076	7200	0.00	0.99	3.00	64.29	98.21	100.00	1	16.95	56
200	14/20	109	3466	7200	0.00	2.67	9.89	8.94	79.94	100.00	1	26.20	106
All	73/80		1274			1.16			94.54			14.21	

Table 3.4: Results on 38-Task Instances without Cuts

TW flex.	# Solved	Time [sec]			Gap at root [%]			Gap closed [%]			# Nodes solved		
		min	avg	max	min	avg	max	min	avg	max	min	avg	max
25	17/20	131	3594	7200	0.00	0.80	2.39	46.52	90.54	100.00	1	22.70	49
50	8/20	282	5419	7200	0.28	1.55	4.30	21.26	67.96	100.00	3	25.90	46
100	2/20	2814	6882	7200	0.47	3.53	6.94	10.70	34.11	100.00	11	25.50	45
200	0/20	7200	7200	7200	6.10	8.48	13.47	1.79	6.27	15.35	5	11.60	19
All	27/80		5775			3.59			49.72			21.43	

Table 3.5: Results on 76-Task Instances without Cuts

Branch-and-Price-and-Cut Results

The dynamic programming algorithm can become quite slow when too many subset-row inequalities are present in the restricted master problem. To overcome this, we use the following two parameters to define the strategy used for separating inequalities: Cut_{max} gives the maximum number of inequalities that can be added to the restricted master problem. When more than Cut_{max} inequalities are violated, we choose the most violated ones. Cut_{task} gives the maximum number of inequalities that a single task can be involved in. Preliminary tests have shown that the best results are obtained choosing $Cut_{max} = 5$ and Cut_{task}=2.

The results for the MK instances are similar to those without using subset-row inequalities: One instance was solved to optimality, the average gap between upper and lower bound after 2 hours was 7.30 %, with a minimum non-zero gap of 3.16 % and a

maximum gap of 13.90 %. Tables 3.6 and 3.7 report the results on the test instances we created. The columns contain the same information as in the previous paragraph, but the gap at the root lower bound is missing because it is the same as in the tables before. In addition, we have the minimum, average and maximum percentage that the root gap was closed only by the subset-row inequalities. Also here the results do not differ significantly. We can solve as many 38-task instances as before, but the average computation time increases by nearly 200 seconds and the average percentage the root gap was closed slightly decreases. The detailed results in the electronic appendix show that one of the previously solved instances was not solved, but another one was solved for the first time. For the 76-task instances, we can solve one less instance, but the average computation time decreases by more than 300 seconds and the average percentage the root gap was closed increases by more than two percent. For both instance sets the number of solved branch-and-bound nodes decreases by around 3.5, implying that the time for solving one branch-and-bound-node increases significantly. This is due to the fact that the exact dominance becomes quite weak, even with a small number of subset-row inequalities. Detailed results have shown that nearly three times more labels are generated in the dynamic programming algorithm when subset-row inequalities are present in the restricted master problem. This is due to the fact that we need several labels to dominate one other label.

TW flex.	# Solved	Time [sec]			Gap closed by cuts [%]			Gap closed overall [%]			# Nodes solved		
		min	avg	max	min	avg	max	min	avg	max	min	avg	max
25	20/20	37	264	1696	3.13	67.57	100.00	100.00	100.00	100.00	1	4.50	18
50	20/20	49	266	1210	0.00	60.97	100.00	100.00	100.00	100.00	1	6.75	34
100	19/20	26	1398	7200	0.00	31.50	100.00	66.52	98.33	100.00	1	15.35	36
200	14/20	68	3896	7200	2.81	31.50	100.00	7.22	78.76	100.00	1	16.20	48
All			1456			47.88			94.27			10.70	

Table 3.6: Results on 38-Task Instances with Subset-Row Inequalities

TW flex.	# Solved	Time [sec]			Gap closed by cuts [%]			Gap closed overall [%]			# Nodes solved		
		min	avg	max	min	avg	max	min	avg	max	min	avg	max
25	17/20	101	2550	7200	0.00	35.90	100.00	67.96	92.48	100.00	1	17.80	64
50	7/20	289	5395	7200	3.23	20.39	58.70	22.06	73.26	100.00	4	26.70	55
100	2/20	1118	6678	7200	1.03	11.64	77.50	9.58	35.92	100.00	7	19.95	40
200	0/20	7200	7200	7200	0.66	2.73	6.56	3.29	5.82	10.48	3	7.75	17
All			5456			17.66			51.87			18.05	

Table 3.7: Results on 76-Task Instances with Subset-Row Inequalities

Impact of Time Window Flexibility

Table 3.8 shows that there is a positive impact of increasing time window flexibility on possible solution quality. The values in the table reflect the objective function improvement obtained with increasing time window flexibility. For example, for the 38-task

instances, the average objective function value improves by 1.20 % if the time window flexibility is increased from 25 to 50.

TW flex. increase	Objective improvement [%]	
	38 tasks	76 tasks
25 → 50	1.20	1.54
50 → 100	1.94	1.56
100 → 200	5.84	N/A

Table 3.8: Impact of Time Window Flexibility

3.7 Conclusions

This paper has investigated the exact solution of the active-passive vehicle-routing problem (APVRP) by means of a branch-and-price-and-cut method. The problem includes synchronization constraints for the operations and the movement of active and passive vehicles. It supports a flexible coupling of these transport resources in order to achieve an efficient resource utilization and high-quality transport solutions. The synchronization constraints introduce linear vertex costs in the ESPPRC pricing problem, thus significantly complicating its solution. To solve the pricing problems, we use a bidirectional labeling algorithm and apply a refinement of the ng-path relaxation approach. What is more, we propose a sophisticated procedure for merging forward and backward labels. Computational experiments show that our method delivers improved results for the APVRP benchmark instances introduced by Meisel and Kopfer (2014). For a newly created benchmark set, it is capable of solving instances with 76 tasks, 4 active, and 8 passive vehicles to optimality.

In its current form, the APVRP already covers a number of relevant applications from the areas of distribution logistics, health care management, the security industry and others. Nevertheless, there are several further features of real-world problems that are not yet supported by the presented model. This includes, for example, the possibility of a temporary drop-off of passive vehicles at intermediate locations (e.g., public parking lots) that neither belong to the set of initial or final locations nor to the set of pickup and delivery locations. Such drop-off locations could reduce the detours needed by active vehicles for the exchange of passive vehicles. Furthermore, a rendezvous of active vehicles is required in some applications where passive vehicles cannot be left behind but must be handed over directly from one active vehicle to another. This is the case if only one of the active vehicles is equipped with a certain loading equipment, if the passive vehicle cannot be left unattended for security reasons, or if cargo documents need to be exchanged among the active vehicles. Moreover, in many or even most practical applications, vehicles with a capacity of more than one are used, and performing load transshipments between such vehicles is quite common in practice. These features and their corresponding synchronization requirements represent practically relevant and

mathematically non-trivial extensions of the APVRP and will be a subject of our future research.

Acknowledgement. This research was funded by the Deutsche Forschungsgemeinschaft (DFG) under grants no. IR 122/5-2 and DR 963/2-1.

References

Afifi, S., Dang, D.-C., and Moukrim, A. (2015). Heuristic solutions for the vehicle routing problem with time windows and synchronized visits. *Optimization Letters, in press.*

Andersson, H., Duesund, J. M., and Fagerholt, K. (2011). Ship routing and scheduling with cargo coupling and synchronization constraints. *Computers & Industrial Engineering*, **61**(4), 1107–1116.

Baldacci, R., Mingozzi, A., and Roberti, R. (2011). New route relaxation and pricing strategies for the vehicle routing problem. *Operations Research*, **59**(5), 1269–1283.

Bredström, D. and Rönnqvist, M. (2008). Combined vehicle routing and scheduling with temporal precedence and synchronization constraints. *European Journal of Operational Research*, **191**(1), 19–31.

Buijs, P., Vis, I. F., and Carlo, H. J. (2014). Synchronization in cross-docking networks: A research classification and framework. *European Journal of Operational Research*, **239**(3), 593–608.

Cheung, R. K., Shi, N., Powell, W.-B., and Simao, H. P. (2008). An attribute-decision model for cross-border drayage problem. *Transportation Research Part E: Logistics and Transportation Review*, **44**(2), 217–234.

Christiansen, M. and Nygreen, B. (1998). A method for solving ship routing problems with inventory constraints. *Annals of Operations Research*, **81**(0), 357–378.

Desaulniers, G., Desrosiers, J., Ioachim, I., Solomon, M., Soumis, F., and Villeneuve, D. (1998). A unified framework for deterministic time constrained vehicle routing and crew scheduling problems. In T. Crainic and G. Laporte, editors, *Fleet Management and Logistics*, chapter 3, pages 57–93. Kluwer Academic Publisher, Boston, Dordrecht, London.

Desaulniers, G., Desrosiers, J., and Solomon, M., editors (2005). *Column Generation.* Springer, New York, NY.

Desaulniers, G., Madsen, O. B., and Ropke, S. (2014). *The Vehicle Routing Problem with Time Windows*, chapter 5, pages 119–159. In Toth and Vigo (2014).

Dohn, A., Kolind, E., and Clausen, J. (2009). The manpower allocation problem with time windows and job-teaming constraints: A branch-and-price approach. *Computers & Operations Research*, **36**, 1145–1157.

Dohn, A., Rasmussen, M., and Larsen, J. (2011). The vehicle routing problem with time windows and temporal dependencies. *Networks*, **58**, 273–289.

Drexl, M. (2007). *On Some Generalized Routing Problems*. Ph.D. thesis, Faculty of Business and Economics, RWTH Aachen University.

Drexl, M. (2012). Synchronization in vehicle routing-a survey of vrps with multiple synchronization constraints. *Transportation Science*, **46**(3), 297–316.

Drexl, M. (2013). Applications of the vehicle routing problem with trailers and transshipments. *European Journal of Operational Research*, **227**(2), 275–283.

Drexl, M. (2014). Branch-and-cut algorithms for the vehicle routing problem with trailers and transshipments. *Networks*, **63**(1), 119–133.

Drexl, M., Rieck, J., Sigl, T., and Press, B. (2013). Simultaneous vehicle and crew routing and scheduling for partial- and full-load long-distance road transport. *BuR - Business Research*, **6**(2), 242–264.

Dror, M. (1994). Note on the complexity of the shortest path models for column generation in VRPTW. *Operations Research*, **42**(5), 977–978.

Feillet, D., Gendreau, M., and Rousseau, L.-M. (2007). New refinements for the solution of vehicle routing problems with branch and price. *INFOR*, **45**(4), 239–256.

Hachemi, N. E., Gendreau, M., and Rousseau, L.-M. (2013). A heuristic to solve the synchronized log-truck scheduling problem. *Computers & Operations Research*, **40**(3), 666–673.

Hollis, B., Forbes, M., and Douglas, B. (2006). Vehicle routing and crew scheduling for metropolitan mail distribution at Australia Post. *European Journal of Operational Research*, **173**(1), 133–150.

Huber, S. and Geiger, M. (2014). Swap body vehicle routing problem: A heuristic solution approach. In R. González-Ramírez, F. Schulte, S. Voß, and J. Ceroni Díaz, editors, *Computational Logistics*, volume 8760 of *Lecture Notes in Computer Science*, pages 16–30. Springer International Publishing.

Ioachim, I., Gélinas, S., Desrosiers, J., and Soumis, F. (1998). A dynamic programming algorithm for the shortest path problem with time windows and linear node costs. *Networks*, **31**, 193–204.

Irnich, S. and Desaulniers, G. (2005). Shortest path problems with resource constraints. In Desaulniers *et al.* (2005), chapter 2, pages 33–65.

Irnich, S., Toth, P., and Vigo, D. (2014). *The Family of Vehicle Routing Problems*, chapter 1, pages 1–33. In Toth and Vigo (2014).

Jans, R. (2010). Classification of Dantzig-Wolfe reformulations for binary mixed integer programming problems. *European Journal of Operational Research*, **204**(2), 251–254.

Jepsen, M., Petersen, B., Spoorendonk, S., and Pisinger, D. (2008). Subset-row inequalities applied to the vehicle-routing problem with time windows. *Operations Research*, **56**(2), 497–511.

Kergosien, Y., Lenté, C., Piton, D., and Billaut, J.-C. (2011). A tabu search heuristic for the dynamic transportation of patients between care units. *European Journal of Operational Research*, **214**(2), 442–452.

Kergosien, Y., Lenté, C., Billaut, J.-C., and Perrin, S. (2013). Metaheuristic algorithms for solving two interconnected vehicle routing problems in a hospital complex. *Computers & Operations Research*, **40**(10), 2508 – 2518.

Kim, B.-I., Koo, J., and Park, J. (2010). The combined manpower-vehicle routing problem for multi-staged services. *Expert Systems with Applications*, **37**(12), 8424–8431.

Labadie, N., Prins, C., and Yang, Y. (2014). Iterated local search for a vehicle routing problem with synchronization constraints. In *ICORES 2014 - Proceedings of the 3rd International Conference on Operations Research and Enterprise Systems, Angers, Loire Valley, France, March 6-8, 2014*, pages 257–263.

Lahyani, R., Khemakhem, M., and Semet, F. (2015). Rich vehicle routing problems: From a taxonomy to a definition. *European Journal of Operational Research*, **241**(1), 1–14.

Laporte, G. (2016). Scheduling issues in vehicle routing. *Annals of Operations Research*, **236**(2), 463–474.

Liberatore, F., Righini, G., and Salani, M. (2011). A column generation algorithm for the vehicle routing problem with soft time windows. *4OR*, **9**(1), 49–82.

Lübbecke, M. and Desrosiers, J. (2005). Selected topics in column generation. *Operations Research*, **53**(6), 1007–1023.

Mankowska, D. S., Meisel, F., and Bierwirth, C. (2013). The home health care routing and scheduling problem with interdependent services. *Health Care Management Science*, **17**(1), 15–30.

Meisel, F. and Kopfer, H. (2014). Synchronized routing of active and passive means of transport. *OR Spectrum*, **36**(2), 297–322.

Morais, V. W., Mateus, G. R., and Noronha, T. F. (2014). Iterated local search heuristics for the vehicle routing problem with cross-docking. *Expert Systems with Applications*, **41**(16), 7495–7506.

Mues, C. and Pickl, S. (2005). Transshipment and time windows in vehicle routing. In *Proceedings of the 8th International Symposium on Parallel Architectures, Algorithms and Networks*, pages 113–119, Piscataway. IEEE.

Salani, M. (2005). *Branch-and-price algorithms for Vehicle Routing Problems*. Ph.D. dissertation, Università degli Studi di Milano, Facoltà die Scienze Matematiche, Fisiche e Naturali, Dipartimento di Tecnologie dell'Informatione, Milan, Italy.

Salazar-Aguilar, M. A., Langevin, A., and Laporte, G. (2013). The synchronized arc and node routing problem: Application to road marking. *Computers & Operations Research*, **40**(7), 1708–1715.

Smilowitz, K. (2006). Multi-resource routing with flexible tasks: An application in drayage operations. *IIE Transactions*, **38**(7), 555–568.

Spliet, R. and Gabor, A. (2014). The time window assignment vehicle routing problem. *Transportation Science*.

Tilk, C. and Irnich, S. (2016). Dynamic programming for the minimum tour duration problem. *Transportation Science*. Forthcoming.

Toth, P. and Vigo, D., editors (2014). *Vehicle Routing*. MOS-SIAM Series on Optimization. Society for Industrial and Applied Mathematics, Philadelphia, PA.

Xue, Z., Zhang, C., Lin, W.-H., Miao, L., and Yang, P. (2014). A tabu search heuristic for the local container drayage problem under a new operation mode. *Transportation Research Part E: Logistics and Transportation Review*, **62**, 136–150.

Zhang, R., Yun, W. Y., and Kopfer, H. (2010). Heuristic-based truck scheduling for inland container transportation. *OR Spectrum*, **32**(3), 787–808.

Zhang, R., Yun, W. Y., and Kopfer, H. (2013). Multi-size container transportation by truck: modeling and optimization. *Flexible Services and Manufacturing Journal*, **27**(2-3), 403–430.

Zhang, R., Lu, J.-C., and Wang, D. (2014). Container drayage problem with flexible orders and its near real-time solution strategies. *Transportation Research Part E: Logistics and Transportation Review*, **61**, 235–251.

Appendix

3.A Appendix: Notation

Sets	
R	Set of all requests
r	A request
A	Set of classes of active vehicles
a	A class of active vehicles
K_a	Number of active vehicles of class a
P	Set of passive vehicles
p	A passive vehicle
R^p	Set of requests compatible with passive vehicle p
P^r	Set of passive vehicles compatible with request r
P^a	Set of passive vehicles compatible with active vehicle a
A^p	Set of classes of active vehicles compatible with passive vehicle p
Locations	
ℓ_r^+	The pickup location of request r
ℓ_r^-	The delivery location of request r
o	Initial location of all active vehicles
d	Final location of all active vehicles
o_p	Initial location of passive vehicle p
d_p	Final location of passive vehicle p
Cost parameters	
α	Weight for travel distance
β	Weight for arrival time at the final depot d
γ	Penalty for not fulfilling a request
c_{ij}	Travel distance from vertex i to vertex j
Time parameters	
t_{ij}	Travel and service time from vertex i to vertex j
$[e_r, l_r]$	Time window for fulfilling request $r \in R$
s_r^+	The service duration to process an empty passive vehicle at request r
s_r^-	The service duration to process a loaded passive vehicle at request r
$[0, t^{max}]$	Planning horizon
Network parameters	
$G^a =$	
(V^a, E^a)	Extended graph for active vehicle class $a \in A$ with vertex set V^a and arc set E^a

(Continued on next page)

(Continued from previous page)

$\delta^+(i)$	$\{j \in V^a : (i,j) \in E^a\}$; forward star of vertex $i \in V^a$
$\delta^-(i)$	$\{j \in V^a : (j,i) \in E^a\}$; backward star of vertex $i \in V^a$
N^R	$\bigcup_{r \in R} (V_r^- \cup W_r^+ \cup W_r^- \cup V_r^+)$; set of request vertices

For each passive vehicle $p \in P^a$ and each compatible request $r \in R^p$:

v_{rp}^-	Vertex where empty passive vehicle is delivered
w_{rp}^+	Vertex where loaded passive vehicle is picked up
w_{rp}^-	Vertex where loaded passive vehicle is delivered
v_{rp}^+	Vertex where empty passive vehicle is picked up
N^a	$\cup_{p \in P^a, r \in R^p} \{v_{rp}^-, w_{rp}^+, w_{rp}^-, v_{rp}^+\}$; set of request vertices for each passive vehicle $p \in P^a$ and each compatible request $r \in R^p$

For each request $r \in R$:

V_r^-	$\{v_{rp}^- : p \in P^r\}$; set of vertices where empty passive vehicle may be delivered for pickup of r
W_r^+	$\{w_{rp}^+ : p \in P^r\}$; set of vertices where passive vehicle loaded with r may be picked up
W_r^-	$\{w_{rp}^- : p \in P^r\}$; set of vertices to where passive vehicle loaded with r may be delivered
V_r^+	$\{v_{rp}^+ : p \in P^r\}$; set of vertices where empty passive vehicle may be picked up after delivery of r

Set-partitioning formulation parameters

Ω^a	Set of all routes that can be defined in graph $G^a = (V^a, E^a)$, $a \in A$
q	A route in $\bigcup_{a \in A} \Omega^a$
X_{ij}^q	Number of times arc (i,j) is traversed on route q
T_i^q	Service start time at request vertex i on route q
b_i^q	Number of times vertex i is visited on route q
c^q	Cost of route q; weighted sum of arrival time at the destination d and length of the route
λ^{aq}	Continuous variables measuring the flow of active vehicles of class $a \in A$ along route $q \in \Omega^a$
x_{ij}^a	Binary variables indicating the number of times arc $(i,j) \in E^a$ is traversed by an active vehicle of class $a \in A$
u_r	Binary variable indicating whether or not request $r \in R$ remains unfulfilled

Pricing problem parameters

π_r	Unrestricted dual variable of constraint (3.1b)
ϕ_{rp}^+	Unrestricted dual variable of constraint (3.1c)

(Continued on next page)

(Continued from previous page)

ϕ_{rp}^{-}	Unrestricted dual variable of constraint (3.1d)
τ_r^{+}	Nonnegative dual variable of constraint (3.1e)
τ_r^{-}	Nonnegative dual variable of constraint (3.1f)
μ_p^P	Unrestricted dual variable of constraint (3.1g)
μ_a^A	Nonpositive dual variable of constraint (3.1h)
$\tilde{c}_{ij}$	Reduced costs of arc (i, j)
$\tilde{c}_i$	Linear vertex costs of vertex i
x_{ij}	Binary variable indicating whether or not arc $(i, j) \in E^a$ is traversed
T_i	Continuous variable indicating the point in time when the service at vertex $i \in V^a$ begins

Labeling algorithm parameters

τ	A task
$\mathcal{N}_i$	ng-neighborhood at vertex i; $\mathcal{N}_i \subset N^R$; $\mathcal{N}_i$ contains i and the ν closest request vertices j for which a cycle $(j, \ldots, i, \ldots, j)$ would be feasible when taking time windows and travel and service times into account
ν	Size of ng-neighborhood
τ_p^o	Task of picking up the passive vehicle $p \in P$ at its initial location
τ_r^1	Task of delivering an empty passive vehicle for request $r \in R$
τ_r^2	Task of picking up and delivering a passive vehicle loaded with request $r \in R$
τ_r^3	Task of picking up an empty passive vehicle after delivering request $r \in R$
τ_p^d	Task of delivering the passive vehicle $p \in P$ at its final location
L	$L = ((t^p, s^p)_{p=1}^{n+1}, c^1, S, (I^d)_{d=1}^k)$; a label
n	Number of time-slope pairs/pieces in tradeoff curve
$(t^p, s^p)_{p=1}^n$	The n pieces of the tradeoff curve; $s^p < 0, p = 1, \ldots, n-1$ and $s^n \leq 0$
t^{n+1}	End time of last piece n in tradeoff curve
$c(T)$	$= c^1 + \sum_{p=0}^{q-1} s^p(t^{p+1} - t^p) + s^q(T - t^q)$ for $t^q \leq T \leq t^{q+1}$; the piecewise linear cost function/tradeoff curve of a label, i.e., a label's associated partial path; provides the minimal costs $c(t)$ incurred by the path when the service at the last vertex of the path starts at time t
c^1	Reduced costs at start time of first piece in tradeoff curve
S	Tasks that have already been performed or are unreachable along the route and which are taken into account for checking elementarity in the label extension step according to the ng-path relaxation
k	Number of intervals on which the associated label is dominated
$(I^d)_{d=1}^k$	The k intervals on which the associated label is dominated

(Continued on next page)

(Continued from previous page)

c^*	$= c^1 - \sum_{p=1}^{n} s^p(t^{p+1} - t^p)$; optimal (reduced) costs
t^*	$= t^n$ for $s^n = 0$, t^{n+1} otherwise; earliest time to obtain costs c^*
s^{n+1}	$= 0$; slope of $(n+1)$th piece; defined for convenience
$\mathcal{T}_j^{\text{test}}$	Before extending a label L at vertex i to vertex j, tasks in $\mathcal{T}_j^{\text{test}}$ are tested and L is extended only if none of these tasks has already been fulfilled along the partial path represented by L
$\mathcal{T}_j^{\text{set}}$	Upon extending a label L at vertex i to vertex j, tasks in $\mathcal{T}_j^{\text{set}}$ are marked as fulfilled
f	Index of first linear slope piece to be kept when extending a label
g	Index of last linear slope piece to be kept when extending a label
δ	Piece indicator; one if a new linear slope piece must be created upon extension of a label, zero otherwise
$f_{ij}(t, s)$	$= (\{\max\{e_j, t + t_{ij}\}\}, \min\{0, s + \tilde{c}_j\})$; function for extending pieces during label extension

3.B Detailed Computational Results for Individual Instances

This section contains detailed computational results for all instances. For all experiments, we used objective function weights of $\alpha = 10$, $\beta = 1$, and $\gamma = 10{,}000$, i.e., a hierarchical objective of first fulfilling as many requests as possible, then minimizing traveled distance, and then minimizing route completion time. The configuration was as follows: First, bidirectional labeling with an ng-neighborhood size of 15 was applied. Second, the labeling was stopped as soon as 100 or more negative reduced cost routes had been found. Third, limited discrepancy search and the heuristic dominance were used as pricing heuristics.

A hard CPU time limit of 2 hours was set. Tables 3.10-3.18 present the results obtained without using subset-row inequalities. The first column indicates the name of the instances, the second gives the value of the best known feasible solution computed by any of the various settings used in preliminary tests. The third column gives the best upper bound computed without using subset-row inequalities. Numbers printed in bold indicate that the instance was solved to optimality. The subsequent two columns show the value of the lower bound at the root node and the lower bound at the end of the optimization. The sixth column specifies the gap between the best known solution and the lower bound at the root node in percent, and the seventh column lists the percentage of the root gap closed at the end of the optimization. The last two columns indicate the overall CPU time of the algorithm and the number of branch-and-bound nodes solved. Tables 3.19-3.27 present the results obtained with the full branch-and-price-and-cut algorithm. Herein, additional information on the lower bound after applying subset-row inequalities at the root node is given as well as the percentage the root gap was closed by this procedure.

Instance	Best	UB	LB		Gap [%]		Time	# Nodes
	known	**(Opt)**	Root	Tree	Root	closed	[sec]	solved
A01TW25	7721	**7721**	7721	7721	0.00	100.00	50	1
A02TW25	7577	**7577**	7544	7577	0.44	100.00	156	3
A03TW25	8198	**8198**	8198	8198	0.00	100.00	48	1
A04TW25	8114	**8114**	8096	8114	0.22	100.00	142	5
A05TW25	8861	**8861**	8850	8861	0.12	100.00	90	3
A06TW25	7233	**7233**	7233	7233	0.00	100.00	57	1
A07TW25	7767	**7767**	7767	7767	0.00	100.00	36	1
A08TW25	8168	**8168**	8168	8168	0.00	100.00	74	1
A09TW25	7653	**7653**	7590	7653	0.82	100.00	314	7
A10TW25	9583	**9583**	9425	9583	1.65	100.00	258	9
A11TW25	9237	**9237**	9225	9237	0.13	100.00	106	3
A12TW25	7720	**7720**	7720	7720	0.00	100.00	41	1
A13TW25	8791	**8791**	8742	8791	0.56	100.00	237	7
A14TW25	7632	**7632**	7600	7632	0.42	100.00	230	7
A15TW25	7539	**7539**	7521	7539	0.24	100.00	126	7
A16TW25	7814	**7814**	7703	7814	1.42	100.00	931	23
A17TW25	8603	**8603**	8502	8603	1.17	100.00	593	15
A18TW25	6943	**6943**	6943	6943	0.00	100.00	39	1
A19TW25	9231	**9231**	9106	9231	1.35	100.00	235	11
A20TW25	8665	**8665**	8665	8665	0.00	100.00	54	1
All					0.43	100.00	191	5.40

Table 3.10: Results of the Branch-and-Price Algorithm on 38-Task Instances with Time Window Flexibility 25

Instance	Best	UB	LB		Gap [%]		Time	# Nodes
	known	**(Opt)**	Root	Tree	Root	closed	[sec]	solved
A01TW50	7671	**7671**	7648	7671	0.30	100.00	214	5
A02TW50	7527	**7527**	7475	7527	0.69	100.00	529	13
A03TW50	8165	**8165**	8165	8165	0.00	100.00	43	1
A04TW50	7972	**7972**	7972	7972	0.00	100.00	51	1
A05TW50	8662	**8662**	8662	8662	0.00	100.00	57	1
A06TW50	7183	**7183**	7183	7183	0.00	100.00	52	1
A07TW50	7742	**7742**	7654	7742	1.14	100.00	501	11
A08TW50	8131	**8131**	8104	8131	0.33	100.00	178	3
A09TW50	7448	**7448**	7448	7448	0.00	100.00	74	1
A10TW50	9489	**9489**	9313	9489	1.85	100.00	820	23
A11TW50	9124	**9124**	9084	9124	0.44	100.00	207	7
A12TW50	7670	**7670**	7579	7670	1.19	100.00	696	17
A13TW50	8741	**8741**	8593	8741	1.69	100.00	670	17
A14TW50	7453	**7453**	7453	7453	0.00	100.00	53	1
A15TW50	7454	**7454**	7431	7454	0.31	100.00	115	5
A16TW50	7520	**7520**	7511	7520	0.12	100.00	176	5
A17TW50	8499	**8499**	8432	8499	0.79	100.00	199	5
A18TW50	6898	**6898**	6898	6898	0.00	100.00	56	1
A19TW50	9119	**9119**	8927	9119	2.11	100.00	2485	47
A20TW50	8633	**8633**	8633	8633	0.00	100.00	56	1
All					0.55	100.00	362	8.30

Table 3.11: Results of the Branch-and-Price Algorithm on 38-Task Instances with Time Window Flexibility 50

Instance	Best	UB	LB		Gap [%]		Time	# Nodes
	known	**(Opt)**	Root	Tree	Root	closed	[sec]	solved
A01TW100	7562	**7562**	7509	7562	0.70	100.00	1020	17
A02TW100	7477		7253	7397	3.00	64.29	7200	56
A03TW100	8140	**8140**	8140	8140	0.00	100.00	38	1
A04TW100	7852	**7852**	7841	7852	0.14	100.00	245	3
A05TW100	8537	**8537**	8489	8537	0.56	100.00	141	5
A06TW100	7122	**7122**	7117	7122	0.07	100.00	213	7
A07TW100	7448	**7448**	7385	7448	0.85	100.00	769	15
A08TW100	8050	**8050**	8036	8050	0.17	100.00	243	5
A09TW100	7287	**7287**	7287	7287	0.00	100.00	93	1
A10TW100	9054	**9054**	8996	9054	0.64	100.00	144	7
A11TW100	8993	**8993**	8924	8993	0.77	100.00	599	13
A12TW100	7626	**7626**	7516	7626	1.44	100.00	1758	43
A13TW100	8533	**8533**	8385	8533	1.73	100.00	740	25
A14TW100	7400	**7400**	7260	7400	1.89	100.00	2093	27
A15TW100	7368	**7368**	7312	7368	0.76	100.00	1164	33
A16TW100	7354	**7354**	7318	7354	0.49	100.00	157	3
A17TW100	8335	**8335**	8201	8335	1.61	100.00	647	19
A18TW100	6812	**6812**	6696	6812	1.70	100.00	3297	37
A19TW100	8418	**8418**	8373	8418	0.53	100.00	371	7
A20TW100	8490	**8490**	8262	8490	2.69	100.00	592	15
All					0.99	98.21	1076	16.95

Table 3.12: Results of the Branch-and-Price Algorithm on 38-Task Instances with Time Window Flexibility 100

Instance	Best	UB	LB		Gap [%]		Time	# Nodes
	known	**(Opt)**	Root	Tree	Root	closed	[sec]	solved
A01TW200	7562		6814	6887	9.89	9.76	7203	16
A02TW200	7063	**7063**	6976	7063	1.23	100.00	6643	21
A03TW200	7565	**7565**	7555	7565	0.13	100.00	158	3
A04TW200	7534	**7534**	7441	7534	1.23	100.00	2826	25
A05TW200	7851	**7851**	7787	7851	0.82	100.00	616	5
A06TW200	6389	**6389**	6295	6389	1.47	100.00	1290	13
A07TW200	7448		6741	6986	9.49	34.65	7201	59
A08TW200	7759	**7759**	7750	7759	0.12	100.00	153	3
A09TW200	7158	**7158**	7154	7158	0.06	100.00	310	3
A10TW200	8141	**8141**	8141	8141	0.00	100.00	109	1
A11TW200	8472	**8472**	8384	8472	1.04	100.00	2501	17
A12TW200	7495	**7495**	7414	7495	1.08	100.00	3222	33
A13TW200	8114		7880	8093	2.88	91.03	7200	106
A14TW200	7154	**7154**	6996	7154	2.21	100.00	4640	33
A15TW200	6804	**6804**	6770	6804	0.50	100.00	1431	11
A16TW200	7354		6880	7018	6.45	29.11	7200	74
A17TW200	7829	**7829**	7759	7829	0.89	100.00	1273	19
A18TW200	6812		6398	6435	6.08	8.94	7232	8
A19TW200	8418		7877	8014	6.43	25.32	7201	59
A20TW200	7842	**7842**	7728	7.842	1.45	100.00	900	15
All					2.67	79.94	3466	26.20

Table 3.13: Results of the Branch-and-Price Algorithm on 38-Task Instances with Time Window Flexibility 200

Instance	Best	UB	LB		Gap [%]		Time	# Nodes
	known	**(Opt)**	Root	Tree	Root	closed	[sec]	solved
B01TW25	16106	**16106**	15992	16106	0.71	100.00	4854	37
B02TW25	17617	**17617**	17350	17617	1.52	100.00	1612	19
B03TW25	15170		14808	15041	2.39	64.36	7200	45
B04TW25	18085	**18085**	18085	18085	0.00	100.00	131	1
B05TW25			17396	17555	100.00	0.00	7200	17
B06TW25	16743	**16743**	16531	16743	1.27	100.00	3938	17
B07TW25	16921	**16921**	16725	16921	1.16	100.00	5808	35
B08TW25	18508	**18508**	18313	18508	1.05	100.00	4495	49
B09TW25	18933	**18933**	18821	18933	0.59	100.00	328	3
B10TW25	16382	**16382**	16334	16382	0.29	100.00	748	5
B11TW25	17014	**17014**	16941	17014	0.43	100.00	571	7
B12TW25	18028	**18028**	17939	18028	0.49	100.00	2922	13
B13TW25	16332	**16332**	16255	16332	0.47	100.00	1424	11
B14TW25	16721	**16721**	16673	16721	0.29	100.00	1170	15
B15TW25	18031		17758	17885	1.51	46.52	7200	41
B16TW25	16596	**16596**	16496	16596	0.60	100.00	6030	33
B17TW25	15587	**15587**	15454	15587	0.85	100.00	6348	37
B18TW25	17685	**17685**	17685	17685	0.00	100.00	169	1
B19TW25	15804	**15804**	15684	15804	0.76	100.00	6292	39
B20TW25	19398	**19398**	19238	19398	0.82	100.00	3447	29
All					5.76	90.54	3594.40	22.70

Table 3.14: Results of the Branch-and-Price Algorithm on 76-Task Instances with Time Window Flexibility 25

Instance	Best	UB	LB		Gap [%]		Time	# Nodes
	known	**(Opt)**	Root	Tree	Root	closed	[sec]	solved
B01TW50	16106		15676	15872	2.67	45.58	7200	45
B02TW50	16838	**16838**	16690	16838	0.88	100.00	4161	23
B03TW50	15170		14597	14798	3.78	35.08	7200	28
B04TW50	17564	**17564**	17477	17564	0.50	100.00	2083	17
B05TW50			17096	17301	100.00	0.00	7200	17
B06TW50	16743		16321	16473	2.52	36.02	7200	30
B07TW50	16596	**16596**	16423	16596	1.04	100.00	6512	33
B08TW50	18139		17900	18090	1.32	79.50	7200	46
B09TW50	18667	**18667**	18577	18667	0.48	100.00	1442	9
B10TW50	16330	**16330**	16198	16330	0.81	100.00	3063	17
B11TW50	16963		16711	16893	1.49	72.22	7200	33
B12TW50	17952		17780	17891	0.96	64.53	7200	30
B13TW50	16112	**16112**	15998	16112	0.71	100.00	3942	17
B14TW50	16519	**16519**	16473	16519	0.28	100.00	439	5
B15TW50	18031		17283	17442	4.15	21.26	7200	31
B16TW50	16318	16337	16193	16274	0.77	64.80	7200	28
B17TW50	15587		14916	15094	4.30	26.53	7200	29
B18TW50	17509	**17509**	17451	17509	0.33	100.00	282	3
B19TW50	15804		15565	15681	1.51	48.54	7200	32
B20TW50	19173		18987	19108	0.97	65.05	7200	45
All					6.47	67.96	5416.20	25.90

Table 3.15: Results of the Branch-and-Price Algorithm on 76-Task Instances with Time Window Flexibility 50

Instance	Best	UB	LB		Gap [%]		Time	# Nodes
	known	**(Opt)**	Root	Tree	Root	closed	[sec]	solved
B01TW100	16106		15294	15450	5.04	19.21	7200	22
B02TW100	16838		16253	16384	3.47	22.39	7200	21
B03TW100	15170		14298	14452	5.75	17.66	7200	24
B04TW100	17383		17083	17245	1.73	54.00	7200	29
B05TW100			16775	16942	100.00	0.00	7200	24
B06TW100	16743		15966	16103	4.64	17.63	7200	26
B07TW100	16596		16176	16286	2.53	26.19	7200	25
B08TW100	18139		17226	17356	5.03	14.24	7200	17
B09TW100	18667		18263	18496	2.16	57.67	7200	43
B10TW100	16330		15760	15910	3.49	26.32	7200	27
B11TW100	16963		16425	16573	3.17	27.51	7200	30
B12TW100	17952		17095	17243	4.77	17.27	7200	31
B13TW100	16112		15549	15680	3.49	23.27	7200	22
B14TW100	16250	**16250**	16170	16250	0.49	100.00	2814	11
B15TW100	18031		16779	16913	6.94	10.70	7200	22
B16TW100	15866	**15866**	15792	15866	0.47	100.00	5161	15
B17TW100	15587		14746	14887	5.40	16.77	7200	25
B18TW100	17366	17366	17220	17356	0.84	93.15	7200	45
B19TW100	15804		15223	15383	3.68	27.54	7200	29
B20TW100	19173		18426	18506	3.90	10.71	7200	22
All					8.35	34.11	6878.75	25.50

Table 3.16: Results of the Branch-and-Price Algorithm on 76-Task Instances with Time Window Flexibility 100

Instance	Best	UB	LB		Gap [%]		Time	# Nodes
	known	**(Opt)**	Root	Tree	Root	closed	[sec]	solved
B01TW200	16106		14738	14809	8.49	5.19	7200	9
B02TW200	16838		15693	15735	6.80	3.67	7200	8
B03TW200	15170		13579	13641	10.49	3.90	7200	8
B04TW200	17383		16028	16132	7.79	7.68	7200	15
B05TW200			15613	15684	100.00	0.00	7200	12
B06TW200	16743		15578	15653	6.96	6.44	7200	11
B07TW200	16596		15469	15539	6.79	6.21	7200	14
B08TW200	18139		16735	16795	7.74	4.27	7200	6
B09TW200	18667		17162	17393	8.06	15.35	7200	19
B10TW200	16330		14314	14350	12.35	1.79	7200	5
B11TW200	16963		15584	15677	8.13	6.74	7200	12
B12TW200	17952		15958	16006	11.11	2.41	7200	8
B13TW200	16112		15059	15161	6.54	9.69	7200	15
B14TW200	16250		15216	15299	6.36	8.03	7200	11
B15TW200	18031		15603	15690	13.47	3.58	7200	15
B16TW200	15866		14813	14881	6.64	6.46	7200	10
B17TW200	15587		14235	14288	8.67	3.92	7200	13
B18TW200	17366		16307	16468	6.10	15.20	7200	18
B19TW200	15804		14526	14599	8.09	5.71	7200	15
B20TW200	19173		17160	17345	10.50	9.19	7200	8
All					13.05	6.27	7200.00	11.60

Table 3.17: Results of the Branch-and-Price Algorithm on 76-Task Instances with Time Window Flexibility 200

Instance	Best	UB	LB		Gap [%]		Time	# Nodes
	known	**(Opt)**	Root	Tree	Root	closed	[sec]	solved
A01	13883		12744	12775	8.20	0.03	7200	3
A02	15030		12941	13006	13.90	0.03	7200	4
A04	15541		14011	14028	9.84	0.01	7200	5
A05	16576		15328	15442	7.53	0.09	7200	16
A06	10358	**10.358**	10358	10358	0.00	100.00	1858	1
A07	13848		12901	13052	6.84	0.16	7200	11
A08	15841		14932	15127	5.74	0.21	7200	20
A09	14872		13542	13654	8.94	0.08	7200	7
A10	16415		15646	15685	4.68	0.05	7200	3
A11	16342						7200	0
A12	14676		13761	13901	6.23	0.15	7200	11
A13	16160		14620	14898	9.53	0.18	7200	20
A14	15432		14201	14370	7.98	0.14	7200	4
A16	16100		15150	15150	5.90	0.00	7200	1
A17	15963		14932	14949	6.46	0.02	7200	4
A18	12233		11722	11722	4.18	0.00	7200	1
A19	17114		14908	14908	12.89	0.00	7200	2
A20	15896		14537	14604	8.55	0.05	7200	9
A21	14742		13732	13943	6.85	0.21	7200	31
A22	15269		13886	13972	9.06	0.06	7200	10
A23	16565		15600	15675	5.83	0.08	7200	11
A24	16625		15596	15663	6.19	0.07	7200	9
A25	18913		17864	17881	5.55	0.02	7200	4
A26	15178		13911	14074	8.35	0.13	7200	16
A27	14939		13621	13652	8.82	0.02	7200	6
A28	13822		13343	13518	3.47	0.37	7200	22
A29	15501		13799	13985	10.98	0.11	7200	11
A30	15907		14387	14616	9.56	0.15	7200	24
All					7.48	3.79	7009	10

Table 3.18: Results of the Branch-and-Price Algorithm on MK Instances

Instance	Best	UB	LB			Gap[%]			Time	# Nodes
		(Opt)	Root	Cut	Tree	Root	clo. Cut	clo. Tree	[sec]	solved
A01TW25	7721	**7721**	7721	7721	7721	0.00	100.00	100.00	37	1
A02TW25	7577	**7577**	7544	7559	7577	0.44	45.45	100.00	186	3
A03TW25	8198	**8198**	8198	8198	8198	0.00	100.00	100.00	52	1
A04TW25	8114	**8114**	8096	8114	8114	0.22	100.00	100.00	48	5
A05TW25	8861	**8861**	8850	8860	8861	0.12	90.91	100.00	122	3
A06TW25	7233	**7233**	7233	7233	7233	0.00	100.00	100.00	57	1
A07TW25	7767	**7767**	7767	7767	7767	0.00	100.00	100.00	45	1
A08TW25	8168	**8168**	8168	8168	8168	0.00	100.00	100.00	88	1
A09TW25	7653	**7653**	7590	7607	7653	0.82	26.98	100.00	1696	7
A10TW25	9583	**9583**	9425	9462	9583	1.65	23.42	100.00	325	9
A11TW25	9237	**9237**	9225	9237	9237	0.13	100.00	100.00	65	3
A12TW25	7720	**7720**	7720	7720	7720	0.00	100.00	100.00	48	1
A13TW25	8791	**8791**	8742	8775	8791	0.56	67.35	100.00	132	7
A14TW25	7632	**7632**	7600	7601	7632	0.42	3.13	100.00	171	7
A15TW25	7539	**7539**	7521	7522	7539	0.24	5.56	100.00	438	7
A16TW25	7814	**7814**	7703	7729	7814	1.42	23.42	100.00	1263	23
A17TW25	8603	**8603**	8502	8533	8603	1.17	30.69	100.00	219	15
A18TW25	6943	**6943**	6943	6943	6943	0.00	100.00	100.00	42	1
A19TW25	9231	**9231**	9106	9149	9231	1.35	34.40	100.00	200	11
A20TW25	8665	**8665**	8665	8665	8665	0.00	100.00	100.00	47	1
All						0.43	67.57	100.00	264	5

Table 3.19: Results of the Branch-and-Price-and-Cut Algorithm on 38-Task Instances with Time Window Flexibility 25

Instance	Best	UB	LB			Gap[%]			Time	# Nodes
		(Opt)	Root	Cut	Tree	Root	clo. Cut	clo. Tree	[sec]	solved
A01TW50	7671	**7671**	7648	7668	7671	0.30	86.96	100.00	148	5
A02TW50	7527	**7527**	7475	7495	7527	0.69	38.46	100.00	230	13
A03TW50	8165	**8165**	8165	8165	8165	0.00	100.00	100.00	54	1
A04TW50	7972	**7972**	7972	7972	7972	0.00	100.00	100.00	63	1
A05TW50	8662	**8662**	8662	8662	8662	0.00	100.00	100.00	61	1
A06TW50	7183	**7183**	7183	7183	7183	0.00	100.00	100.00	69	1
A07TW50	7742	**7742**	7654	7710	7742	1.14	63.64	100.00	310	11
A08TW50	8131	**8131**	8104	8131	8131	0.33	100.00	100.00	115	3
A09TW50	7448	**7448**	7448	7448	7448	0.00	100.00	100.00	76	1
A10TW50	9489	**9489**	9313	9339	9489	1.85	14.77	100.00	777	23
A11TW50	9124	**9124**	9084	9085	9124	0.44	2.50	100.00	206	7
A12TW50	7670	**7670**	7579	7632	7670	1.19	58.24	100.00	180	17
A13TW50	8741	**8741**	8593	8612	8741	1.69	12.84	100.00	1210	17
A14TW50	7453	**7453**	7453	7453	7453	0.00	100.00	100.00	49	1
A15TW50	7454	**7454**	7431	7436	7454	0.31	21.74	100.00	138	5
A16TW50	7520	**7520**	7511	7511	7520	0.12	0.00	100.00	185	5
A17TW50	8499	**8499**	8432	8434	8499	0.79	2.99	100.00	219	5
A18TW50	6898	**6898**	6898	6898	6898	0.00	100.00	100.00	61	1
A19TW50	9119	**9119**	8927	8960	9119	2.11	17.19	100.00	1104	47
A20TW50	8633	**8633**	8633	8633	8633	0.00	100.00	100.00	70	1
All						0.55	60.97	100.00	266	8

Table 3.20: Results of the Branch-and-Price-and-Cut Algorithm on 38-Task Instances with Time Window Flexibility 50

Instance	Best	UB	LB			Gap[%]			Time	# Nodes
		(Opt)	Root	Cut	Tree	Root	clo. Cut	clo. Tree	[sec]	solved
A01TW100	7562	**7562**	7509	7519	7562	0.70	18.87	100.00	1160	17
A02TW100	7477		7253	7312	7402	3.00	26.34	66.52	7200	56
A03TW100	8140	**8140**	8140	8140	8140	0.00	100.00	100.00	26	1
A04TW100	7852	**7852**	7841	7846	7852	0.14	45.45	100.00	473	3
A05TW100	8537	**8537**	8489	8495	8537	0.56	12.50	100.00	201	5
A06TW100	7122	**7122**	7117	7117	7122	0.07	0.00	100.00	233	7
A07TW100	7448	**7448**	7385	7395	7448	0.85	15.87	100.00	590	15
A08TW100	8050	**8050**	8036	8049	8050	0.17	92.86	100.00	256	5
A09TW100	7287	**7287**	7287	7287	7287	0.00	100.00	100.00	77	1
A10TW100	9054	**9054**	8996	9025	9054	0.64	50.00	100.00	83	7
A11TW100	8993	**8993**	8924	8941	8993	0.77	24.64	100.00	1053	13
A12TW100	7626	**7626**	7516	7535	7626	1.44	17.27	100.00	1561	43
A13TW100	8533	**8533**	8385	8413	8533	1.73	18.92	100.00	1400	25
A14TW100	7400	**7400**	7260	7272	7400	1.89	8.57	100.00	3589	27
A15TW100	7368	**7368**	7312	7319	7368	0.76	12.50	100.00	1996	33
A16TW100	7354	**7354**	7318	7331	7354	0.49	36.11	100.00	440	3
A17TW100	8335	**8335**	8201	8231	8335	1.61	22.39	100.00	1499	19
A18TW100	6812	**6812**	6696	6715	6812	1.70	16.38	100.00	4249	37
A19TW100	8418	**8418**	8373	8373	8418	0.53	0.00	100.00	273	7
A20TW100	8490	**8490**	8262	8288	8490	2.69	11.40	100.00	1604	15
All						0.99	31.50	98.33	1398	17

Table 3.21: Results of the Branch-and-Price-and-Cut Algorithm on 38-Task Instances with Time Window Flexibility 100

Instance	Best	UB	LB			Gap[%]			Time	# Nodes
		(Opt)	Root	Cut	Tree	Root	clo. Cut	clo. Tree	[sec]	solved
A01TW200	7562		6814	6835	6868	9.89	2.81	7.22	7200	16
A02TW200	7063		6976	6986	7051	1.23	11.49	86.21	7200	21
A03TW200	7565	**7565**	7555	7565	7565	0.13	100.00	100.00	68	3
A04TW200	7534	**7534**	7441	7448	7534	1.23	7.53	100.00	6556	25
A05TW200	7851	**7851**	7787	7796	7851	0.82	14.06	100.00	869	5
A06TW200	6389	**6389**	6295	6354	6389	1.47	62.77	100.00	812	13
A07TW200	7448		6741	6850	6900	9.49	15.42	22.49	7200	59
A08TW200	7759	**7759**	7750	7759	7759	0.12	100.00	100.00	120	3
A09TW200	7158	**7158**	7154	7158	7158	0.06	100.00	100.00	125	3
A10TW200	8141	**8141**	8141	8141	8141	0.00	100.00	100.00	84	1
A11TW200	8472	**8472**	8384	8387	8472	1.04	3.41	100.00	5370	17
A12TW200	7495	**7495**	7414	7422	7495	1.08	9.88	100.00	3127	33
A13TW200	8114	**8114**	7880	7927	8114	2.88	20.09	100.00	4388	106
A14TW200	7154	**7154**	6996	7003	7154	2.21	4.43	100.00	7000	33
A15TW200	6804	**6804**	6770	6783	6804	0.50	38.24	100.00	3375	11
A16TW200	7354		6880	6933	7028	6.45	11.18	31.22	7200	74
A17TW200	7829	**7829**	7759	7767	7829	100.00	11.43	100.00	1558	19
A18TW200	6812		6398	6416	6432	100.00	4.35	8.21	7200	8
A19TW200	8418		7877	7904	7984	100.00	4.99	19.78	7200	59
A20TW200	7842	**7842**	7728	7737	7842	100.00	7.89	100.00	1272	15
All						21.93	31.50	78.76	3896	26

Table 3.22: Results of the Branch-and-Price-and-Cut Algorithm on 38-Task Instances with Time Window Flexibility 200

Instance	Best	UB	LB			Gap[%]			Time	# Nodes
		(Opt)	Root	Cut	Tree	Root	clo. Cut	clo. Tree	[sec]	solved
B01TW25	16106	**16106**	15992	16061	16106	0.71	60.53	100.00	589	6
B02TW25	17617	**17617**	17350	17388	17617	1.52	14.23	100.00	1012	12
B03TW25	15170		14808	14857	15054	2.39	13.54	67.96	7200	44
B04TW25	18085	**18085**	18085	18085	18085	0.00	100.00	100.00	127	1
B05TW25			17396	17427	17546	100.00	0.00	0.00	7200	35
B06TW25	16743	**16743**	16531	16533	16743	1.27	0.94	100.00	3629	18
B07TW25	16921	**16921**	16725	16815	16921	1.16	45.92	100.00	2350	18
B08TW25	18508	**18508**	18313	18313	18508	1.05	0.00	100.00	6875	64
B09TW25	18933	**18933**	18821	18821	18933	0.59	0.00	100.00	307	3
B10TW25	16382	**16382**	16334	16337	16382	0.29	6.25	100.00	457	4
B11TW25	17014	**17014**	16941	16959	17014	0.43	24.66	100.00	527	8
B12TW25	18028	**18028**	17939	17964	18028	0.49	28.09	100.00	1170	10
B13TW25	16332	**16332**	16255	16301	16332	0.47	59.74	100.00	564	6
B14TW25	16721	**16721**	16673	16711	16721	0.29	79.17	100.00	332	4
B15TW25	18031		17758	17850	17981	1.51	33.70	81.68	7200	46
B16TW25	16596	**16596**	16496	16509	16596	0.60	13.00	100.00	3646	24
B17TW25	15587	**15587**	15454	15579	15587	0.85	93.98	100.00	466	6
B18TW25	17685	**17685**	17685	17685	17685	0.00	100.00	100.00	101	1
B19TW25	15804	**15804**	15684	15731	15804	0.76	39.17	100.00	2235	16
B20TW25	19398	**19398**	19238	19246	19398	0.82	5.00	100.00	5016	30
All						5.76	35.90	92.48	2550	18

Table 3.23: Results of the Branch-and-Price-and-Cut Algorithm on 76-Task Instances with Time Window Flexibility 25

Instance	Best	UB	LB			Gap[%]			Time	# Nodes
		(Opt)	Root	Cut	Tree	Root	clo. Cut	clo. Tree	[sec]	solved
B01TW50	16106		15676	15735	15872	2.67	13.72	45.58	7200	31
B02TW50	16838		16690	16695	16837	0.88	3.38	99.32	7200	39
B03TW50	15170		14597	14667	14867	3.78	12.22	47.12	7200	30
B04TW50	17564	**17564**	17477	17502	17564	0.50	28.74	100.00	1167	12
B05TW50			17096	17198	17331	100.00	0.00	0.00	7200	26
B06TW50	16743		16321	16409	16583	2.52	20.85	62.09	7200	27
B07TW50	16596	**16596**	16423	16501	16596	1.04	45.09	100.00	1468	12
B08TW50	18139		17900	17931	18115	1.32	12.97	89.96	7200	43
B09TW50	18667	**18667**	18577	18624	18667	0.48	52.22	100.00	1162	8
B10TW50	16330	**16330**	16198	16209	16330	0.81	8.33	100.00	4305	22
B11TW50	16963	16963	16711	16740	16957	1.49	11.51	97.62	7200	55
B12TW50	17952		17780	17809	17925	0.96	16.86	84.30	7200	28
B13TW50	16112	**16112**	15998	16022	16112	0.71	21.05	100.00	4441	20
B14TW50	16519	**16519**	16473	16500	16519	0.28	58.70	100.00	1470	12
B15TW50	18031		17283	17403	17501	4.15	16.04	29.14	7200	28
B16TW50	16318	16337	16193	16210	16279	0.77	13.60	68.80	7200	25
B17TW50	15587		14916	14944	15064	4.30	4.17	22.06	7200	25
B18TW50	17509	**17509**	17451	17480	17509	0.33	50.00	100.00	289	4
B19TW50	15804		15565	15601	15706	1.51	15.06	59.00	7200	38
B20TW50	19173	19222	18987	18993	19099	0.97	3.23	60.22	7200	49
All						6.47	20.39	73.26	5395	27

Table 3.24: Results of the Branch-and-Price-and-Cut Algorithm on 76-Task Instances with Time Window Flexibility 50

Instance	Best	UB	LB			Gap[%]			Time	# Nodes
		(Opt)	Root	Cut	Tree	Root	clo. Cut	clo. Tree	[sec]	solved
B01TW100	16106		15294	15382	15481	5.04	10.84	23.03	7200	10
B02TW100	16838		16253	16259	16419	3.47	1.03	28.38	7200	27
B03TW100	15170		14298	14350	14495	5.75	5.96	22.59	7200	16
B04TW100	17383		17083	17115	17278	1.73	10.67	65.00	7200	38
B05TW100			16775	16832	16931	100.00	0.00	0.00	7200	17
B06TW100	16743		15966	16052	16126	4.64	11.07	20.59	7200	16
B07TW100	16596		16176	16204	16301	2.53	6.67	29.76	7200	22
B08TW100	18139		17226	17333	17376	5.03	11.72	16.43	7200	11
B09TW100	18667		18263	18292	18457	2.16	7.18	48.02	7200	37
B10TW100	16330		15760	15826	15920	3.49	11.58	28.07	7200	12
B11TW100	16963		16425	16456	16590	3.17	5.76	30.67	7200	23
B12TW100	17952		17095	17120	17236	4.77	2.92	16.45	7200	25
B13TW100	16112		15549	15590	15681	3.49	7.28	23.45	7200	15
B14TW100	16250	**16250**	16170	16232	16250	0.49	77.50	100.00	1118	8
B15TW100	18031		16779	16823	16899	6.94	3.51	9.58	7200	7
B16TW100	15866	**15866**	15792	15820	15866	0.47	37.84	100.00	2845	10
B17TW100	15587		14746	14767	14885	5.40	2.50	16.53	7200	18
B18TW100	17366	17382	17220	17231	17361	0.84	7.53	96.58	7200	40
B19TW100	15804		15223	15264	15371	3.68	7.06	25.47	7200	23
B20TW100	19173		18426	18458	18559	3.90	4.28	17.80	7200	24
All						8.35	11.64	35.92	6678	20

Table 3.25: Results of the Branch-and-Price-and-Cut Algorithm on 76-Task Instances with Time Window Flexibility 100

Instance	Best	UB	LB			Gap[%]			Time	# Nodes
		(Opt)	Root	Cut	Tree	Root	clo. Cut	clo. Tree	[sec]	solved
B01TW200	16106		14738	14782	14826	8.49	3.22	6.43	7200	6
B02TW200	16838		15693	15731	15753	6.80	3.32	5.24	7200	9
B03TW200	15170		13579	13620	13659	10.49	2.58	5.03	7200	6
B04TW200	17383		16028	16037	16106	7.79	0.66	5.76	7200	6
B05TW200			15613	15638	15694	100.00	0.00	0.00	7200	7
B06TW200	16743		15578	15626	15663	6.96	4.12	7.30	7200	6
B07TW200	16596		15469	15504	15531	6.79	3.11	5.50	7200	8
B08TW200	18139		16735	16795	16829	7.74	4.27	6.70	7200	7
B09TW200	18667		17162	17196	17262	8.06	2.26	6.64	7200	11
B10TW200	16330		14314	14345	14384	12.35	1.54	3.47	7200	4
B11TW200	16963		15584	15622	15695	8.13	2.76	8.05	7200	7
B12TW200	17952		15958	16040	16040	11.11	4.11	4.11	7200	3
B13TW200	16112		15059	15081	15138	6.54	2.09	7.50	7200	11
B14TW200	16250		15216	15236	15278	6.36	1.93	6.00	7200	9
B15TW200	18031		15603	15636	15683	13.47	1.36	3.29	7200	9
B16TW200	15866		14813	14838	14882	6.64	2.37	6.55	7200	4
B17TW200	15587		14235	14261	14290	8.67	1.92	4.07	7200	8
B18TW200	17366		16307	16353	16418	6.10	4.34	10.48	7200	9
B19TW200	15804		14526	14552	14594	8.09	2.03	5.32	7200	8
B20TW200	19173		17160	17292	17341	10.50	6.56	8.99	7200	17
All						13.05	2.73	5.82	7200	8

Table 3.26: Results of the Branch-and-Price-and-Cut Algorithm on 76-Task Instances with Time Window Flexibility 200

Instance	Best	UB	LB			Gap[%]			Time	# Nodes
		(Opt)	Root	Cut	Tree	Root	clo. Cut	clo. Tree	[sec]	solved
A01	13883		12744	12744	12744	8.21	0.00	0.00	7200	1
A02	15030		12941	12941	12941	13.90	0.00	0.00	7200	1
A04	15541		14011	14011	14011	9.85	0.00	0.00	7200	1
A05	16576		15327	15352	15352	7.53	0.02	0.02	7200	2
A06	10358	**10358**	10358	10358	10358	0.00	100.00	100.00	1645	1
A07	13848		12901	12930	12930	6.84	0.03	0.03	7200	2
A08	15841		14932	15003	15003	5.74	0.08	0.08	7200	3
A09	14872		13542	13629	13629	8.94	0.07	0.07	7200	2
A10	16415		15646	15646	15646	4.69	0.00	0.00	7200	1
A11	16342								7200	0
A12	14676		13761	13828	13860	6.24	0.07	0.11	7200	4
A13	16160		14619	14652	14707	9.53	0.02	0.06	7200	6
A14	15432		14200	14200	14200	7.98	0.00	0.00	7200	1
A16	16100		15149	15149	15149	5.91	0.00	0.00	7200	1
A17	15963		14932	14932	14932	6.46	0.00	0.00	7200	1
A18	12233		11722	11722	11722	4.18	0.00	0.00	7200	1
A19	17114		14907	14907	14907	12.89	0.00	0.00	7200	1
A20	15896		14537	14537	14537	8.55	0.00	0.00	7200	1
A21	14742		13732	13776	13809	6.85	0.04	0.08	7200	4
A22	15269		13886	13929	13929	9.06	0.03	0.03	7200	2
A23	16565		15599	15599	15599	5.83	0.00	0.00	7200	1
A24	16625		15596	15650	15650	6.19	0.05	0.05	7200	2
A25	18913		17863	17863	17863	5.55	0.00	0.00	7200	1
A26	15178		13910	13960	13960	8.35	0.04	0.04	7200	2
A27	14939		13620	13620	13620	8.83	0.00	0.00	7200	1
A28	13822		13343	13384	13384	3.47	0.09	0.09	7200	2
A29	15501		13798	13798	13798	10.99	0.00	0.00	7200	1
A30	15907		14387	14412	14469	9.56	0.02	0.05	7200	5
All						7.49	3.72	3.73	7002	2

Table 3.27: Results of the Branch-and-Price-and-Cut Algorithm on MK Instances

Chapter 4

Branch-and-Price-and-Cut for the Vehicle Routing and Truck Driver Scheduling Problem

Christian Tilk

Abstract

Many governments worldwide have imposed hours of service regulations for truck drivers to ensure that break and rest periods are regularly taken. Transport companies have to take these into account and plan the routes and schedules of their truck drivers simultaneously. This problem is called vehicle routing and truck driver scheduling problem (VRTDSP). With their paper "An exact method for vehicle routing and truck driver scheduling problems" [Transportation Science, 2016, `http://dx.doi.org/10.1287/trsc.2016.0678`] Goel and Irnich presented the first exact approach to the VRTDSP. They include hours of service regulations in a vehicle routing problem with time windows and use a branch-and-price algorithm to solve it. The main contribution of the paper at hand is to present a sophisticated branch-and-price-and-cut algorithm for the VRTDSP that is based on the parameter-free auxiliary network and the resource extension functions (REFs) defined in the work of Goel and Irnich. Their labeling algorithm is extended by means of defining backward REFs in order to build a bidirectional labeling. Feasible routes are constructed by a non-trivial merge procedure. Different acceleration techniques are used to speed up the solution process of the pricing problem. In addition, several classes of known valid inequalities are used to further strengthen the LP-relaxation of the master program. We present a detailed computational study to analyze the impact of the different techniques. The resulting algorithm is able to solve all VRTDSP benchmark instances with 25 customers and 44 out of 56 instances with 50 customers in two hours of computation time to proven optimality.

4.1 Introduction

Many governments worldwide have imposed hours of service regulations for truck drivers to avoid fatigue-related accidents. These regulations ensure that break and rest periods are regularly taken, i.e., they define a minimum amount of break and rest times for truck drivers as well as a maximum driving time between two break or rest periods. Transport

companies have to take these into account when planning the routes and schedules of their truck drivers. All kinds of such regulations strongly restrict the feasibility of vehicle routes that last several days and are assigned to a single driver. Two approaches can be used to construct feasible vehicle routes while minimizing overall costs: In a two-phase approach, routes are first built without considering hours of service regulations and later modified to take them into account. Solving the problem exactly makes it necessary to use an integrated approach: Vehicle routes and truck driver schedules have to be planned simultaneously, which is more complex but usually provides lower-cost solutions. This problem is called *vehicle routing and truck driver scheduling problem* (VRTDSP).

The paper at hand deals with the current hours of service regulations of the United States (U.S.) for property-carrying drivers, that entered into force in 2013 (Federal Motor Carrier Safety Administration, 2011). According to these regulations, the day of a truck driver can be separated into four parts: Driving, working that does not include driving, e.g., loading and unloading the truck, break periods and rest periods. These parts are subject to the following restrictions:

(1) The accumulated driving time between two consecutive rest periods is at most eleven hours.

(2) The minimum duration of a break period is 30 minutes.

(3) The minimum duration of a rest period is ten hours.

(4) Driving can only be done until at most eight hours since the last break or rest period have elapsed.

(5) Driving can only be done until at most 14 hours since the last rest period have elapsed.

Note that these regulations only limit a truck driver in terms of driving, i.e., there are no limitations for working that does not include driving. Recently, Goel and Irnich (2016) presented the first exact approach to the VRTDSP. They include hours of service regulations into a *vehicle routing problem with time windows* (VRPTW) and use a branch-and-price algorithm to solve it. The main contribution of the paper at hand is to present a sophisticated branch-and-price-and-cut algorithm for the VRTDSP that is based on the parameter-free auxiliary network and the resource extension functions (REFs, see Irnich, 2008) defined in (Goel and Irnich, 2016). We extend their labeling algorithm by means of defining backward REFs in order to build a bidirectional labeling. To obtain feasible routes a non-trivial merge procedure is presented. The concept of using a dynamic half-way point (Chapter 5) and the *ng*-path relaxation (Baldacci *et al.*, 2011) are used to speed up the solution process of the pricing problem. In addition, different families of known valid inequalities are used to further strengthen the linear relaxation of the master program, namely subset-row (Jepsen *et al.*, 2008), two-path (Kohl *et al.*, 1999) and strong-degree inequalities (Contardo *et al.*, 2014) as well as a dynamic extension of the *ng*-neighborhood (Roberti and Mingozzi, 2014; Bode and Irnich, 2015). We present a detailed computational study to analyze the impact of the different techniques. The resulting algorithm is able to solve to proven optimality all VRTDSP

benchmark instances with 25 customers and 44 out of 56 instances with 50 customers in two hours of computation time.

The remainder of this paper is structured as follows. Section 4.2 briefly reviews the literature on vehicle routing problems with hours of service regulations. In Section 4.3, a set-partitioning formulation is given. Section 4.4 presents the pricing problem and its solution. Families of valid inequalities are discussed in Section 4.5. Section 4.6 presents the computational analysis of the algorithmic components. Concluding remarks are given in Section 4.7.

4.2 Literature

To the best of our knowledge, the first work considering hours of service regulations in a vehicle routing problem is (Savelsbergh and Sol, 1998). The break and rest rules considered in this work were similar to the regulations in the European Union (EU) that were in force at that time. A branch-and-price approach was used to solve the problem heuristically. Xu *et al.* (2003) investigated a pickup-and-delivery problem with several complicating side constraints, including the U.S. hours of service regulations that were in force at that time. Ceselli *et al.* (2009) solved a rich vehicle routing problem with a column-generation algorithm. Among other constraints, they impose an upper limit on the number of consecutive driving hours and enforce drivers' rest periods.

In April 2007, new hours of service regulations were introduced in the EU (European Union, 2006). These regulations are more complex than the current U.S. hours of service regulations. However, they also distinguish between driving, working that do not include driving, break and rest periods. Truck drivers must take a break or rest after at most four and a half hours of driving and a rest after at most nine hours of driving. The duration of a break period is at least 45 minutes and a rest must be an uninterrupted period of at least 11 hours. Additionally, the required rest must be taken within 24 hours after the end of the previous rest period and the accumulated driving time between two rest periods must not exceed nine hours. Furthermore, rest and break periods can be split into two parts. The first part of a break period must take at least 15 minutes and the second part at least 30 minutes, whereas a rest can be split into a first part of at least three hours and a second part of at least nine hours. These rules are only the basic set but guarantee compliance with European law when using a weekly planning horizon. However, there are further rules regulating the weekly and monthly working time or allowing more flexibility, e.g., twice during a calendar week, the daily driving time may be extended up to at most ten hours. These rules are described in detail in (Meyer and Kopfer, 2008) and (Goel, 2010). In Addition, each member state of the EU has additional national regulations.

Kopfer and Meyer (2009) formalized the restrictions on driving times and integrated them in an optimization model for the traveling salesman problem with time windows. Moreover, several heuristic approaches for combined vehicle routing and truck driver scheduling have been proposed for the EU regulations: Zäpfel and Bögl (2008) developed a two-phase algorithm for a VRPTW with certain driver rules. In the first phase, a

metaheuristic is used to solve a vehicle routing problem, while in the second phase a personnel assignment problem is solved with a simple heuristic. Goel (2009) proposed a large neighborhood framework for a VRPTW including only a subset of the EU regulations. Bartodziej *et al.* (2009) designed a column-generation heuristic and three metaheuristics for solving a combined vehicle and crew scheduling problem with time windows and rest regulations. Drexl and Prescott-Gagnon (2010) investigated how European regulations can be formally modeled in an *elementary shortest path problem with resource constraints* (ESPPRC). Kok *et al.* (2010) developed a heuristic dynamic-programming algorithm that can solve a VRPTW with all EU regulations. Prescott-Gagnon *et al.* (2010) also investigated the VRPTW with EU regulations and presented a large neighborhood search method that relies on a column-generation heuristic. Derigs *et al.* (2011) solved a real-world vehicle routing problem respecting the EU regulations.

The literature about the U.S. hours of service regulations is rather scarce. Rancourt *et al.* (2013) developed a tabu search approach to the VRTDSP with respect to the regulations that were in force until July 2013. Rancourt and Paquette (2014) designed a tabu search to solve a multi-objective VRTDSP with respect to the same regulations. Goel (2014) presented a simple heuristic to analyze the impact of the change in the regulations in 2013 in terms of cost and accident risks. Goel and Vidal (2014) presented a hybrid genetic search that has been used for different regulations worldwide, especially also with the new U.S. regulations. To the best of our knowledge, Goel and Irnich (2016) are the only authors presenting an exact solution approach to the VRTDSP complying with the new U.S. regulations. They developed a branch-and-price algorithm for the VRTDSP with the old and new U.S. regulations as well as the regulations in the EU. Solutions of benchmark instances with up to 100 customers are reported.

4.3 Set-Partitioning Formulation

Using column-generation techniques to solve vehicle routing problems exactly has become a standard approach (Desaulniers *et al.*, 2010). In order to solve the VRTDSP with branch-and-price-and-cut, we use a set-partitioning formulation. In this setting, the master problem is quite easy, while capacity and time window constrains as well as hours of service regulations are tackled in the pricing problem (see Section 4.4).

The VRTDSP can be defined as follows: Let $C := \{1, \ldots, n\}$ be the set of customer vertices, o be the start depot and d be the end depot. A time window $[a_i, b_i]$, a non-negative demand q_i, and a non-negative service time s_i are associated with every vertex i. Service times and demands at the start and end depot are defined as zero, i.e., $q_o{=}q_d := 0$ and $s_o = s_d := 0$. Service at a vertex must start inside but may end outside of its time window. Let $A = \{(i, j) \in V \times V : i \neq j\}$ be the set of arcs. Each arc (i, j) is associated with a non-negative travel time t_{ij} (excluding break and rest times) and a non-negative travel cost c_{ij}. Note that we can remove all arcs that can not be feasibly traversed due to time window or capacity restrictions. Furthermore, let K be a fleet of homogeneous vehicles with capacity Q. A vehicle route is defined as a path starting at o, ending at d, and visiting a subset of the customer vertices in between. A route is feasible if the

capacity of the vehicle is not exceeded and if there exists a schedule complying with hours of service regulations and the time windows of the visited vertices. The cost of a route is defined as the sum of the travel cost of the traversed arcs. The goal of the VRTDSP is to find a set of feasible routes visiting each customer exactly once while minimizing the overall travel cost. Assuming that we know the set of all feasible routes R, the VRTDSP can be stated as follows:

$$\min \quad \sum_{r \in R} c_r \lambda_r \tag{4.1a}$$

$$\text{s.t.} \quad \sum_{r \in R} a_{ir} \lambda_r = 1 \qquad \forall i \in C \tag{4.1b}$$

$$\sum_{r \in R} \lambda_r \leq |K| \tag{4.1c}$$

$$\lambda_r \in \{0, 1\} \qquad \forall r \in R \tag{4.1d}$$

The binary variables λ_r indicate if route r is used or not. The coefficients c_r denote the travel cost of route r and coefficients a_{ir} denote the number of times route r visits customer i. Hence, (4.1a) minimizes the overall travel cost. Constraints (4.1b) ensure that each customer is visited exactly once. The number of used vehicles is limited by (4.1c) and the variable domains are given in (4.1d).

In the following, the linear relaxation of Formulation (4.1) in which the set of all feasible routes is additionally replaced by a subset $\bar{R}$ is denoted as the *restricted master program* (RMP). For solving the RMP, a column-generation algorithm (Desaulniers *et al.*, 2005) is employed. The column-generation algorithm alternates between the LP re-optimization of the RMP and the column-generation pricing problem that adds additional variables (=columns) to the RMP. Let π_i be the dual prices of the constraints (4.1b) of the RMP and let μ be the dual price of the aggregated convexity constraint (4.1c). The pricing problem asks for a route r with negative reduced cost $\bar{c}_r := c_r - \mu - \sum_{i \in C} a_{ir} \pi_i$. The reduced cost of an arc $(i, j) \in A$ is defined as $\bar{c}_{ij} := c_{ij} - \frac{1}{2}\pi_i - \frac{1}{2}\pi_j$ with $\pi_o = \pi_d = \mu$. We initialize the set $\bar{R}$ with a set of dummy columns with infinite cost, each visiting one customer and having no impact on constraint (4.1c). Branching on arcs is required to finally ensure integer solutions of (4.1).

In order to stabilize the column-generation process, the partitioning constraints (4.1b) can be replaced by covering constraints if the triangle inequality holds for travel costs and times. This replacement is also possible if the triangle inequality does not hold. Then we have to add to the RMP the additional constraint that the number of visited vertices must not exceed n, i.e., $\sum_{r \in R} l_r \lambda_r \leq n$, where l_r is the number of times route r visits a customer. The effect is that all dual prices π_i for $i \in C$ are non-negative. To incorporate the added constraint in the pricing problem, its dual price has to be subtracted from the reduced cost of all arcs (i, j) with $j \in C$.

4.4 Pricing Problem

In this section, we discuss the pricing problem of the branch-and-price-and-cut algorithm. First, in Subsection 4.4.1, we recapitulate the parameter-free model, the resources, and the REFs developed by Goel and Irnich (2016). This is also the basis for the bidirectional labeling presented in Subsection 4.4.2. Subsection 4.4.3 deals with the ng-path relaxation (Baldacci *et al.*, 2011) for the VRTDSP. Finally, in Subsection 4.4.4, different techniques to accelerate the solution of the pricing problem are presented.

4.4.1 Forward Labeling

The pricing problem is an ESPPRC that can be solved with a labeling algorithm. It tries to find a negative reduced-cost route that is feasible with respect to load, time windows and the hours of service regulations. According to the current U.S. hours of service regulations, a driver may take break or rest periods at any time and with any duration larger than 30 minutes and ten hours, respectively. However, driving periods can be scheduled only depending on the current state of the driver, and no limitations regarding service periods are specified. The relevant parameters are summarized in Table 4.1.

Symbol	Value	Description
t^{drive}	11 hours	The maximum accumulated driving time between two consecutive rest periods
t^{break}	$\frac{1}{2}$ hours	The minimum duration of a break period
t^{rest}	10 hours	The minimum duration of a rest period
$t^{\text{el\|B}}$	8 hours	The maximum time after the end of the last break or rest period until which a driver may drive
$t^{\text{el\|R}}$	14 hours	The maximum time after the end of the last rest period until which a driver may drive

Table 4.1: Parameters Imposed by the New U.S. Hours of Service Regulations (Table 1 in Goel and Irnich, 2016)

Goel and Irnich (2016) defined an auxiliary network to tackle the difficulty of including hours of service regulations in a VRPTW. In this auxiliary network, a driver's working day can be separated into a sequence of four different periods: working, service, break, and rest. We assume that service times can not be interrupted by a break or a rest. Except for this assumption, a driver can take a rest or break at any point in time and space, especially between two customer location. To model this aspect an intermediate vertex n_{ij} is created for each feasible arc $(i, j) \in A$. Now, a driver's state can be modeled with nine resources and his possible activities with five REFs.

Beside the standard VRPTW resources cost, load, and time, the other resources are needed to keep track of the driver's current state. Namely, the accumulated driving time since the last rest, the remaining driving time to reach the next customer, the time elapsed since the last break, and the time elapsed since the last rest. Goel and

Irnich (2016) have shown that it is beneficial in the labeling algorithm to keep rest and break periods as short as possible and to perform driving as long as possible. Obviously, this avoids time periods in which the driver is unproductive, however, this may cause unnecessary waiting times at subsequent customers due to time window restrictions. To overcome this issue, break and rest periods can be scheduled longer than required later on in the labeling algorithm. Therefore, two additional resources are needed to keep track of the latest possible points in time to which the end of the last rest and break period can be extended without violating any time window constraints. In the following, the nine resources needed to represent a driver's state are listed:

- T^{cost}: the accumulated reduced cost of a route
- T^{load}: the accumulated load of a route
- T^{time}: the current time of day
- T^{dist}: the remaining driving time to reach the next customer
- T^{drive}: the accumulated driving time since the last rest
- $T^{el|B}$: the time elapsed since the last break
- $T^{el|R}$: the time elapsed since the last rest
- $T^{la|B}$: The latest possible point in time to which the end of the last break period can be extended without violating any resource constraints
- $T^{la|R}$: The latest possible point in time to which the end of the last rest period can be extended without violating any resource constraints

The possible activities of a driver are represented by the following REFs: First, the REF f_{ij}^{start} models the starting point for going from vertex i to vertex j immediately after i was served. Second, the REF f_{ij}^{drive} is used to model driving from i to j for Δ_{ij} time units. The amount of driving time can be computed as $\Delta_{ij} = \min\{T^{dist}, t^{\text{drive}} - T^{drive}, t^{\text{el|B}} - T^{el|B}, t^{\text{el|R}} - T^{el|R}\}$. Note that Δ_{ij} is the maximal driving time that violates no resource constraints. Third, taking a 30-minute break or a ten-hour rest between the visit of customers i and j is modeled with the REFs f_{ij}^{break} and f_{ij}^{rest}, respectively. Last, performing the service at customer j is modeled with the help of the REF f_{ij}^{visit}. This REF also extends the length of the last rest and break period taken to avoid unnecessary waiting. The subnetwork defined by an arc $(i, j) \in A$ with the corresponding REFs represented as different arcs is depicted in Figure 4.1.

Now, a forward path $P = (o, \ldots, u)$ in the auxiliary network defines a label $L_u := (u, S_u, T_u)$, where u is the last visited vertex, S_u the set of all visited vertices and $T_u := (T_u^{cost}, T_u^{load}, T_u^{time}, T_u^{dist}, T_u^{drive}, T_u^{el|B}, T_u^{el|R}, T_u^{la|B}, T_u^{la|R})$ the resource vector. The initial label representing a fully rested driver at the start depot o is defined as $L_o := (o, \{o\}, T_o)$ with $T_o = (T_o^{cost}, T_o^{load}, T_o^{time}, T_o^{dist}, T_o^{drive}, T_o^{el|B}, T_o^{el|R}, T_o^{la|B}, T_o^{la|R}) := (0, 0, a_o, 0, 0, 0, 0, \infty, \infty)$.

When propagating a label $L_u = (u, S, T_u)$ forward over an arc in the auxiliary network corresponding to a REF f_{ij}^*, some preconditions must be fulfilled to create a feasible new label. The REF f_{ij}^{start} produces a new feasible label if j was not yet visited, the demand of j fits in the vehicle and j can be reached in its time window, i.e., if $j \notin S$, $T^{load} + q_j \leq Q$

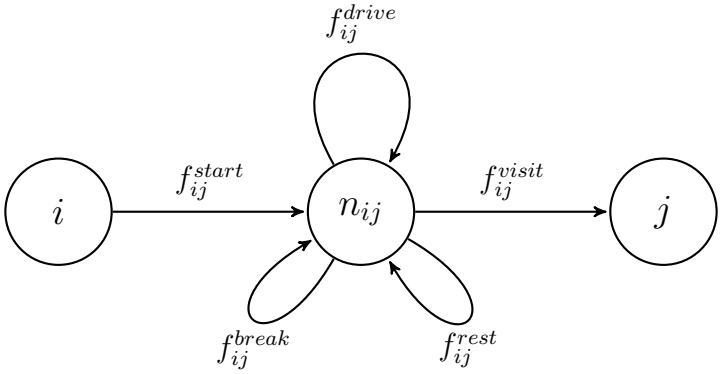

Figure 4.1: Subnetwork for Arc $(i, j) \in A$

and $T^{time} + t_{ij} \leq b_j$, respectively. Driving is only possible if the driver is fit, i.e., f_{ij}^{drive} creates a feasible new label only if $\Delta_{ij} > 0$. Contrariwise, the REFs f_{ij}^{break} and f_{ij}^{rest} return a feasible label only if $\Delta_{ij} \leq 0$. Finally, the service at vertex j is possible only if the full distance between i and j was driven and j is reached before the end of its time window, i.e., f_{ij}^{visit} returns a feasible new label only if $T^{dist} = 0$ and $T^{time} \leq b_j$. Note that f_{ij}^{visit} can be used independently of the value of Δ_{ij}. Thus, the time elapsed since break or rest resources can exceeded $t^{\mathrm{el|B}}$ and $t^{\mathrm{el|R}}$, respectively. As a consequence, Δ_{ij} can have a negative value in the newly created labels.

The resources T^{cost}, T^{load} and the set S are only updated when the REF f_{ij}^{start} is used while their resource values are kept by all other REFs. It holds: $T_{n_{ij}}^{cost} = f_{ij}^{start}(T_i^{cost}) = T_i^{cost} + \bar{c}_{ij}$, $T_{n_{ij}}^{load} = f_{ij}^{start}(T_i^{load}) = T_i^{load} + q_j$, and $S_{n_{ij}} = S_i \cup \{j\}$. All other resource updates of the REFs are summarized in Table 4.2. Blank entries indicate that the resource value is kept.

REF	$\hat{T}^{time}$	$\hat{T}^{dist}$	$\hat{T}^{drive}$	$\hat{T}^{el\|B}$	$\hat{T}^{el\|R}$	$\hat{T}^{la\|B}$	$\hat{T}^{la\|R}$
f_{ij}^{start}		t_{ij}					
f_{ij}^{drive}	$T^{time} + \Delta_{ij}$	$T^{dist} - \Delta_{ij}$	$T^{drive} + \Delta_{ij}$	$T^{el\|B} + \Delta_{ij}$	$T^{el\|R} + \Delta_{ij}$	$\min\{T^{la\|B}, T^{la\|R} + t^{\mathrm{el\|R}} - T^{el\|B} - \Delta_{ij}\}$	
f_{ij}^{break}	$T^{time} + t^{\mathrm{break}}$			0	$T^{el\|R} + t^{\mathrm{break}}$	∞	
f_{ij}^{rest}	$T^{time} + t^{\mathrm{rest}}$		0	0	0	∞	∞
f_{ij}^{visit}	$\max\{T^{time}, a_j\} + s_j$			$\max\{T^{el\|B}, a_j - T^{la\|B}\} + s_j$	$\max\{T^{el\|R}, a_j - T^{la\|R}\} + s_j$	$\min\{T^{la\|B}, b_j - T^{el\|B}\}$	$\min\{T^{la\|R}, b_j - T^{el\|R}\}$

Table 4.2: Resource Extension Functions $f_{ij}(T) = \hat{T}$

Note that all REFs are non-decreasing in the resources $T^{cost}, T^{load}, T^{time}, T^{dist}, T^{drive}$, $T^{el|B}$ and $T^{el|R}$ as well as non-increasing in the resources $T^{la|B}$ and $T^{la|R}$. Hence, a standard dominance rule can be defined (Irnich and Desaulniers, 2005). Additionally,

Goel and Irnich (2016) have developed a second dominance rule to further reduce the number of labels.

Rule 4.1. *(Dominance 1) Let $L = (u, S, T)$ and $L' = (u, S', T')$ be two labels with identical last vertex u. Let $P = P(L)$ and $P' = P(L')$ be the respective partial paths. If $S \subseteq S'$, $T^{cost} \leq T'^{cost}, T^{load} \leq T'^{load}, T^{time} \leq T'^{time}, T^{dist} \leq T'^{dist}, T^{drive} \leq T'^{drive}, T^{el|B} \leq T'^{el|B}, T^{el|R} \leq T'^{el|R}, T^{la|B} \geq T'^{la|B}$ and $T^{la|R} \geq T'^{la|R}$, then any feasible extension of P' towards d is also a feasible extension of P with non-smaller cost. Hence, L' can be discarded.*

Rule 4.2. *(Dominance 2) Let $L = (u, S, T)$ and $L' = (u, S', T')$ be two labels with identical last vertex u. Let $P = P(L)$ and $P' = P(L')$ be the respective partial paths. If $S \subseteq S'$, $T^{cost} \leq T'^{cost}, T^{load} \leq T'^{load}, T^{dist} \leq T'^{dist}$ and $T^{time} + t^{\mathrm{el|R}} \leq T'^{time}$, then any feasible extension of P' towards d is also a feasible extension of P with non-smaller cost. Hence, L' can be discarded.*

4.4.2 Bidirectional Labeling

This subsection presents the bidirectional labeling. First, a backward labeling is defined and the differences to forward labeling are emphasized. Then a non-trivial merge procedure for forward and backward labels is presented.

In order to use the same resources and similar REFs as in the forward labeling, the backward labeling is defined on an inverted network. To be precise, the time window of a vertex i is defined as $[a_i^{bw}, b_i^{bw}] := [-b_i - s_i, -a_i - s_i]$ and the direction of all arcs in the auxiliary backward network is reversed. The time window is shifted by s_i because service at a vertex i must start inside but can end outside the time window. The change in the arc direction leads to a role reversal of the REFs: REF $f_{ij}^{visitBW}$ models the starting point for going backward from vertex j to vertex i and REF $f_{ij}^{startBW}$ models the service at customer i when coming backward from customer j.

Additionally, we have to take care of another aspect of the problem: Since waiting and service times do not count as driving times, performing service is still possible even if the resource $T^{el|R}$ exceeds the maximum value $t^{\mathrm{el|R}}$. In chronological order, this implies that a rest period must succeed the service period that causes this exceeding before any driving period can be executed. In the forward labeling, this is implicitly managed by allowing the use of REF f_{ki}^{visit}, even if the new resource value $T_i^{el|R} = f_{ki}^{visit}(T_{n_{ki}}^{el|R})$ exceeds $t^{\mathrm{el|R}}$ due to the service and waiting times at vertex i. In this case, a rest is implicitly enforced at the next intermediate vertex due to $\Delta \leq 0$.

The same situation means in the reversed chronological order of the backward labeling that the resource $T^{el|R}$ can exceed $t^{\mathrm{el|R}}$ by the service and waiting times at vertex i if and only if REF f_{ij}^{restBW} was applied immediately before REF $f_{ij}^{startBW}$. To model this, two aspects of REF $f_{ij}^{startBW}$ have to be changed. First, when applying REF $f_{ij}^{startBW}$, the service and waiting times at vertex i are not taken into account for computing the new resource values $T_i^{el|R}$ and $T_i^{la|R}$ if and only if REF f_{ij}^{restBW} was applied immediately

before. Note that the resources $T^{el|R}$ and $T^{la|R}$ are both affected since they are interdependent. Second, the resource $T^{el|R}$ must never exceed $t^{\mathrm{el|R}}$ since the service and waiting times that allow an exceeding are not taken into account. Therefore, applying REF $f_{ij}^{startBW}$ returns no feasible new label if the new resource value $T_j^{el|R} = f_{ij}^{startBW}(T_{n_{ij}}^{el|R})$ exceeds $t^{\mathrm{el|R}}$. A similar argument holds for the resource $T^{el|B}$ and the value $t^{\mathrm{el|B}}$.

Hence, the REFs in the backward labeling are defined as follows: The REFs $f_{ij}^{driveBW}$, f_{ij}^{restBW}, $f_{ij}^{breakBW}$ and $f_{ij}^{visitBW}$ use the same preconditions and resource updates as their counterparts f_{ij}^{drive}, f_{ij}^{rest}, f_{ij}^{break} and f_{ij}^{start} in the forward labeling. Besides the preconditions of its counterpart f_{ij}^{visit}, the REF $f_{ij}^{startBW}$ has two additional preconditions to create a feasible label:

$$f_{ij}^{startBW}(T^{el|B}) \leq t^{\mathrm{el|B}} \quad \text{and} \quad f_{ij}^{startBW}(T^{el|R}) \leq t^{\mathrm{el|R}}.$$

Furthermore, $f_{ij}^{startBW}$ updates the set S and the resources T^{cost}, T^{load}, T^{time}, T^{dist} and T^{drive} as the REF f_{ij}^{visit} updates them in the forward labeling. The updates of $T^{el|B}$, $T^{el|R}$, $T^{la|B}$ and $T^{la|R}$ change as stated below. Note that a value $T^{el|R} = 0$ implies that the last used REF was f_{ij}^{restBW}. Similarly, a value $T^{el|B} = 0$ implies that the last used REF was either f_{ij}^{restBW} or $f_{ij}^{breakBW}$.

$$f_{ij}^{startBW}(T^{el|B}) = \begin{cases} 0 & \text{if } T^{el|B} = 0 \\ \max\{T^{el|B}, a_i^{bw} - T^{la|B}\} + s_i & \text{otherwise} \end{cases}$$

$$f_{ij}^{startBW}(T^{el|R}) = \begin{cases} 0 & \text{if } T^{el|R} = 0 \\ \max\{T^{el|R}, a_i^{bw} - T^{la|R}\} + s_i & \text{otherwise} \end{cases}$$

$$f_{ij}^{startBW}(T^{la|B}) = \begin{cases} \infty & \text{if } T^{el|B} = 0 \\ \min\{T^{la|B}, b_i^{bw} - T^{el|B}\} & \text{otherwise} \end{cases}$$

$$f_{ij}^{startBW}(T^{la|R}) = \begin{cases} \infty & \text{if } T^{el|R} = 0 \\ \min\{T^{la|R}, b_i^{bw} - T^{el|R}\} & \text{otherwise} \end{cases}$$

Now, a backward path $P = (u, /dots, d)$ in the auxiliary backward network defines a label $L_u := (u, S_u, T_u)$, where u is the first visited vertex, S_u the set of all visited vertices, and $T_u := (T_u^{cost}, T_u^{load}, T_u^{time}, T_u^{dist}, T_u^{drive}, T_u^{el|B}, T_u^{el|R}, T_u^{la|B}, T_u^{la|R})$ the resource vector. The initial backward label representing a fully rested driver at the end depot d is defined as $L_d := (d, \{d\}, T_d)$ with $T_d = (T_d^{cost}, T_d^{load}, T_d^{time}, T_d^{dist}, T_d^{drive}, T_d^{el|B}, T_d^{el|R}, T_d^{la|B}, T_d^{la|R}) := (0, 0, a_d^{bw}, 0, 0, 0, 0, \infty, \infty)$. Since all backward REFs are non-decreasing in the resources $T^{cost}, T^{load}, T^{time}, T^{dist}, T^{drive}, T^{el|B}$ and $T^{el|R}$ as well as non-increasing in the resources $T^{la|B}$ and $T^{la|R}$, the dominance Rules 4.1 and 4.2 can be applied as in the forward labeling.

Example 4.1. *We give a concrete numerical example to emphasize the difference in forward and backward labeling. A feasible schedule in which $T^{el|B}$ exceeds $t^{\mathrm{el|B}}$ is depicted in Figure 4.2.*

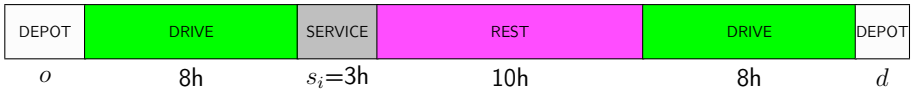

Figure 4.2: A Feasible Schedule Corresponding to a Route *o-i-d*

We assume that the demand of vertex i fits in the vehicle and all time windows are not binding. Hence, no waiting time can occur making the resources $T^{la|B}$ *and* $T^{la|R}$ *obsolete. For the sake of simplicity, we also omit the resources* T^{cost} *and* T^{load}*. The forward and backward path generated by applying the corresponding REF in the auxiliary network is as given in Figure 4.3.*

$$\begin{pmatrix} T_o^{time}=0 \\ T_o^{dist}=0 \\ T_o^{drive}=0 \\ T_o^{el|B}=0 \\ T_o^{el|R}=0 \end{pmatrix} \xrightarrow[t_{oi}=8]{f_{oi}^{start}} \begin{pmatrix} 0 \\ 8 \\ 0 \\ 0 \\ 0 \end{pmatrix} \xrightarrow[\Delta_{oi}=8]{f_{oi}^{drive}} \begin{pmatrix} 8 \\ 0 \\ 8 \\ 8 \\ 8 \end{pmatrix} \xrightarrow[s_i=3]{f_{oi}^{visit}} \begin{pmatrix} 11 \\ 0 \\ 8 \\ 11 \\ 11 \end{pmatrix} \xrightarrow[t_{id}=8]{f_{id}^{start}} \begin{pmatrix} 11 \\ 8 \\ 8 \\ 11 \\ 11 \end{pmatrix} \xrightarrow[\Delta_{id}=-3]{f_{id}^{rest}} \begin{pmatrix} 21 \\ 8 \\ 0 \\ 0 \\ 0 \end{pmatrix} \xrightarrow[\Delta_{id}=8]{f_{id}^{drive}} \begin{pmatrix} 29 \\ 0 \\ 8 \\ 8 \\ 8 \end{pmatrix} \xrightarrow[s_d=0]{f_{id}^{visit}} \begin{pmatrix} 29 \\ 0 \\ 8 \\ 8 \\ 8 \end{pmatrix}$$

$$\begin{pmatrix} -71 \\ 0 \\ 8 \\ 8 \\ 8 \end{pmatrix} \xleftarrow[s_o=0]{f_{oi}^{startBW}} \begin{pmatrix} -71 \\ 0 \\ 8 \\ 8 \\ 8 \end{pmatrix} \xleftarrow[\Delta_{oi}=8]{f_{oi}^{driveBW}} \begin{pmatrix} -79 \\ 8 \\ \not{8}\ 0 \\ \not{8}\ 0 \\ \not{8}\ 0 \end{pmatrix} \xleftarrow[t_{oi}=8]{f_{oi}^{visitBW}} \begin{pmatrix} -79 \\ 0 \\ \not{0}\ 0 \\ \not{0}\ 0 \\ \not{0}\ 0 \end{pmatrix} \xleftarrow[s_i=3]{f_{id}^{startBW}} \begin{pmatrix} -82 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} \xleftarrow[\Delta_{id}=0]{f_{id}^{restBW}} \begin{pmatrix} -92 \\ 0 \\ 8 \\ 8 \\ 8 \end{pmatrix} \xleftarrow[\Delta_{id}=8]{f_{id}^{driveBW}} \begin{pmatrix} -100 \\ 8 \\ 0 \\ 0 \\ 0 \end{pmatrix} \xleftarrow[t_{id}=8]{f_{id}^{visitBW}} \begin{pmatrix} T_d^{time}=-100 \\ T_d^{dist}=0 \\ T_d^{drive}=0 \\ T_d^{el|B}=0 \\ T_d^{el|R}=0 \end{pmatrix}$$

Figure 4.3: Example for propagating labels forward and backward corresponding to the schedule depicted in Figure 4.2

The forward path starts from vertex o, then $t_{oi} = 8$ *hours of driving are allowed due to* Δ_{oi}*. When reaching vertex i, the service can still be performed although* Δ_{id} *gets a negative value. Afterward, a rest is enforced due to* $\Delta_{id} \leq 0$ *complying with the given schedule. Finally, realizing another driving period of eight hours is allowed to reach the depot.*

The backward path starts at vertex d and initially $t_{id} = 8$ *hours of driving are allowed. Subsequently, a rest can be performed due to* $\Delta_{id} = 0$ *complying with the given schedule. The driver can now serve vertex i and here the first difference to the forward REFs occurs: The REF* $f_{id}^{startBW}$ *is used immediately after the REF* f_{id}^{restBW} *was applied indicated by* $T^{el|R} = 0$*. Hence, the service time is not taken into account for the computation of* $T^{el|B}$ *and* $T^{el|R}$*. The crossed-out values show the resource values without that alternation. Afterward, the trip to the start depot o is started, and a driving period of eight hours should be performed according to the schedule. As can be seen from the crossed-out values, it would not be possible to drive eight hours if the service time would have been added to* $T^{el|B}$ *and* $T^{el|R}$*. Hence, without the modification of the REF* $f_{id}^{startBW}$*, the given feasible schedule could not be reproduced in the backward labeling. However, according to the values resulting from the correctly adjusted REF, eight hours of driving are allowed and the depot is reached complying with the given schedule.*

In bidirectional labeling algorithms, forward labels are not necessarily propagated up to the end depot, and backward labels are not necessarily propagated up to the start depot. Instead, labels are propagated only up to a so-called half-way point, thus limiting

the overall number of created labels. Suitable forward and backward labels must then be merged to obtain complete *o*-*d*-paths. As described by Salani (2005), this is done using a half-way point test to avoid creating the same path from different pairs of forward and backward labels.

When using the half-way point $h := (a_o + b_d)/2$, we propagate forward labels at a vertex $i \in V$ in the auxiliary network only if $T_i^{time} \leq h$, and backward labels at a vertex $j \in V$ only if $-T_j^{time} > h$. Labels at an intermediate vertex are always propagated irrespective of their time value. When forward and backward labeling is finished, we merge forward and backward labels at all original vertices $i \in V$. A forward label $L_{fw} = (i, S_{fw}, T_{fw})$ at a vertex $i \in V$ is a candidate for merging if $i = d$ or $T_{fw}^{time} > h$. This condition prevents the creation of identical paths from different pairs of forward and backward labels (Righini and Salani, 2006). We check all backward labels at vertex i whether or not they can be merged with the chosen forward label. A merge of a forward label $L_{fw} = (i, S_{fw}, T_{fw})$ and a backward label $L_{bw} = (i, S_{bw}, T_{bw})$ is feasible if the resulting path fulfills the following conditions:

- the path is elementary: $|S^{fw} \cap S^{bw}| = 1$
- the path respects the vehicle capacity: $T_{fw}^{load} + T_{bw}^{load} - q_i \leq Q$
- the path is feasible with respect to the time resource: $T_{fw}^{time} - s_i \leq -T_{bw}^{time}$
- the accumulated driving time does not exceed the maximum time allowed: $T_{fw}^{drive} + T_{bw}^{drive} \leq t^{\text{drive}}$
- The time between the last break and rest period of the forward and the backward label do not exceed the maximum time allowed: $-T_{bw}^{la|B} - T_{fw}^{la|B} \leq t^{\text{el|B}}$ and $-T_{bw}^{la|R} - T_{fw}^{la|R} \leq t^{\text{el|R}}$
- Either the sum of the elapsed time since break resources respects the maximum time allowed or a rest or break was performed immediately before the visit in one of both labels: $T_{fw}^{el|B} + T_{bw}^{el|B} - s_i \leq t^{\text{el|B}}$ or $T_{fw}^{el|B} \leq s_i$ or $T_{bw}^{el|B} \leq s_i$
- Either the sum of the elapsed time since rest resources respects the maximum time allowed or a rest was performed immediately before the visit in one of both labels: $T_{fw}^{el|R} + T_{bw}^{el|R} - s_i \leq t^{\text{el|R}}$ or $T_{fw}^{el|R} \leq s_i$ or $T_{bw}^{el|R} \leq s_i$

The last two conditions model that exceeding the maximum elapsed time allowed at the merge vertex i is possible if only service and waiting times immediately before a rest cause this exceeding. Note that all backward resources referring to a point in time are multiplied with -1 due to the inversion of the network. Moreover, the demand q_i and the service time s_i at vertex i are subtracted in the capacity and time conditions because they are already taken into account in both labels.

The reduced cost of a merged path can be obtained as $\bar{c}_r = T_{fw}^{cost} + T_{bw}^{cost}$. After the merge, we perform a final dominance test with dominance Rules 4.1 and 4.2 for all routes resulting from merged labels. Finally, we add all undominated negative reduced-cost routes to the RMP.

4.4.3 Elementarity

It is well known that the ESPPRC is NP-hard in the strong sense (Dror, 1994). Hence, in column-generation algorithms for VRPTWs, many authors solve non-elementary SPPRCs as pricing problems (Desaulniers *et al.*, 2014). This can be done in pseudo-polynomial time (Irnich and Desaulniers, 2005). Although this yields weaker lower bounds and routes with cycles must be removed in the branching process, solving only a relaxed pricing problem often pays off with respect to overall computation time. One recent approach that has been very successfully used for different types of vehicle routing problems is the *ng*-path relaxation introduced by Baldacci *et al.* (2011).

We use the *ng*-path relaxation in the VRTDSP as follows: A specific *ng*-path relaxation requires the definition of neighborhoods $N_i \subset C$ with $i \in N_i$ for all vertices $i \in C$. A forward *ng*-path $P = (o, \ldots, i)$ is a non-necessarily elementary path starting at vertex o, ending at vertex i and visiting all vertices within their time window. P defines a forward *ng*-label $L = (i, S_i^{ng}, T_i)$, where i is the last visited vertex and S_i^{ng} is a (generally proper) subset of the visited vertices. The vector T_i contains the same resources as in the elementary formulation. The key point of this relaxations is the update of the set S_i^{ng} via REF f_{ij}^{start}, which differs from the update of the set S_i in the elementary formulation: Forward propagation of a label $L = (i, S_i^{ng}, T_i)$ via f_{ij}^{start}, produces the new label $L = (n_{ij}, S_{n_{ij}}^{ng}, T_{n_{ij}})$, where $T_{n_{ij}} = f_{ij}^{start}(T_i)$ and $S_{n_{ij}}^{ng} = (S_i^{ng} \cup \{j\}) \cap N_j$. The interpretation is that the new label forgets the visited vertices that are not in the set N_j, so that cycles become possible. This change implies that more labels become comparable in the dominance rules, and hence, more labels can be dominated. The definition of a backward *ng*-label and its backward propagation are analogous. For the sake of simplicity, we skip the index *ng* and write S instead of S^{ng} in the following.

The quality of the root lower bound computed by an *ng*-path relaxation strongly depends on the choice of the neighborhoods N_i. We limit the number of neighbors by a constant ν and test different values of ν in our computational experiments. For each vertex $i \in C$, we use an *ng*-neighborhood N_i containing i and the ν closest customers j for which a cycle (j, i, j) would be feasible with respect to time windows and travel and service times. Determining if cycles are possible and defining a good proximity criterion requires the computation of a lower bound for the driving time from i to j that takes the minimum number of rest and break periods between customers i and j into account. Goel and Irnich (2016) have computed the minimum number of mandatory rest and break periods (k_{ij}^{rest} and k_{ij}^{break}) between two customers $i, j \in C$ as

$$k_{ij}^{rest} := \max\left\{0, \left\lceil \frac{t_{ij}}{t^{\text{drive}}} - 1 \right\rceil\right\} \text{ and } k_{ij}^{break} := \left\lceil \frac{\max\{0, t_{ij} - (k_{ij}^{rest}+1)t^{\text{el|B}}\}}{t^{\text{drive}} - t^{\text{el|B}}} \right\rceil.$$

With the help of these two values, a lower bound on the duration of a trip along arc $(i, j) \in A$ can be easily computed as:

$$\hat{t}_{ij} = t_{ij} + k_{ij}^{rest} t^{\text{rest}} + k_{ij}^{break} t^{\text{break}} \tag{4.2}$$

Now, a cycle (j, i, j) may be possible if $\max\{a_i + \hat{t}_{ij}, a_j\} + \hat{t}_{ji} \leq b_i$ and the distance between i and j can be estimated as $\hat{t}_{ij} + \hat{t}_{ji}$.

4.4.4 Acceleration Techniques

The labeling takes by far the largest portion of the time in the overall branch-and-price-and-cut algorithm. To speed up the labeling, four acceleration techniques are used. First, instead of using a static half-way point in bidirectional labeling, a dynamic half-way point is used to balance the time spent in forward and backward labeling (Tilk *et al.*, 2016). Second, an additional set of unreachable customers is defined to strengthen the dominance as proposed by Feillet *et al.* (2004). Third, the labeling is solved heuristically using a *limited discrepancy search* (LDS, see Feillet *et al.*, 2007). Fourth, a heuristic dominance rule is applied to further strengthen the dominance. Goel and Irnich (2016) used the last three techniques in a similar manner.

Dynamic half-way point The forward and backward labeling can take a very different amount of time due to the asymmetry arising from the time windows and dual prices. This also affects the bidirectional labeling because the number of created and dominated labels in the forward and backward part may differ significantly resulting in a huge difference in the computation times of both parts. Chapter 5 introduces a dynamic determination of the half-way point to balance the forward and backward parts of the bidirectional labeling resulting in a reduction of the overall runtime.

We adapt this approach to the VRTDSP as follows: Since the time resource T^{time} is non-decreasing, labels can be propagated in ascending order of the time resource. This enables the use of a dynamic half-way point in bidirectional labeling instead of a static half-way point in the middle of the time horizon. Bidirectional labeling with a dynamic half-way point chooses the next direction to propagate, i.e., forward or backward, dynamically depending on the number of non-processed labels in both parts. Thereby, the number of processed labels in forward and backward labeling is balanced and the dynamic half-way point is implicitly chosen. Let z^{fw} and z^{bw} be the current time resource values of the next label that should be propagated in forward and backward labeling. Note that z^{bw} is negative due to the inversion of the auxiliary network in backward labeling. Now, the idea is to use different dynamic half-way points h_{fw} and h_{bw} in forward and backward labeling that are initialized at the opposite ends of the time horizon and updated depending on z^{bw} and z^{fw}, respectively. More precisely, initially the half-way points are set to $h_{fw} = b_d$ and $h_{bw} = a_o$. Every time a forward label is propagated the backward half-way point is updated to $h_{bw} = \min\{z^{fw}, h_{fw}\}$. Vice versa, when processing a backward label, the forward half-way point is updated to $h_{fw} = \max\{-z^{bw}, h_{bw}\}$. Obviously, as soon as h_{fw} and h_{bw} are equal, both half-way points are fixed and the rest of the labeling continues as the bidirectional labeling with a static half-way point value h_{fw}. The impact of this technique is analyzed in the computational results in Section 4.6.

Unreachable Customers Feillet *et al.* (2004) proposed the use of a set of unreachable customers instead of the set S to strengthen the dominance. Here, we use an additional resource set U in each label to manage the unreachable customers because replacing S by U leads to a problem in the merge step: Given a forward and a backward label with unreachable set U_{fw} and U_{bw}, respectively, a customer i can be unreachable in both labels resulting in $|U_{fw} \cap U_{bw}| > 1$ which violates the merge condition although none of the labels has visited i and the merge could be feasible. Therefore, the merge conditions are still tested with the set S while REF preconditions and the dominance are realized with the set U.

The update of U is done just as the update of S in f_{ij}^{visit} and $f_{ij}^{startBW}$, respectively. Afterwards, unreachable customers are identified and added to U. A customer can become unreachable due to capacity or time window restrictions. While capacity restrictions are easy to check, checking time window restrictions in the VRTDSP is rather complicated due to break and rest periods. However, Goel and Irnich (2016) have proposed a heuristic method to determine an appropriate subset of all unreachable customers using $\hat{t}_{ij}$. Given a label $L = (i, S, T)$ a customer j is unreachable if $T^{load} + q_j > Q$ or $T^{time} + \hat{t}_{ij} > b_j$ where $\hat{t}_{ij}$ is defined in Equation 4.2.

Limited discrepancy search Using heuristic solvers for the pricing problem in branch-and-price algorithms to find negative reduced-cost columns quickly is quite standard. If all heuristic solvers fail in finding a negative reduced-cost column, the exact solver has to be invoked. Feillet *et al.* (2007) have proposed LDS as a heuristic solver. They divide the set of all arcs in good and bad arcs and limit the overall number of bad arcs used in a path by a parameter κ.

We adapt this method to the VRTDSP as follows: In each iteration of the pricing problem, the set of good arcs is newly determined with respect to the current reduced cost of the arcs. All arcs are sorted by increasing reduced cost and scanned in that order. An arc (i, j) is then added to the set of good arcs, if the number of outgoing good arcs from i and ingoing good arcs in j do not exceed five. In addition, all outgoing arcs of the start depot o and all ingoing arcs in the end depot d are added to the set of good arcs. A binary resource n^{bad} is added to each forward and backward label to count the number of bad arcs traversed in the label. n^{bad} is initialized to zero. The REFs f_{ij}^{start} and $f_{ij}^{visitBW}$ increase n^{bad} by one if (i, j) is a bad arc. Afterwards, the REFs check if n^{bad} is greater than κ, and if so, the label is discarded.

We do not include n^{bad} in the dominance and we also do not forbid merging two labels when the sum of their n^{bad} resources exceed κ. The former is done to strengthen the (heuristic) dominance. The latter increases the number of generated paths and thereby raises the probability of finding a negative reduced-cost path while not increasing the computational effort. In our computational experiments, we use several solvers with increasing values of κ, see Section 4.6.

Heuristic dominance In order to further speed up the solution process of the heuristic solvers, we use a heuristic dominance rule similar to the one proposed by Goel and

Irnich (2016). The advantage of a heuristic dominance rule is that more labels are comparable and hence, more labels can be discarded. On the downside, a label may be discarded although it will lead to a pareto-optimal or even a minimum-cost solution. However, if the heuristic pricing solver finds no negative reduced-cost path although one exists, this path will be found by the exact pricing problem solver at the latest. In our dominance rule, two labels are only compared with respect to the set S and the resources T^{cost}, T^{load}, T^{time}, T^{dist} and T^{drive}. The other resources remain disregarded. We combine LDS with the heuristic dominance rule in our computational experiments.

4.5 Valid Inequalities

This section describes the four classes of valid inequalities that are used in our branch-and-price-and-cut algorithm to strengthen the linear relaxation of the master problem. In the computational results, we test different cutting strategies and the combination of these classes of valid inequalities (see Section 4.6).

4.5.1 k-Path Inequalities

The k-path inequalities were introduced by Kohl *et al.* (1999) for the VRPTW. For the special case $k = 2$, we adapt the 2-path inequalities for the VRTDSP as follows: Let $W \subset C$ be a subset of customers that can not be visited by one single vehicle due to capacity or time window restrictions. Moreover, let $\delta^-(W)$ be the set of all arcs $(i, j) \in A$ with $i \in W$ and $j \notin W$. The corresponding 2-path inequality is given by $\sum_{r \in R} \sum_{(i,j) \in \delta^-(W)} b_{ij}^r \lambda_r \geq 2$, where b_{ij}^r is the number of times route r traverses arc $(i, j) \in A$.

Let $\bar{\lambda}_r$ be the value of variable λ_r in the current solution of the RMP. We use the heuristic proposed by Kohl *et al.* (1999) to generate candidate sets W of maximal cardinality with $\sum_{r \in R} \sum_{(i,j) \in \delta^-(W)} b_{ij}^r \bar{\lambda}_r < 2$. Each candidate set is then tested whether or not it has to be served by at least two vehicles. The capacity test is done by simply summing up all demands of vertices in W if it fails at least two vehicles are needed. If the capacity fits, we have to check the time window restrictions. This requires the solution of an ESPPRC in the auxiliary network induced by $W \cup \{o, d\}$ where all arcs have identical reduced cost $\bar{c}_{ij} = -1$. Now, the candidate set can be served by a single vehicle, if there exists a path with reduced cost $\bar{c}_r < -|W|$. This decision problem is very time-consuming for larger sets W, therefore, we limit the maximal cardinality of candidate sets W by a constant parameter w^{max}.

The 2-path inequalities are robust, hence incorporating them needs no adjustments in the structure of the pricing problem. We only need to subtract the current dual value of each 2-path inequality present in the master problems from the reduced cost of the corresponding arcs: Let $\eta_k \geq 0$ be the dual price of the 2-path inequality corresponding to W_k, then η_k is subtracted from the reduced cost of all arcs $(i, j) \in \delta^-(W_k)$.

4.5.2 Subset-Row Inequalities

Subset-row inequalities were first introduced by Jepsen *et al.* (2008) for the VRPTW. They are Chvatal-Gomory rank-1 cuts based on a subset of the constraints in the master program. For the VRTDSP, a subset-row inequality is defined on a subset of customers in the original network. The inequality for a customer set $U_k \subset C$ and a parameter $1 \leq h \leq |U_k|$, in the following denoted by $SR(U_k, h)$, is given by $\sum_{r \in R} \lfloor \frac{\sum_{i \in U_k} a_{ir}}{h} \rfloor \lambda_r \leq \lceil \frac{|U|}{h} \rceil$, where a_{ir} is the number of times route r visits customer i. We restrict ourselves to those inequalities defined for $h = 2$ and $|U_k| = 3$ as proposed by Jepsen *et al.* (2008) because they can be separated by straightforward enumeration. In the following, we write $SR(U_k)$ instead of $SR(U_k, 2)$.

The addition of subset-row inequalities in the master problem requires the following adjustments to our pricing problem: Let $\sigma_k \leq 0$ be the dual price of the subset-row inequality $SR(U_k)$. The value σ_k must be subtracted from the reduced cost for every second visit to vertices in U_k. Therefore, an additional binary resource sr^k, one for each inequality $SR(U_k)$, is necessary in the labeling algorithm for indicating the parity of the number of times a vertex in U_k is visited. Note that the same vertex may be visited more than once in an ng-path relaxation.

Jepsen *et al.* (2008) have proposed a tailored dominance rule that avoids a point-wise comparison of all resources sr^k and thereby reduces the number of incomparable labels significantly. Here, the dominance rules change in the condition regarding the resource T^{cost} as follows: Let $L = (u, S, T)$ and $L' = (u, S', T')$ be two labels with identical last vertex u and let $H := \{k : sr^k_{L'} = 0, sr^k_L = 1\}$ be the index set of the new resources on which L is inferior compared to L'. The condition regarding the resource T^{cost} for label L to dominate label L' changes to $T^{cost} - \sum_{k \in H} \sigma_k \leq T'^{cost}$.

The dominance rules can be strengthened further by taking *unreachable inequalities* into account. A subset-row inequality $SR(U_k)$ is unreachable for a label, if none of the vertices in U_k can be reached due to capacity or time window restrictions. More precisely, $SR(U_k)$ is unreachable for a forward label $L = (u, S, T)$ if $T^{load} > Q - \min_{v \in U_k}\{q_v\}$ or $T^{time} > \max_{v \in U_k}\{b_v - \bar{t}_{uv}\}$. Each unreachable inequality is not taken into account for the dominance procedure, i.e., it is removed from H. A similar rule can be defined for backward labels. We can compute the values at the right-hand side of the two inequalities above in a preprocessing step. Note that the computation is not exact, i.e., the two inequalities provide a necessary but generally not sufficient condition for a subset-row inequality to be unreachable. An exact computation would require the consideration of break and rest times for each label individually which would be too time-consuming.

In addition, the computation of the reduced cost of a merged path in the bidirectional labeling algorithm slightly changes. Let $L_{fw} = (u, S_{fw}, T_{fw})$ and $L_{bw} = (u, S_{bw}, T_{bw})$ be a forward and a backward label, respectively. The computation of the reduced cost changes in two cases: First, if forward and backward label have both visited an odd number of customers in U_k, i.e., $sr^k_{fw} = sr^k_{bw} = 1$, and $u \notin U_k$, then the dual price σ_k of the subset-row inequality $SR(U_k)$ has to be subtracted from the reduced cost. Second, if forward and backward label have both visited an even number of customers in U_k,

i.e., $sr^k_{fw} = sr^k_{bw} = 0$, and $u \in U_k$, then the dual price σ_k of the subset-row inequality $SR(U_k)$ has to be added to the reduced cost.

4.5.3 Strong Degree Inequalities

Strong degree inequalities were first introduced by Contardo *et al.* (2014) for the capacitated location-routing problem. A strong degree inequality $SD(i)$ in the VRTDSP can be defined for each customer vertex $i \in C$ as $\sum_{r \in R} g_{ir}\lambda_r \geq 1$, where $g_{ir} = 1$ if route r visits i at least once and $g_{ir} = 0$ otherwise. $SD(i)$ enforces partial elementary regarding vertex i. Separation is done by straightforward enumeration.

The addition of strong degree inequalities in the master problem requires the following adjustments to our pricing problem: Let $\psi_i \geq 0$ be the dual price of the strong degree inequality $SD(i)$. The value ψ_i must be subtracted from the reduced cost of a label only when vertex i is visited for the first time. Therefore, an additional binary resource sd^i, one for each inequality $SD(i)$, is necessary for the labeling algorithm to indicate if i was visited or not. Recall that the same vertex may be visited more than once in an ng-path relaxation.

Similar as for the subset-row inequalities, we can also avoid a point-wise comparison of all resources sd^i to strengthen the dominance as proposed by Contardo *et al.* (2015). The dominance rules of the VRTDSP change the condition regarding resource T^{cost} as follows: Let $L = (u, S, T)$ and $L' = (u, S', T')$ be two labels with identical last vertex u and let $G := \{k : sd^k_{L'} = 0, sd^k_L = 1\}$ be the index set of the new resources on which L is inferior compared to L'. The condition regarding the resource T^{cost} for label L to dominate label L' changes to $T^{cost} + \sum_{i \in G} \psi_i \leq T'^{cost}$. We can again use unreachable inequalities to further strengthen the dominance. A strong degree inequality $SD(i)$ is unreachable for a forward label $L = (u, S, T)$, if $T^{load} > Q - u_i$ or $T^{time} > b_i - \bar{t}_{ui}$. Each unreachable inequality is removed from the set G. A similar rule can be defined for backward labels.

In bidirectional labeling, we have to slightly change the merge procedure. The dual price ψ_i of a strong degree inequality $SD(i)$ must be added to the reduced cost of a merged path if forward and backward paths have visited node i.

4.5.4 Dynamic Neighborhood Extension

As mentioned in Subsection 4.4.3, the quality of the root lower bound computed by an ng-path relaxation strongly depends on the choice of the neighborhoods $(N_i)_{i \in C}$, but it is not obvious what a good choice is. To overcome this issue, a dynamic neighborhood extension was proposed for several different vehicle and arc routing problems (Roberti and Mingozzi, 2014; Bode and Irnich, 2015; Tilk and Irnich, 2016). This procedure can be seen as adding valid inequalities to the RMP that forbid routes with certain cycles. Herein, the ng-neighborhood is determined as proposed in Subsection 4.4.3 and the linear relaxation is solved. Now, the solution is scanned for cycles. Let $D = (i, \ldots, i)$ be a cycle in a route in the current solution. The node i is added to the ng-neighborhoods N_j of all vertices $j \in D$ to forbid cycle D in subsequent solutions of the pricing problem. Adding

new vertices to a neighborhood strengthens the resulting relaxation so that a formerly feasible route then becomes infeasible. Routes that become infeasible are removed from the RMP. This can be done by inspection. Clearly, a larger neighborhood leads to a more difficult pricing problem. Therefore, we limit the maximal size of all neighborhoods N_i by a constant ν^{max}.

4.6 Computational Results

The results reported in this section were obtained using a standard PC with an Intel(R) Core(TM) i7-5930k 3.5 GHz processor and 64 GB of main memory. The algorithms were coded in C++ and compiled at 64 bit with MS-Visual Studio 2013. The callable library of CPLEX 12.6 was used for solving the linear relaxations of the restricted master program in the column-generation algorithm. We tested our algorithm on the 56 benchmark instances for the VRTDSP proposed by Goel (2009) which can be obtained at `http://www.telematique.eu/research/downloads`. These instances are derived from the VRPTW benchmark instances of Solomon (1987) that can be grouped into six different classes: Randomly distributed customers (R1 and R2), clustered customers (C1 and C2) and a mixed distribution (RC1 and RC2). Instance classes R1, C1 and RC1 have tight time windows and a strict vehicle capacity, while C2, RC2, and R2 have wide time windows and a loose vehicle capacity. Each instance contains 100 customers and the service time at every customer is set to 60 minutes. Like Goel and Irnich (2016), we create smaller instances by considering only the first 25 or 50 customers.

In the following, we compare different variants of the labeling algorithm, different sizes of the *ng*-neighborhoods and various cutting strategies. Finally, we give detailed results of the best setting found. We set a CPU time limit of two hours for all computations. Preliminary tests showed that applying LDS and heuristic dominance is beneficial. In total, we use six different heuristic solvers with LDS, each with a limit of five good arcs. The first three solvers use heuristic dominance and a bad arc limit $\kappa = 0, 1$ or 2. The last three solvers use the exact dominance and the same bad arc limits as above. Moreover, we stop each labeling procedure as soon as 500 or more negative reduced-cost routes have been found.

4.6.1 Bidirectional Labeling and Dynamic Half-way Point

This subsection compares the monodirectional forward labeling with the bidirectional labeling. Moreover, the impact of using a dynamic half-way point is evaluated. We run two different tests, each with an *ng*-neighborhood size of $\nu = 10$: In a first test, we compare the solution of the linear relaxation of the RMP when either monodirectional forward labeling or bidirectional labeling with the dynamic or the static version of the half-way point is used in the pricing problem. Table 4.3 summarizes the results aggregated over the different instance sizes. The table contains the number of successfully solved linear relaxations, the average time in seconds, and the average number of pricing problem iterations needed in the static and dynamic version, respectively. The last

column indicates the average of the absolute values of the difference in the number of pricing problem iterations for the bidirectional labeling with static and the bidirectional labeling with dynamic half-way point. All average values are computed only over the instances that were solved by all versions.

$\|C\|$	no. solved			avg time			avg no. pricing			
	fw	sta	dyn	fw	sta	dyn	fw	sta	dyn	|sta-dyn|
25	56	56	56	76.86	76.56	39.11	55.96	35.71	34.68	10.39
50	47	51	54	848.13	441.25	288.09	152.71	105.97	104.60	48.20
100	12	19	26	1509.32	683.91	415.85	312.42	175.08	192.67	35.47
all	115	126	136	538.86	287.65	179.22	122.00	78.73	79.52	29.18

Table 4.3: Comparison between different labeling algorithms

The results of the first test demonstrates the superiority of the bidirectional labeling algorithms. Comparing the values of the forward labeling with the values of the bidirectional labeling with a static half-way point shows that eleven more linear relaxations can be solved and the computation time decreases by more than 45 % on average. Note that the results for the 25 customer instances do not differ significantly while for the 50 and 100 customer instances the gain in terms of computation time is even larger. The algorithm with the forward labeling needs on average 30 % more pricing iteration to solve the linear relaxation. This can be caused by the larger number of negative reduced-cost columns that can be found in each iteration with bidirectional labeling.

Moreover, the results of the first test show also that the labeling with a dynamic half-way point is superior to the static version. Ten more linear relaxations can be solved and the solution of a single instance takes on average nearly half of the time. The average number of pricing problem iterations needed to solve a single instance is nearly identical for both versions but the absolute value of the difference fluctuates over all instances. This is due to the different columns added in the static and dynamic version when using pricing heuristics. Therefore a second test is necessary to investigate the benefit of the dynamic half-way point compared to the static half-way point: We solve the linear relaxation of the RMP while the pricing problem is solved with the static as well as with the dynamic half-way point in each iteration. Thereby, a fair comparison of both labeling algorithms is given because they are solved with the same reduced cost in each iteration. In the second test, we computed the following values per instance: the ratio dynamic to static of the values for added labels, extended labels, and time needed in pricing. The ratios are computed as the geometrical mean over all pricing iterations. Additionally, we compute for all instances the coefficient of variation (COV) of the dynamic half-way point as standard deviation divided by arithmetic mean. Whenever the linear relaxation is not completely solved within the time limit, the values are taken over the iterations solved up to this point. Table 4.4 contains the minimum, geometrical mean, and maximum of these values, aggregated over the different instance sizes.

The results of the second test confirm the superiority of the dynamic half-way point version. The average ratio of the added and extended labels is for all instance sizes nearly

$\|C\|$	added labels			extended labels			time			COV		
	min	avg	max	min	avg	max	min	avg	max	min	avg	max
25	0.48	0.86	1.07	0.45	0.73	0.90	0.11	0.64	1.68	0.02	0.10	0.72
50	0.75	0.87	1.02	0.59	0.75	0.95	0.32	0.67	1.02	0.02	0.14	0.50
100	0.65	0.87	0.96	0.56	0.75	0.87	0.21	0.59	0.86	0.02	0.11	0.55
all		0.87			0.74			0.63			0.12	

Table 4.4: Ratios for Using the Dynamic and Static Half-Way Point with Identical Reduced Cost

identical around 0.87 and 0.74, respectively. This denotes that only 87 % of the labels needed in the static version, are needed in the dynamic version and that only 74 % of the labels extended in the static version are extended in the dynamic version. This results in an average computation time of 63 % for the dynamic version in relation to the static. Moreover, the detailed results show that the overall problem can be solved faster with the static half-way point in only eleven out of 168 instances, where none of these instances take more than 20 seconds to solve. The coefficient of variation of the dynamic half-way point denotes that during the solution of a single instance, the value of the dynamic half-way point varied by twelve percent on average. This illustrates the positive impact of the dynamic half-way point, because the value of the computational optimal half-way point for a single instance is different for each iteration of the pricing problem due to the current reduced cost. Therefore, we use the labeling with the dynamic half-way point in the following computations.

4.6.2 Neighborhood Size

In this subsection, we compare the results for different ng-path relaxations with sizes of $\nu = 6, 8, 10, 12, 14$, and 16 when only the linear relaxation is solved. Table 4.5 contains for all instance sizes, the number of times the linear relaxation was successfully solved in the time limit and the average time it has taken. Table 4.6 contains for all instance sizes the average and maximum improvement of the linear relaxation value when the ng-neighborhood size ν is increased. In order to provide a fair comparison, the maximum and average times and linear relaxation values are taken only over those instances for which the linear relaxation was solved by all ng-path relaxations.

As can be seen from Table 4.5, the number of successfully solved linear relaxations takes the maximum value for $\nu = 10$ and 12. Moreover, the time to solve the linear relaxation takes its minimum for $\nu = 10$. This is surprising because one would expect that a smaller value of ν would lead to smaller computation times. We ascribe this result to the increasing number of feasible routes for smaller values of ν. Table 4.6 shows that the increase in the bound provided by the linear relaxation decreases as the value of ν gets larger, e.g., the average increase is only 0.05 and the maximum 1.35 percent when

|C|	number of solved root nodes						time [sec]					
	ν =6	8	10	12	14	16	ν =6	8	10	12	14	16
25	56	56	56	56	56	56	46.0	39.3	28.6	68.5	138.6	320.7
50	52	52	54	54	51	49	691.3	637.8	393.4	576.4	739.1	846.0
100	25	27	26	26	26	27	1414.7	1617.2	1556.3	1387.7	1505.9	1694.2
all	133	135	136	136	133	132	515.2	524.5	416.4	477.1	590.4	744.8

Table 4.5: Comparison of the Solution Time for Different ng-Neighborhoods

|C|	average increase of the root LB [%]					maximum increase of the root LB [%]				
	$6 \to 8$	$8 \to 10$	$10 \to 12$	$12 \to 14$	$14 \to 16$	$6 \to 8$	$8 \to 10$	$10 \to 12$	$12 \to 14$	$14 \to 16$
25	0.38	0.08	0.07	0.00	0.01	6.11	2.46	1.35	0.24	0.23
50	0.36	0.15	0.02	0.01	0.00	4.55	3.63	0.31	0.16	0.06
100	0.21	0.03	0.05	0.01	0.01	1.57	0.21	0.39	0.07	0.09
all	0.35	0.10	0.05	0.01	0.00	6.11	3.63	1.35	0.24	0.23

Table 4.6: Comparison of the Root Lower Bound for Different ng-Neighborhoods

ν is increased from 10 to 12. Therefore, we set the ng-neighborhood size to $\nu = 10$ for all following computations.

4.6.3 Cutting Strategy

This subsection presents the results for solving Formulation (4.1) with different cutting strategies. After cutting is finished, we branch on the arcs of the original network to finally ensure integer solutions. Branching is implemented by deleting arcs from the auxiliary network. The following variable-selection strategy is used: Among all arcs (i, j) with a fractional value in the current solution, we choose the one closest to 0.5. Ties are broken by preferring arcs with higher cost c_{ij}. Moreover, nodes in the branch-and-bound tree are processed according to the minimum solution value of their linear relaxation.

In a first test, we compare the results for using each class of valid inequalities on its own. A valid inequality is generated when it is violated by more than 0.05. We use the following parameters to define the further cutting strategy: the node level up to which the cuts can be added in the branch-and-bound tree as well as the maximum number of cuts in total and per iteration. The level of a node in the branch-and-bound-tree is defined as the number of branching decisions taken up to this node. In addition, there are some class specific parameters: The maximum size ν^{max} up to which the ng-neighborhood can be dynamically extended, the maximum size w^{max} of the candidate sets W for the 2-path inequalities, and the maximum number s^{max} of subset-row cuts a customer can be included in. Pre-tests have shown that it is important to carefully choose these parameters. The best settings found for each class of valid inequalities are summarized in Table 4.7.

class	level	maximum number		specific
		in total	per iteration	
Strong Degree	0	20	10	-
Dyn. Ext.	0	-	10	$\nu^{max} = 20$
2-path	0	100	50	$w^{max} = 8$
Subset-row	∞	20	10	$s^{max} = 2$

Table 4.7: Best Settings for Applying each Class of Valid Inequalities Individually

Table 4.8 compares the results of applying each class of valid inequalities on its own and the results when no valid inequalities are used. To ensure a fair comparison, we take only the instances into account for which the linear relaxation was solved in the time limit with a fractional solution. The table contains, for each instance size, the number of remaining instances, the number of solved instances, and the average time the solution process has taken in seconds for each setting. The results show that applying strong degree inequalities or the dynamic neighborhood extension has only little impact on the number of solved instances and the solution time. However, the dynamic neighborhood extension is superior to the strong degree inequalities. Applying 2-path inequalities increases the number of solved instances by two and decreases the solution time on average by nearly five percent. Moreover, detailed results show that these two instances can not be solved by any other of the settings. However, the best results are obtained with applying subset-row inequalities.

|C|	no.	no. of solved instances					time [sec]				
	inst	none	SD	DynEx	2Path	SR	none	SD	DynEx	2Path	SR
25	21	21	21	21	21	21	439.64	441.10	391.04	416.71	281.60
50	27	10	10	10	11	14	4754.41	4752.17	4730.83	4484.04	4526.05
100	15	3	3	4	4	4	5993.77	5993.77	5986.75	5815.30	5643.69
all	63	34	34	35	36	39	3650.06	3649.62	3622.45	3488.34	3419.78

Table 4.8: Results for Applying each Class of Valid Inequalities Individually
SD=Strong degree inequalities, DynEx= Dynamic neighborhood extension, 2Path = 2-path inequalities,SR=subset-row inequalities

Based on this results we decided to combine the subset-row inequalities and the 2-path inequalities together with one or none of the other two classes. Note that we do not combine strong degree inequalities and dynamic neighborhood extension together, since they are both used to ensure elementarity. Moreover, we separate them in order: First, 2-path-inequalities because they are robust. Second, subset-row-inequalities because they have the biggest influence. Third, dynamic neighborhood extension or strong degree inequalities. Further tests have confirmed that this order is superior to all other orders. Table 4.9 contains the number of solved instances and the average solution time per instance size for the three resulting settings. The results for all three settings are similar but all three are superior to applying a single class of valid inequalities on its

own. However, applying 2-path inequalities, subset-row inequalities and dynamic neighborhood extensions lead to the best results. This confirms the superiority of the dynamic neighborhood extension compared to the strong degree inequalities.

$\|C\|$	no. of solved instances			time[sec]		
	2path+SR	+DynEx	+SD	2path+SR	+DynEx	+SD
25	21	21	21	186.68	183.63	182.54
50	16	17	13	4138.18	3937.09	4178.47
100	5	5	5	5344.13	5332.41	5342.31
all	42	43	39	3156.59	3067.05	3176.51

Table 4.9: Results for Applying Classes of Valid Inequalities Together
2Path+SR = 2-path and subset-row inequalities,+Dynex=2-path, subset-row inequalities and dynamic neighborhood extension, +SD=2-path, subset-row and strong degree inequalities

4.6.4 Comparison with Goel and Irnich (2014, 2016)

In this subsection, we compare our best setting found in the previous tests with the results of Goel and Irnich (2014, 2016). We also give detailed results of our best setting for each instance individually.

Table 4.10 shows the results of the comparison aggregated by instance size. The table contains the number of successfully solved root linear relaxations, the number of the instances solved to optimality, the time the solution of the root node takes on average, the time the overall solution process takes on average over all instances, and the time the overall solution process takes on average only over those instances that were solved by both algorithms.

	Best Setting					Goel and Irnich (2014, 2016)				
	no. solved		time[sec]			no. solved		time[sec]		
|C|	root	tree	root	tree	solved	root	tree	root	tree	solved
25	56	56	37.86	90.51	48.86	56	54	117.03	404.25	152.55
50	54	44	1052.13	2460.74	236.58	45	28	2069.04	3955.56	711.12
100	26	12	4951.06	6000.44	419.31	14	6	5733.51	6530.21	948.63
all	136	112	2013.69	2850.56	133.85	115	88	2639.86	3630.01	384.56

Table 4.10: Comparison of our Best Setting with the Results in (Goel and Irnich, 2014, 2016)

In total, we can solve 24 more instances and the solution time decreases by around 21 %. Moreover, the average solution time for the instances solved by both algorithms decreases in our setting about more than 60 %. The results for solving only the root node

are similar: Our algorithm is able to solve 21 more linear relaxations and the solution time decreases by around 24 %.

Table 4.11 contains the detailed results of the best setting for all instances. The first column contains the name of the instance, followed by the solution time in seconds, the gap at the root node in percent, the percentage the root gap was closed at the end of the optimization, and the value of the best known solution for instance sizes 25, 50 and 100. Values marked with an * are optimal solution values. The algorithm is able to solve all 25 customer instances in 90 seconds on average. Moreover, the computation time never exceeds 30 minutes. 44 out of 56 instances with 50 customers are solved to optimality and for only seven of them, the computation time exceeds 45 minutes. The 100 customer instances are hard to solve and only twelve optimal solutions were found.

4.7 Conclusion

This paper has investigated a new branch-and-price-and-cut algorithm for *the vehicle routing and truck driver scheduling problem* (VRTDSP) with U.S. hours of service regulations. We presented backward resource extension functions in order to build a bidirectional labeling algorithm with a sophisticated merge procedure. The concept of using a dynamic half-way point and different other known techniques, e.g., the *ng*-path relaxation and limited discrepancy search, were used to speed up the solution process of the pricing problem. In addition, valid inequalities for the vehicle routing problem with time windows were adapted to strengthen the linear relaxation. In the computational experiments, we tested the impact of using a dynamic half-way point, different *ng*-neighborhood sizes and cutting strategies. Our findings are the following: First, using a dynamic half-way point reduces the number of extended labels significantly resulting in a huge gain in terms of computation time. Second, choosing a moderate *ng*-neighborhood size of ten provides an excellent tradeoff between the strength of the resulting bounds and the computational effort. Third, a careful choice of the parameters is necessary when applying valid inequalities. Using subset-row inequalities with the right parameters has by far the biggest impact when applying a single class of valid inequalities on its own. Fourth, using a dynamic neighborhood extension for the *ng*-path relaxation is superior to applying strong degree inequalities. Fifth, when combining different classes of valid inequalities the choice of the classes and the order of applying them are crucial. The best setting found was applying 2-path, subset-row inequalities, and the dynamic neighborhood extension in that order. The computational experiments with this setting show that our method delivers improved results for the VRTDSP benchmark instances introduced by Goel (2009). Our algorithm is able to solve all 25 customer instances in less than half an hour to optimality. In addition, nearly 80 % of the 50 customer instances were solved to optimality with a time limit of two hours of computation time. In total, 24 new optimal solutions could be found and the solution time could be significantly reduced compared to Goel and Irnich (2014, 2016).

inst	25 customer				50 customer				100 customer			
	Time	Gap		UB	Time	Gap		UB	Time	Gap		UB
		Root	clo.			Root	clo.			Root	clo.	
C101	0.10	0.0	-	191.17*	0.45	0.0	-	362.17*	6.56	0.0	-	826.83*
C102	9.05	0.0	-	190.08*	10.06	0.0	-	361.08*	1683.34	0.0	-	826.83*
C103	52.23	0.0	-	189.42*	354.91	0.0	-	360.42*	7200.00	0.0	-	825.17*
C104	856.35	0.1	100.0	186.67*	7200.00	-	-	358.17	7200.00	-	-	821.08
C105	0.27	0.0	-	191.17*	1.17	0.0	-	362.17*	13.01	0.0	-	826.83*
C106	0.20	0.0	-	191.17*	1.68	0.0	-	362.17*	64.79	0.0	-	826.83*
C107	0.60	0.0	-	191.17*	4.32	0.0	-	362.17*	148.92	0.0	-	826.83*
C108	2.08	0.0	-	189.75*	18.58	0.0	-	360.75*	599.26	0.1	100.0	825.42*
C109	39.58	0.1	100.0	187.83*	192.19	0.1	100.0	358.83*	7200.00	1.0	9.7	823.50
C201	2.58	8.7	100.0	248.00*	272.85	11.7	100.0	434.75*	3199.48	18.1	100.0	825.42*
C202	3.39	0.0	-	217.83*	7200.00	4.9	46.9	390.50	7200.00	17.5	0.0	782.08
C203	10.06	0.0	-	217.83*	941.54	0.0	-	362.75*	7200.00	-	-	703.42
C204	32.79	0.0	-	214.17*	7200.00	-	-	349.50	7200.00	-	-	652.58
C205	1.07	0.0	-	214.42*	81.01	0.0	-	359.33*	7200.00	5.2	0.0	627.00
C206	1.36	0.0	-	214.42*	171.61	0.0	-	359.33*	2121.84	0.0	-	585.33*
C207	38.17	0.0	-	214.17*	754.42	0.0	-	358.58*	7200.00	-	-	642.67
C208	4.76	0.0	-	214.17*	597.41	0.0	-	359.08*	7200.00	-	-	639.00
R101	0.29	1.3	100.0	502.25*	1.89	0.0	-	847.83*	1021.34	0.6	100.0	1280.50*
R102	0.33	0.0	-	446.25*	33.71	0.5	100.0	753.92*	7200.00	1.0	66.0	1152.08
R103	1.04	0.0	-	401.08*	1450.91	1.1	100.0	649.08*	7200.00	12.7	0.0	1013.33
R104	1.47	0.0	-	359.42*	7200.00	0.6	51.6	536.42	7200.00	-	-	935.58
R105	0.48	0.7	100.0	438.17*	90.81	0.8	100.0	749.58*	7200.00	2.4	49.1	1076.67
R106	1.49	0.0	-	407.08*	1429.34	1.4	100.0	687.67*	7200.00	12.5	0.0	1059.08
R107	8.64	1.4	100.0	391.83*	4147.12	1.4	100.0	610.58*	7200.00	-	-	907.00
R108	359.27	1.6	100.0	349.42*	7200.00	1.4	0.0	530.17	7200.00	-	-	932.75
R109	4.66	2.3	100.0	385.08*	6457.09	1.9	100.0	637.83*	7200.00	7.7	3.8	964.25
R110	99.12	1.1	100.0	354.42*	2540.30	0.6	100.0	571.92*	7200.00	-	-	926.58
R111	5.96	0.8	100.0	387.67*	7200.00	6.6	17.6	615.58	7200.00	-	-	972.67
R112	700.96	1.7	100.0	337.33*	7200.00	1.8	34.3	529.42	7200.00	-	-	851.08
R201	0.44	0.7	100.0	463.58*	17.21	0.2	100.0	798.92*	2726.06	0.7	100.0	1157.00*
R202	0.61	0.0	-	410.75*	255.73	0.4	100.0	714.33*	7200.00	2.9	31.9	1063.25

Continued on next page

Table 4.11 – Continued from previous page

inst	25 customer				50 customer				100 customer			
	Time	Gap		UB	Time	Gap		UB	Time	Gap		UB
		Root	clo.			Root	clo.			Root	clo.	
R203	5.54	1.4	100.0	391.83*	2092.42	1.6	100.0	626.00*	7200.00	-	-	952.33
R204	34.27	1.6	100.0	355.17*	4540.60	0.8	100.0	516.17*	7200.00	-	-	905.33
R205	1.42	0.3	100.0	404.08*	173.77	0.6	100.0	695.25*	7200.00	4.0	18.2	994.08
R206	19.76	0.5	100.0	378.08*	5325.31	1.4	100.0	642.17*	7200.00	-	-	997.42
R207	3.64	0.0	-	367.17*	6325.04	1.5	100.0	578.42*	7200.00	-	-	961.67
R208	102.35	1.4	100.0	341.08*	7200.00	7.0	0.0	527.33	7200.00	-	-	944.17
R209	7.25	1.9	100.0	376.75*	2037.77	0.8	100.0	615.58*	7200.00	12.0	0.0	976.58
R210	10.47	0.9	100.0	411.75*	7200.00	5.2	27.6	681.58	7200.00	11.4	0.0	1015.33
R211	23.10	0.2	100.0	351.17*	7200.00	8.1	13.5	595.25	7200.00	-	-	858.17
RC101	0.30	0.0	-	358.25*	2.45	0.0	-	632.58*	7200.00	5.0	15.6	1287.75
RC102	3.78	0.0	-	335.92*	41.59	0.0	-	604.42*	7200.00	7.4	0.0	1177.08
RC103	9.14	0.0	-	327.08*	841.76	0.0	-	584.67*	7200.00	-	-	1119.67
RC104	91.48	0.0	-	299.75*	3460.04	0.0	-	522.92*	7200.00	-	-	996.67
RC105	0.62	0.0	-	334.75*	16.43	0.0	-	613.75*	7200.00	7.7	0.0	1217.92
RC106	3.18	0.0	-	310.83*	92.33	0.0	-	564.92*	7200.00	5.7	0.0	1111.67
RC107	31.57	0.0	-	296.33*	1152.58	0.0	-	522.67*	7200.00	-	-	1066.17
RC108	821.22	0.0	-	294.50*	5131.88	0.0	-	517.67*	7200.00	-	-	1008.08
RC201	0.17	0.0	-	360.50*	2.31	0.0	-	684.83*	439.94	0.2	100.0	1294.58*
RC202	1.15	0.0	-	338.17*	13.65	0.0	-	613.83*	7200.00	2.5	0.0	1144.92
RC203	4.34	0.0	-	327.08*	63.32	0.0	-	594.92*	7200.00	-	-	1068.25
RC204	77.41	0.0	-	299.75*	7200.00	1.5	0.0	491.58	7200.00	-	-	937.08
RC205	0.28	0.0	-	338.08*	11.65	0.0	-	631.83*	7200.00	1.2	54.6	1184.25
RC206	0.52	0.0	-	324.25*	23.85	0.0	-	610.17*	7200.00	3.8	31.8	1104.75
RC207	2.67	0.0	-	298.33*	226.49	0.0	-	560.00*	7200.00	-	-	1041.75
RC208	1573.67	1.4	100.0	294.50*	7200.00	2.2	0.0	517.67	7200.00	-	-	879.08
all	90.51	0.5	100.0		2460.74	1.2	70.1		6000.44	4.6	32.5	

Table 4.11: Detailed Results for Best Cutting Strategy

References

Baldacci, R., Mingozzi, A., and Roberti, R. (2011). New route relaxation and pricing strategies for the vehicle routing problem. *Operations Research*, **59**(5), 1269–1283.

Bartodziej, P., Derigs, U., Malcherek, D., and Vogel, U. (2009). Models and algorithms for solving combined vehicle and crew scheduling problems with rest constraints: An application to road feeder service planning in air cargo transportation. *OR Spectrum*, **31**, 405–429.

Bode, C. and Irnich, S. (2015). In-depth analysis of pricing problem relaxations for the capacitated arc-routing problem. *Transportation Science*, **49**(2), 369–383.

Ceselli, A., Righini, G., and Salani, M. (2009). A column generation algorithm for a rich vehicle-routing problem. *Transportation Science*, **43**(1), 56–69.

Contardo, C., Cordeau, J.-F., and Gendron, B. (2014). An exact algorithm based on cut-and-column generation for the capacitated location-routing problem. *INFORMS Journal on Computing*, **26**(1), 88–102. (Forthcoming.).

Contardo, C., Desaulniers, G., and Lessard, F. (2015). Reaching the elementary lower bound in the vehicle routing problem with time windows. *Networks*, **65**(1), 88–99.

Derigs, U., Kurowsky, R., and Vogel, U. (2011). Solving a real-world vehicle routing problem with multiple use of tractors and trailers and EU-regulations for drivers arising in air cargo road feeder services. *European Journal of Operational Research*, **213**, 309–319.

Desaulniers, G., Desrosiers, J., and Solomon, M., editors (2005). *Column Generation*. Springer, New York, NY.

Desaulniers, G., Desrosiers, J., and Spoorendonk, S. (2010). The vehicle routing problem with time windows: State-of-the-art exact solution methods. In J. J. Cochran, L. A. Cox, P. Keskinocak, J. P. Kharoufeh, and J. C. Smith, editors, *Wiley Encyclopedia of Operations Research and Management Science*, volume 8, pages 5742–5749. John Wiley & Sons, Inc.

Desaulniers, G., Madsen, O. B., and Ropke, S. (2014). The vehicle routing problem with time windows. In Toth and Vigo (2014), chapter 5, pages 119–159.

Drexl, M. and Prescott-Gagnon, E. (2010). Labelling algorithms for the elementary shortest path problem with resource constraints considering eu drivers' rules. *Logistics Research*, **2**(2), 79–96.

Dror, M. (1994). Note on the complexity of the shortest path models for column generation in VRPTW. *Operations Research*, **42**(5), 977–978.

European Union (2006). Regulation (EC) No 561/2006 of the European Parliament and of the Council of 15 March 2006 on the harmonisation of certain social legislation relating to road transport and amending Council Regulations (EEC) No 3821/85 and (EC) No 2135/98 and repealing Council Regulation (EEC) No 3820/85. Official Journal of the European Union L 102, 11.04.2006.

Federal Motor Carrier Safety Administration (2011). Hours of service of drivers. Federal Register Vol. 76, No. 248, pages 81134 - 81188.

Feillet, D., Dejax, P., Gendreau, M., and Guéguen, C. (2004). An exact algorithm for the elementary shortest path problem with resource constraints: Application to some vehicle routing problems. *Networks*, **44**(3), 216–229.

Feillet, D., Gendreau, M., and Rousseau, L.-M. (2007). New refinements for the solution of vehicle routing problems with branch and price. *INFOR*, **45**(4), 239–256.

Goel, A. (2009). Vehicle scheduling and routing with drivers' working hours. *Transportation Science*, **43**(1), 17–26.

Goel, A. (2010). Truck driver scheduling in the european union. *Transportation Science*, **44**(4), 429–441.

Goel, A. (2014). Hours of service regulations in the United States and the 2013 rule change. *Transport Policy*, **33**, 48–55.

Goel, A. and Irnich, S. (2014). An exact method for vehicle routing and truck driver scheduling problems. Technical Report No. 33, Jacobs University, School of Engineering and Science, Bremen, Germany.

Goel, A. and Irnich, S. (2016). An exact method for vehicle routing and truck driver scheduling problems. *Transportation Science*. Forthcoming.

Goel, A. and Vidal, T. (2014). Hours of service regulations in road freight transport: An optimization-based international assessment. *Transportation Science*, **48**, 391–412.

Irnich, S. (2008). Resource extension functions: Properties, inversion, and generalization to segments. *OR Spectrum*, **30**(1), 113–148.

Irnich, S. and Desaulniers, G. (2005). Shortest path problems with resource constraints. In Desaulniers *et al.* (2005), chapter 2, pages 33–65.

Jepsen, M., Petersen, B., Spoorendonk, S., and Pisinger, D. (2008). Subset-row inequalities applied to the vehicle-routing problem with time windows. *Operations Research*, **56**(2), 497–511.

Kohl, N., Desrosiers, J., Madsen, O., Solomon, M., and Soumis, F. (1999). 2-path cuts for the vehicle routing problem with time windows. *Transportation Science*, **33**(1), 101–116.

Kok, A., Hans, E., and Schutten, J.M.J.and Zijm, W. (2010). A dynamic programming heuristic for vehicle routing with time-dependent travel times and required breaks. *Flexible Services and Manufacturing Journal*, **22**, 83–108.

Kopfer, H. and Meyer, C. M. (2009). A model for the traveling salesman problem including the ec regulations on driving hours. In B. Fleischmann, K.-H. Borgwardt, R. Klein, and A. Tuma, editors, *Operations Research Proceedings 2008: Selected Papers of the Annual International Conference of the German Operations Research Society (GOR) University of Augsburg, September 3-5, 2008*, pages 289–294. Springer Berlin Heidelberg, Berlin, Heidelberg.

Meyer, C. M. and Kopfer, H. (2008). Restrictions for the operational transportation planning by regulations on drivers' working hours. In A. Bortfeldt, J. Homberger, H. Kopfer, G. Pankratz, and R. Strangmeier, editors, *Intelligent Decision Support: Current Challenges and Approaches*, pages 177–186. Gabler, Wiesbaden.

Prescott-Gagnon, E., Desaulniers, G., Drexl, M., and Rousseau, L.-M. (2010). European driver rules in vehicle routing with time windows. *Transportation Science*, **44**(4), 455–473.

Rancourt, M.-E. and Paquette, J. (2014). Multicriteria optimization of a long-haul routing and scheduling problem. *Journal of Multi-Criteria Decision Analysis*, **21**(5-6), 239–255. MCDA-12-0037.R1.

Rancourt, M.-E., Cordeau, J.-F., and Laporte, G. (2013). Long-haul vehicle routing and scheduling with working hour rules. *Transportation Science*, **47**(1), 81–107.

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

Roberti, R. and Mingozzi, A. (2014). Dynamic ng-path relaxation for the delivery man problem. *Transportation Science*, **48**(3), 413–424.

Salani, M. (2005). *Branch-and-price algorithms for Vehicle Routing Problems*. Ph.D. dissertation, Università degli Studi di Milano, Facoltà die Scienze Matematiche, Fisiche e Naturali, Dipartimento di Tecnologie dell'Informatione, Milan, Italy.

Savelsbergh, M. W. P. and Sol, M. (1998). DRIVE: dynamic routing of independent vehicles. *Operations Research*, **46**, 474–490.

Solomon, M. (1987). Algorithms for the vehicle routing and scheduling problem with time window constraints. *Operations Research*, **35**(2), 254–265.

Tilk, C. and Irnich, S. (2016). Dynamic programming for the minimum tour duration problem. *Transportation Science*. Forthcoming.

Tilk, C., Rothenbächer, A.-K., Gschwind, T., and Irnich, S. (2016). Asymmetry helps: Dynamic half-way points for solving shortest path problems with resource constraints faster. Technical Report LM-2016-05, Chair of Logistics Management, Gutenberg School of Management and Economics, Johannes Gutenberg University Mainz, Mainz, Germany.

Toth, P. and Vigo, D., editors (2014). *Vehicle Routing*. MOS-SIAM Series on Optimization. Society for Industrial and Applied Mathematics, Philadelphia, PA.

Xu, H., Chen, Z., Rajagopal, S., and Arunapuram, S. (2003). Solving a practical pickup and delivery problem. *Transportation Science*, **37**(3), 347–364.

Zäpfel, G. and Bögl, M. (2008). Multi-period vehicle routing and crew scheduling with outsourcing options. *International Journal of Production Economics*, **113**, 980–996.

Chapter 5

Asymmetry Helps: Dynamic Half-Way Points for Solving Shortest Path Problems with Resource Constraints Faster

Christian Tilk, Ann-Kathrin Rothenbächer, Timo Gschwind, Stefan Irnich

Abstract

With their paper "Symmetry helps: Bounded bi-directional dynamic programming for the elementary shortest path problem with resource constraints" [Discrete Optimization 3, 2006, pp. 255–273] Righini and Salani introduced bounded bidirectional dynamic programming (DP) as an acceleration technique for solving variants of the shortest path problem with resource constraints (SPPRC). SPPRCs must be solved iteratively when vehicle routing and scheduling problems are tackled via Lagrangian relaxation or column-generation techniques. Righini and Salani and several subsequent works have shown that bounded bidirectional DP algorithms are often superior to their monodirectional counterparts, since the former can mitigate the fact that the number of labels increases strongly with the path length. Bidirectional DP has become a quasi-standard for solving SPPRCs with general resource extension functions. In computational experiments, however, one can still observe that the number of forward and backward label extensions is very unbalanced despite a symmetric bounding of a critical resource in the middle of its feasible domain. We exploit this asymmetry in forward and backward label extensions to reduce the overall workload by introducing a so-called dynamic half-way point, which is a dynamic bounding criterion based on the current state of the simultaneously solved forward and backward DPs. Experiments with the standard and the electric vehicle routing problem with time windows as well as the vehicle routing and truck driver scheduling problem confirm that dynamic half-way points better balance forward and backward labeling and reduce the overall runtime.

5.1 Introduction

Vehicle routing and scheduling problems have been the subject of intensive study for more than half of a century now. Standard variants of the *vehicle routing problem*

(VRP), such as the capacitated VRP and the *vehicle routing problem with time windows* (VRPTW), are well studied and effective solution algorithms can be found in the literature. Even more, a plethora of variants of the VRP has been investigated in thousands of scientific papers and powerful commercial vehicle routing software is available. Research on these more complex VRP variants is constantly ongoing, motivated by unsolved theoretical problems as well as continuous input from logistics practice (Drexl, 2012a; Irnich *et al.*, 2014).

For solving VRPs exactly, algorithms based on column generation and Lagrangian relaxation are the state-of-the-art (see, e.g., Poggi and Uchoa, 2014; Desaulniers *et al.*, 2014). Herein, the master program typically is a (possibly extended) set-partitioning formulation and the subproblem a variant of the *shortest path problem with resource constraints* (SPPRC, Irnich and Desaulniers, 2005). Any feasible solution in the SPPRC describes a feasible route in the respective VRP. The task of the pricing subproblem is to find at least one negative reduced-cost route, or to prove that no such route exists. In most applications, the solution of the SPPRC subproblem requires by far the largest portion of the overall computation time. Therefore, a lot of research has been spent in enhancing SPPRC algorithms. This includes the handling of elementarity constraints (Feillet *et al.*, 2004; Boland *et al.*, 2006) and relaxing elementarity (Irnich and Villeneuve, 2006; Baldacci *et al.*, 2011; Bode and Irnich, 2014), handling of non-robust cuts (Jepsen *et al.*, 2008; Contardo *et al.*, 2015), the use of completion bounds (Baldacci *et al.*, 2011), and SPPRC heuristics either dynamic-programming (DP) based (Irnich and Desaulniers, 2005; Feillet *et al.*, 2007) or based on metaheuristics (e.g., Desaulniers *et al.*, 2008).

The variants of SPPRC we focus on in this paper are those with constrained resources that are propagated in a general, not necessarily purely additive fashion. This includes the handling of time windows, resource-dependent cost functions, and interdependent resources as needed in many rich VRPs. In contrast, the special case of paths constrained by completely additive resources is often referred to as the *(resource) constrained shortest path problem (CSPP)*. Several highly effective algorithms have been developed for the CSPP (e.g., Garcia, 2009; Lozano and Medaglia, 2013; Pugliese and Guerriero, 2013; Horváth and Kis, 2016).

For the exact solution of SPPRCs with general resource constraints, DP based labeling algorithms are predominant (Irnich and Desaulniers, 2005). Let $G = (V, A)$ be the digraph of the SPPRC with vertex set V and arc set A. In labeling algorithms, a label refers to a partial path from the given source vertex $s \in V$ to some vertex $i \in V$. It holds the information about the resource consumption along this path and refers to its predecessor, the label of the partial path from which the actual label was extended. Starting with the initial partial path $P = (s)$, labeling algorithms

(1) select an unprocessed partial path $P = (s, \ldots, i)$,
(2) extend the partial path P along all arcs (i, j) of the forward star of vertex i,
(3) check whether the new partial paths $(P, j) = (s, \ldots, i, j)$ are feasible, and if so,
(4) store them so that they can be extended at a later point.

Labeling algorithms allow the steps (1)–(4) to be performed implicitly with the help of specific attributes, the resource values of the label. In particular, the extension step (2) propagates labels directly using so-called *resource extension functions* (REFs,

Desaulniers *et al.*, 1998). Repeating the steps (1)–(4) creates all possible paths starting at s. Dominance procedures are therefore invoked to identify and discard those partial paths and their labels that cannot lead to an optimal SPPRC solution.

The above labeling scheme is known as a monodirectional forward labeling algorithm because it extends partial paths only in the forward direction. For many types of SPPRCs, the direction of propagation can be reversed. It means that the first partial path is $P = (t)$, where t is the sink vertex, and partial paths $P = (j, \ldots, t)$ are then extended against the orientation of an arc (i, j) creating new partial paths $P' = (i, j, \ldots, t)$. This requires that REFs can be inverted. A systematic way to do this is described in (Irnich, 2008). However, reversion may be complicated or create a different set of resources needed to describe backward partial paths. Whether forward or backward labeling is then beneficial still depends on many factors, and is often not clear in advance. Even worse, both monodirectional forward and monodirectional backward labeling typically suffer from an *explosion of labels*. This effect, also known as *combinatorial explosion*, describes that progressively more labels are created, are extended, and need to be stored when the length (=number of arcs) of partial paths increases.

Righini and Salani (2006) have presented bounded bidirectional labeling in order to mitigate the explosion of labels. Both forward partial paths and backward partial paths are created, but processed only up to a so-called *half-way point*. The half-way point is defined with the help of a monotone resource (e.g., the time), often as the midpoint of its feasible domain. When labeling terminates, suitable forward and backward labels must be merged to obtain complete feasible s-t-paths.

The paper by Righini and Salani (2006) and several subsequent works have shown that bounded bidirectional labeling algorithms are usually superior to their monodirectional counterparts. Hence, bidirectional labeling has become a quasi-standard for solving SPPRCs with general resource extension functions. It works particularly well if forward and backward labeling are similar, i.e., the same number of resources can be used and fathoming criteria of comparable strength can be applied in both directions. However, in applications in which forward and backward labeling differs significantly, it is unclear how a reasonable half-way point should be defined. For example, in VRPs with renewable resources such as the electric vehicle routing problem with time windows (Desaulniers *et al.*, 2016), forward and backward labeling can require a different number and different types of resources (see Section 5.3.2). In such a situation, the computation time spent in one labeling direction can strongly exceed the time spent in the other direction. But even in cases with the same number of resources and similar fathoming criteria for both directions, it can still be observed in computational experiments that the number of forward and backward label extensions is very unbalanced despite of a symmetric bounding with the critical resource in the middle of its feasible domain. This can be caused by asymmetric VRP instance data, e.g., time windows being unevenly spread over the planning horizon. In addition, oscillation of dual prices over the column-generation iterations can cause further asymmetry in SPPRC instances. Note also that in many VRPs, the dual price of one vertex may either be assigned to the ingoing or outgoing arcs of this vertex, producing another form of asymmetry. With the goal of balancing the workload between the forward and backward labeling, one may define a different half-

way point in each column-generation iteration. However, it is unclear how to a priori set a computationally convenient half-way point so that the overall workload is minimized.

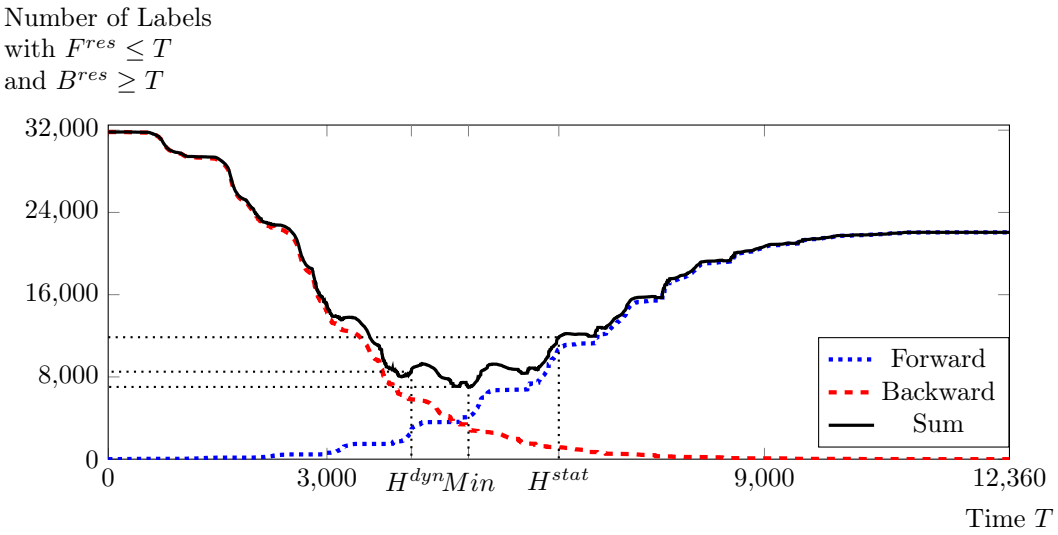

Figure 5.1: Example of an SPPRC instance with unbalanced forward and backward labeling; static half-way point $H^{stat} = 6{,}180$ in the middle of the time horizon and dynamic half-way point $H^{dyn} = 4{,}158$

In Figure 5.1, an example of an SPPRC instance with unbalanced forward and backward labeling is depicted. Herein, time is chosen as the monotone resource. The functions show, for any value T of the monotone resource, the number of processed forward labels with time $\leq T$, the number of processed backward labels with time $\geq T$, and the sum of both values. Obviously, forward labeling creates less labels than backward labeling, and bidirectional labeling even less whenever a half-way point is chosen somewhere in the middle (here between 2,523 and 11,118). Note that with a half-way point in the midpoint 6,180 of the time horizon $[0; 12{,}360]$, bidirectional labeling produces more forward and backward labels in total than with appropriately chosen smaller half-way points.

The contribution of the paper at hand is the introduction of a *dynamic half-way point*, which is a dynamic bounding criterion based on the current state of the simultaneously solved forward and backward SPPRC. The dynamic half-way point exploits the asymmetry in forward and backward label extensions. It can be used in all labeling algorithms that have the same monotone resource available in both directions. The idea of a dynamic bounding criterion has been sketched in an earlier work of Pecin (2014) and was used in (Pecin *et al.*, 2016) for successfully solving large-scale capacitated VRPs using a tailored arc-load indexed formulation that creates multiple start-labels on one side and herewith a strong imbalance between forward and backward labeling. As far as we know, however, the idea has not been thoroughly analyzed for VRPs with general resources. Our experiments with three types of VRPs confirm that dynamic half-way points better

balance forward and backward workload. We have selected the VRPTW as an example of a standard VRP with relatively few resources that have similar forward and backward REFs. The *electric vehicle routing problem with time windows* (EVRPTW), on the contrary, has a different number of forward and backward resources. Finally, the *vehicle routing and truck driver scheduling problem* (VRTDSP) has a larger number of resources resulting in generally more difficult SPPRC instances.

The paper is organized as follows. In Section 5.2, we formally describe bidirectional labeling for SPPRCs while emphasizing the differences of using a static or a dynamic half-way point. The definition of the three VRP variants and their forward and backward labels is given in Section 5.3. The impact of using a dynamic half-way point is computationally evaluated in Section 5.4. Finally, concluding remarks are given in Section 5.5.

5.2 Bidirectional Labeling

We start with a formal description of a generic bidirectional labeling algorithm. An instance of the SPPRC is given by the SPPRC network $G = (V, A)$, the source vertex $s \in V$, the sink vertex $t \in V$, and forward and backward REFs f_{ij} and b_{ij}, respectively, for each arc $(i, j) \in A$. For convenience, we assume that G is simple so that arcs (i, j) and their REFs can be uniquely described by tail and head vertex. Forward labels are denoted by F and backward labels by B.

The following assumptions are made for the remainder of the paper: Both forward and backward labels have a specific resource res. Let $F_i = F(P)$ denote the forward label associated with a forward partial path $P = (s, \ldots, i)$ and define $v(F_i) = i$. Then, for any arc $(i, j) \in A$ and the extension $F_j := f_{ij}(F_i)$ the inequality $F_i^{res} \leq F_j^{res}$ holds. Similarly, let $B_j = B(P)$ denote the backward label associated with a backward partial path $P = (j, \ldots, t)$ and define $v(B_j) = j$. Then, for any arc $(i, j) \in A$ and the extension $B_i := b_{ij}(B_j)$ the inequality $B_i^{res} \leq B_j^{res}$ holds. We call this resource res the *monotone resource* and assume that its domain is the interval $[L, U]$. Note that in most applications at least one monotone resource is naturally included in the problem, since time and load/residual capacity usually fulfill the requirements. If this is not the case, a monotone resource can always be created artificially, e.g., by counting the number of arcs in a path.

Different strategies to implement bidirectional labeling algorithms were presented in the literature and these strategies are very important for the effectiveness of the overall algorithm (Righini and Salani, 2006). Algorithm 5.1 shows a generic version of a labeling algorithm for solving an SPPRC. The generic parts are the label selection strategy (label setting, label correcting) determined by the functions getNextForwardLabel() and getNextBackwardLabel(), the dominance strategy determined by the function checkDominance(), and the function getNextDirection() that controls whether the next label to extend is a forward or a backward label.

We comment on Algorithm 5.1 in more detail. The algorithm requires the forward half-way point H_F and the backward half-way point H_B as an input. Only for $H_F \geq H_B$ optimality is guaranteed. Initial labels F and B are created for the trivial forward and

Algorithm 5.1: Generic Labeling Algorithm for SPPRCs

Input: Forward half-way point H_F and backward half-way point H_B (required $H_F \geq H_B$)

1 $F := F(s)$, $B := B(t)$;
2 **while** $F \neq null$ or $B \neq null$ **do**
3 $\quad$ *direction* := getNextDirection();
4 $\quad$ **if** *direction* $= forward$ **then**
5 $\quad\quad$ **if** $F^{res} \leq H_F$ **then**
6 $\quad\quad\quad$ **for** $(i,j) \in A$ *with* $i = v(F)$ **do**
7 $\quad\quad\quad\quad$ Propagate F to vertex j with REF f_{ij};
8 $\quad\quad$ $H_B := \max\{H_B, \min\{F^{res}, H_F\}\}$;
9 $\quad\quad$ $F :=$ getNextForwardLabel();
10 $\quad$ **else**
11 $\quad\quad$ **if** $B^{res} > H_B$ **then**
12 $\quad\quad\quad$ **for** $(i,j) \in A$ *with* $j = v(B)$ **do**
13 $\quad\quad\quad\quad$ Propagate B to vertex i with REF b_{ij};
14 $\quad\quad$ $H_F := \min\{H_F, \max\{B^{res}, H_B\}\}$;
15 $\quad\quad$ $B :=$ getNextBackwardLabel();
16 $\quad$ **if** checkDominance() **then**
17 $\quad\quad$ Apply dominance rules;

backward partial paths in Step 1. As long as there are unprocessed labels, the Steps 3–17 are repeated. First, in Step 3, the function getNextDirection() controls the direction of the next extension step. One must ensure that it returns $forward$ if $F \neq null$ and $B = null$, and *backward* if $F = null$ and $B \neq null$, while in case of $F, B \neq null$ both directions are possible. In Steps 5 and 11, it is ensured that only labels respecting the half-way criterion are extended. The forward propagation in Step 7 includes the creation of the tentative new label $F_j := f_{ij}(F)$, the feasibility check of the path associated with F_j, and if feasible, the storage of the new label F_j so that it can later be retrieved via getNextForwardLabel(). The backward propagation in Step 13 performs the respective operations for the tentative label $B_i := b_{ij}(B)$ and the corresponding partial backward path. The half-way points are updated in Steps 8 and 14.

In Steps 9 and 15, the function getNextForwardLabel()/getNextBackwardLabel() returns an unprocessed forward/backward label that is used for extension the next time the algorithm chooses the respective direction. If no unprocessed forward/backward label exists, the value *null* is returned. To allow the efficient retrieval of the next label to process, the unprocessed labels should be managed with an appropriate data structure. For example, labels may be stored in hash tables according to the value of the critical resource so that a label-setting approach can be implemented by retrieving forward/backward labels in ascending/descending order of the critical resource.

In Step 17, a dominance algorithm is called in order to identify and discard provably non-optimal labels. Note that the invocation of a dominance algorithm can always be

delayed to some convenient point in time. In particular, the determination of dominated labels, every time a new feasible label is created, is generally not efficient because it requires a quadratic number of comparisons of labels in the worst case. More efficient algorithms, e.g., based on the multidimensional divide-and-conquer principle exist (see Bentley, 1980) but require that batches of labels are tested for dominance. The function CHECKDOMINANCE() in Step 16 controlling the dominance algorithm should therefore call it only when there is the chance that the next label to process is dominated.

The two half-way points H_F and H_B are upper and lower bounds respectively for the final half-way point H. The update Steps 8 and 14 guarantee that $H_F \geq H_B$. Initially setting H_F and H_B to the same value, therefore, leaves these parameters fixed, and $H = H_F = H_B$ results. Moreover, the backward half-way point H_B can only be increased. This happens when the processed forward label has a larger value of the monotone resource (Step 8). Conversely, the forward half-way point H_F can only be decreased when the processed backward label has a smaller value of the monotone resource (Step 14). In the course of Algorithm 5.1, H_F and H_B approach each other and it terminates with $H = H_F = H_B$. Thus, depending on the initial values of H_F and H_B, the resulting algorithm behaves in the following manner:

$H_F = H_B > U$: monodirectional forward labeling algorithm;

$H_F = H_B < L$: monodirectional backward labeling algorithm;

$H_F = H_B \in (L, U)$: bidirectional labeling algorithm with static half-way point;

$U = H_F > H_B = L$: bidirectional labeling algorithm with dynamic half-way point.

In the two monodirectional cases $H_F = H_B > U$ and $H_F = H_B < L$ either of the conditions in Step 11 or Step 5 is always false. Consequently, extensions occur only in one direction. There is no specific requirement on the ordering in which labels are retrieved in Steps 9 and 15.

In the version with a static half-way point, H_F and H_B are initialized to the same value inside the domain of the monotone resource, e.g., the middle $(U + L)/2$. By defining GETNEXTDIRECTION() as $forward$ whenever $F \neq null$, and *backward* otherwise, Algorithm 5.1 performs all forward extensions before the backward extensions. Any other strategy, however, still produces the same set of forward and backward labels. Again, in the forward and in the backward part, labels can be processed in any order.

On the contrary, in the version with a dynamic half-way point ($H_F > H_B$), the forward processing should be performed in ascending order while the backward processing should be done in descending order of the value of the monotone resource. More precisely, in subsequent iterations $F :=$ GETNEXTFORWARDLABEL() should return non-decreasing values F^{res} and $B :=$ GETNEXTBACKWARDLABEL() should return non-increasing values B^{res}. Otherwise, e.g., prematurely processing a forward label with a large value of the monotone resource would force the (backward) half-way point towards U resulting in almost pure forward labeling. Anyway, Algorithm 5.1 maintains $H_B \leq H_F$ so that even a non-monotone label extension does not produce incorrect results. Furthermore, non-decreasing values F^{res} and non-increasing values B^{res} guarantee that all

forward/backward labels with a value of the monotone resource smaller/greater than H_B/H_F have already been extended.

In the course of Algorithm 5.1, the two half-way points approach each other due to the update Steps 8 and 14. Clearly, as soon as $H_F = H_B$, the labeling behaves like the bidirectional labeling with this static half-way point. Note that choosing initial values $H_F < U$ and/or $H_B > L$ restricts the value of the dynamic half-way point to lie in the interval $[H_F, H_B]$. However, the difference to a classical bidirectional labeling algorithm is that the half-way point is implicitly chosen with the function GETNEXTDIRECTION(). Therefore, the criterion for alternating between forward and backward labeling is of crucial importance for choosing a good half-way point and the overall performance of Algorithm 5.1.

An ideal strategy to choose a direction would be one that minimizes the overall computation time. It is not clear in advance how this can be achieved, since the computational effort is influenced by many different factors. A good indicator may be the overall number of labels that have to be considered in the course of Algorithm 5.1. However, this overall number is not known at the time when a direction must be determined. Reasonable choices try to estimate the remaining effort, e.g., by choosing

(1) the direction with the smaller number of *generated* labels,
(2) the direction with the smaller number of *processed* labels,
(3) the direction with the smaller number of *unprocessed* labels.

All these measures are estimates only because the overall computational effort is also influenced, e.g., by the portion of the time spent in dominance tests. This computation time is however not directly proportional to the overall number of labels.

When Algorithm 5.1 terminates, compatible forward and backward labels must be merged to obtain complete s-t-paths. Righini and Salani (2006) elaborate a half-way point test that avoids the creation of the same s-t-path from different pairs of forward and backward labels.

We now consider again the example shown in Figure 5.1. Recall that bidirectional labeling with a static half-way point $H^{stat} = 6{,}180$ in the middle of the time horizon reduces the number of labels by approximately 50 % compared to monodirectional forward labeling (even more compared to backward). Although the overall number of backward labels is larger than the number of forward labels, our dynamic half-way point is smaller than the static half-way point. This is justified by the course of the forward and backward labeling, since the strong increase of backward labels occurs at small time values T that are not relevant for the bidirectional labeling if the half-way point ist chosen larger. The dynamic half-way point shown in Figure 5.1 results from the strategy of choosing the direction with the smaller number of unprocessed labels. For this SPPRC instance, the dynamic half-way point $H^{dyn} = 4{,}158$ misses the global minimum $Min = 4{,}941$ of the overall number of generated labels. Nevertheless, compared to the static half-way point, the number of generated labels is further reduced by around 30 %.

5.3 Problem Descriptions

In this section, we sketch the three considered VRP variants, give references for the most effective column-generation based algorithms, and describe the resources that have been used in the DP labeling algorithms to solve the associated SPPRC pricing problems. In particular, we point out those cases in which forward and backward labeling differ significantly.

5.3.1 Vehicle Routing Problem with Time Windows (VRPTW)

The VRPTW is an extension of the capacitated VRP in which additionally travel times between locations are given and it is required that the service at a customer starts within a pre-specified hard time window. Surveys on the VRPTW are (Cordeau *et al.*, 2002; Desaulniers *et al.*, 2010; Baldacci *et al.*, 2012b; Desaulniers *et al.*, 2014). Different branch-and-price-and-cut algorithms were proposed for the VRPTW in the literature (e.g., Desrochers *et al.*, 1992; Kohl *et al.*, 1999; Irnich and Villeneuve, 2006; Desaulniers *et al.*, 2008).

When solving the SPPRC of the VRPTW pricing problem, the following resources describe the state of a vehicle at the end of a forward partial path P from the source vertex s to a vertex $i \in V$, represented by a label $F_i = \left(F_i^{cost}, F_i^{load}, F_i^{time}, (F_i^{cust_n})_{n \in N}\right)$, where N is the set of all customers. The components of a label are:

F_i^{cost}: reduced cost of path P;

F_i^{load}: total load collected along path P;

F_i^{time}: earliest service start time at vertex i;

$F_i^{cust_n}$: number of times that customer $n \in N$ is visited along path P. The attribute is also set to 1 if customer n is unreachable from P (for details see Feillet *et al.*, 2004).

A backward label B_i describes the state of a vehicle at the start of a backward partial path P from a vertex $i \in V$ to the sink vertex t. Its components B_i^{cost} and $(B_i^{cust_n})_{n \in N}$ are defined as in the forward case, whereas the time-related resource B_i^{time} and the load-related resource B_i^{load} have a slightly different meaning:

B_i^{load}: residual vehicle capacity with respect to the collection along path P;

B_i^{time}: latest service start time at vertex i.

Details about the initial labels, the forward and backward propagation of the resources, the dominance between two forward or two backward labels, and the conditions for a concatenation of a forward and a backward partial path can be found in (Righini and Salani, 2006; Irnich, 2008). Note that it is generally not necessary to solve the described elementary version of the SPPRC pricing problem which is $\mathcal{NP}$-hard in the strong sense. Indeed, state-of-the-art approaches to most VRP variants rely on the so-called ng-path relaxation that allows certain non-elementarities. For the corresponding SPPRCs, the labeling algorithms have pseudo-polynomial complexity. Infeasibility with respect to

the elementarity condition of routes is then eliminated by branching or by dynamically enlarging the ng-neighborhood size (see Boland *et al.*, 2006; Righini and Salani, 2008; Baldacci *et al.*, 2012b). For necessary modifications of the resources $(F_i^{cust_n})_{n\in N}$ and $(B_i^{cust_n})_{n\in N}$ when using the ng-path relaxation see (Baldacci *et al.*, 2011).

Note that along a partial path the time-related and load-related resources are monotone. Along a forward path, earliest service start times and collected quantities are non-decreasing. Along a backward path, now considered in the direction from the sink vertex t to the source vertex s, latest service start times and residual capacities are non-increasing. Hence, both resources are suitable to define the monotone resource. If the time resource is chosen, the static half-way point is often set to the middle of the time horizon, i.e., $H^{time} = (a_s + b_t)/2$, where a_s is the earliest start time at the source vertex s and b_t is the latest arrival time at the sink vertex t. If the load resource is chosen, the static half-way point is typically set to $H^{load} = Q/2$, where Q is the vehicle capacity. The decision of using H^{time} or H^{load} should take into account whether time-window or capacity constraints are more restrictive in a given VRPTW instance. The more constrained the monotone resource is, the more effective is bidirectional labeling.

5.3.2 Electric Vehicle Routing Problem with Time Windows (EVRPTW)

The EVRPTW is a special variant of the VRPTW taking into account the limited driving range of electric vehicles and the possibility to recharge at recharging stations. As in the VRPTW, a set of customers must be visited exactly once during their service time windows by vehicles with a limited capacity. Because of rather short cruising ranges of electric vehicles and the restricted number of recharging stations, the refueling has to be considered explicitly with a non-negligible recharging time.

The problem was first mentioned by Schneider *et al.* (2014) who proposed an effective hybrid metaheuristic. The first exact branch-and-price algorithm for the EVRPTW was developed by Desaulniers *et al.* (2016). They consider four problem variants by distinguishing between a maximum of one recharge per route and multiple possible recharges as well as between the full and the partial recharge policies. The full recharge policy requires that the battery is always filled up completely at every recharge while the partial recharge policy allows any amount of electrical energy to be recharged. In the following, we focus on the problem variant with multiple full recharges (MF) that requires a difficult and the most asymmetric SPPRC pricing problem among these four variants.

In addition to the above mentioned SPPRC resources (F_i^{cost}, F_i^{load}, F_i^{time}, and $F_i^{cust_n}$) of the VRPTW, the following information has to be stored in a forward label F_i to determine the status of the vehicle routed along a forward partial path P from the source vertex s up to vertex $i \in V$:

F_i^{rch}: number of recharges performed along path P;

F_i^{rt}: cumulated required recharging time since the last recharge along path P (or since the beginning of P if P contains no recharge).

There are four types of vertices in the EVRTPW: the source vertex s, the sink vertex t, the customers $n \in N$, and the recharging stations $r \in R$. Moreover, there are two types of REFs f_{ij} depending on the type of vertex i from which the label is extended. Leaving a customer or depot vertex requires the same resource update as in the VRPTW and an increase of the required recharging time. However, when a recharging station is left, the required recharging time F_i^{rt} is added to the time resource F_j^{time} to represent the full recharge, F_j^{rt} is reset, and the recharge counter is increased by 1.

The backward propagation is more complicated, since the necessary recharging amount at a recharging station is not exactly known. It depends on the energy consumed chronologically before the current position, which is not yet determined when a recharging station is reached in a backward extension step. Thus, for describing a backward partial path P from a vertex $i \in V$ to the sink vertex t represented by a label B_i, the standard resources (B_i^{cost}, B_i^{load}, B_i^{time}, and $B_i^{cust_n}$) of the VRPTW are complemented by four other resources; two similar to the ones of the forward labeling and two new resources:

B_i^{nrch}: negative number of recharges performed along path P except at the last vertex i;

B_i^{rt}: cumulated required recharging time needed to recover the energy consumed up to the next recharging station r (or consumed along P if no recharge is performed);

B_i^{sl}: if a recharging station is visited along path P, this is the cumulated slack time at vertex i that can be used for recharging at the next recharging station without changing the latest service start time at i;

B_i^{avrt}: if a recharging station is visited along path P, this is the maximum additionally available recharging time at the next recharging station that ensures time window feasibility along P.

Backward REFs b_{ij} depend on whether a recharging station was visited on the path and on the type of vertex j. Further details such as the initial labels, the definition of all REFs, the feasibility checks, the dominance rules, and the merging conditions can be found in (Desaulniers *et al.*, 2016). Summarizing, the backward labeling uses two additional resources, has a weaker dominance than the forward labeling, and requires more computational effort.

As in the VRPTW, both the time and the load resource are non-decreasing along a forward path and non-increasing along a backward path. Therefore, they are suitable monotone resources for the half-way point definition.

5.3.3 Vehicle Routing and Truck Driver Scheduling Problem (VRTDSP)

The VRTDSP is the variant of the VRPTW that schedules the vehicles according to customer service time windows and hours of service regulations. Hours of service regulations for truck drivers have been imposed by many governments worldwide to ensure that break and rest periods are regularly taken. Transport companies have to take these into account and plan the routes and schedules of their truck drivers simultaneously. Recently, Goel and Irnich (2016) presented the first exact approach to the VRTDSP complying with the new U.S. hours of service regulations. They use a branch-and-price algorithm to solve the resulting VRTDSP. According to the current U.S. regulations, a driver may take break and rest periods at any time and with any duration larger than 30 minutes and ten hours, respectively. However, driving periods must be scheduled in accordance with the current state of the driver. On the contrary, no limitations regarding service activities are imposed by the U.S. regulations. The relevant parameters are summarized in Table 5.1.

Symbol	Value	Description
t^{drive}	11 hours	The maximum accumulated driving time between two consecutive rest periods
t^{break}	$\frac{1}{2}$ hours	The minimum duration of a break period
t^{rest}	10 hours	The minimum duration of a rest period
$t^{\text{el\|B}}$	8 hours	The maximum time after the end of the last break or rest period until which a driver may drive
$t^{\text{el\|R}}$	14 hours	The maximum time after the end of the last rest period until which a driver may drive

Table 5.1: Parameters imposed by the new U.S. hours of service regulations (Table 1 in Goel and Irnich, 2016)

The column-generation subproblem of the VRTDSP is an SPPRC that takes into account capacity and time-window constraints as well as hours of service regulations. In (Goel and Irnich, 2016), it is solved with a forward labeling algorithm in an auxiliary network where an intermediate vertex n_{ij} is created for every feasible arc $(i, j) \in A$. Five different REFs are then needed to model the activities of a vehicle and its driver on the trip from i to j, see Figure 5.2. The feasibility of a partial path strongly depends on the driver's current state. Therefore, the standard VRPTW resources ($F_i^{cost}, F_i^{load}, F_i^{time}$ and $(F_i^{cust_n})_{n \in N}$) of a forward label F_i associated with the partial path P from the source vertex s to a vertex $i \in V$ are supplemented by six additional resources:

F_i^{dist}: remaining driving time to reach the next customer;

F_i^{drive}: accumulated driving time since the last rest;

$F_i^{el|B}$: time elapsed since the last break;

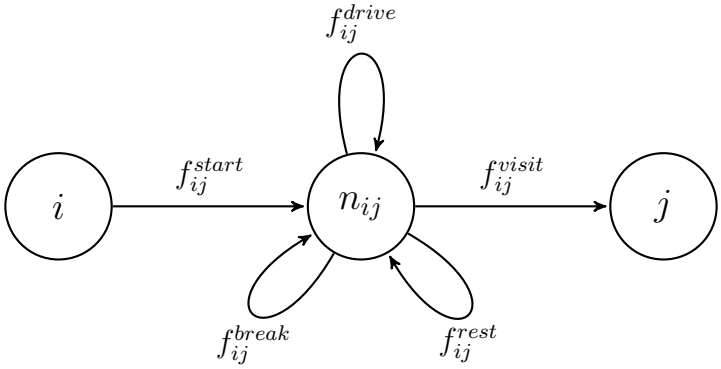

Figure 5.2: Subnetwork for arc $(i, j) \in A$ (similar to Figure 1 in Tilk, 2016)

$F_i^{el|R}$: time elapsed since the last rest;

$F_i^{la|B}$: latest possible point in time to which the end of the last break period can be extended without violating any resource constraints;

$F_i^{la|R}$: latest possible point in time to which the end of the last rest period can be extended without violating any resource constraints.

The five REFs can be summarized as follows: First, the REF f_{ij}^{start} models the start of the trip from i to j immediately after i was serviced. Second, the REF f_{ij}^{drive} implements a driving activity of Δ_{ij} time units within the trip from i to j. The amount of driving time can be computed as $\Delta_{ij} = \min\{F_i^{dist}, t^{\text{drive}} - F_i^{drive}, t^{\text{el|B}} - F_i^{el|B}, t^{\text{el|R}} - F_i^{el|R}\}$. It is the maximal driving time that does not violate any time-window or hours of service constraints. Third, taking a 30-minute break or a ten-hour rest on the trip from i to j is modeled with the REFs f_{ij}^{break} and f_{ij}^{rest}, respectively. Finally, performing the service at customer j is modeled with the help of the REF f_{ij}^{visit}. This REF also extends the length of the last rest and break period by any unavoidable waiting time.

Tilk (2016) refined this approach and presented, among other enhancements, a bidirectional labeling algorithm for the VRTDSP-specific SPPRC. The backward labeling is defined on a time-reversed network in which all time windows are inverted and arcs are reversed. This allows to use the same resources with the same meaning as in the forward labeling. However, the REFs need to be slightly altered in order to take into account that waiting and service times do not count as driving times and are, therefore, not restricted by the hours of service regulations. As waiting occurs only before and service after the start of a service time window, the forward and backward labeling are asymmetric due to this fixed chronological order. Details about the initial labels, the forward and backward propagation of the resources, the dominance rules, the feasibility conditions for concatenating a forward and a backward label, the adaption of the ng-path relaxation, and other acceleration techniques can be found in (Tilk, 2016).

Since the backward times are negative due to the inversion of the network, the time-related resource B^{time} is non-decreasing in the backward labeling, i.e., $B_i^{time} \geq B_j^{time}$

for a backward extension along arc $(i, j) \in A$. To nevertheless apply Algorithm 5.1, the following small modification is necessary: B^{time} has to be replaced by its negative value $B^{res} = -B^{time}$ in Steps 11 and 14. The load-related resource behaves as in the VRPTW, thus, both resources are suitable to define a half-way point.

5.4 Computational Results

This section reports the computational results on the three aforementioned VRP variants. All results were obtained using a standard PC with an Intel(R) Core(TM) i7-5930k 3.5 GHz processor and 64 GB of main memory. The algorithms were coded in C++ and compiled into 64-bit single-thread code with MS Visual Studio 2013. The callable library of CPLEX 12.6.0 was used for solving the restricted master programs (RMPs) in the column-generation algorithms. No branching is performed, since in most cases different branch-and-bound trees result when columns are generated by different SPPRC algorithms. This makes computation time for the same problem instance scatter more and the impact of using a dynamic half-way point cannot be determined accurately with the (relatively small) benchmark sets used in the literature. However, the general behaviour of a branch-and-price(-and-cut) does not differ much, on average, from the results we present here.

Pre-tests revealed that the time resource is better suited to define the half-way point than the load resource in all considered problem variants. We set $H^{time} = (a_s + b_t)/2$ to define the static half-way point and initialize the dynamic labeling algorithm by setting $H_F = b_t$ and $H_B = a_s$. Labels are processed in order of the monotone resource and ties are broken arbitrarily. The dominance algorithm is invoked every time the value of H_F or H_B changes. Moreover, we decide whether the next label to extend is a forward or a backward label by choosing the direction with the smaller number of *unprocessed* labels (forward if the numbers are identical). In pre-tests on reduced instance sets, this strategy has proven to be superior on average to the other strategies mentioned in Section 5.2 for the three considered VRP variants.

We run two different types of tests. In the first type of test, we solve every instance of the SPPRC pricing problem using both the static and the dynamic half-way point algorithm in each iteration of the linear relaxation of the master program. Since both versions are solved with identical reduced costs in each iteration, the comparison is not impacted by an otherwise different trajectory of the dual prices. Thus, we expect the dynamic-to-static ratios to be more stable leading to more reliable results. We compute the following values for each instance: the ratios dynamic to static of the number of generated labels, the number of processed labels, and the time spent in the pricing as well as the coefficient of variation (COV) of the dynamic half-way point. All ratios are computed as geometric means over all pricing iterations. We provide minimum, geometric mean, and maximum of these values, aggregated over the different instance groups. For the computation of the time ratio, we take into account only those iterations of the pricing problem that take more than 0.1 seconds in the static and the dynamic version. Herewith, the impact of inaccuracies in time measurement is reduced. Whenever

the linear relaxation is not completely solved within the time limit, the value is taken over the iterations solved by both versions up to this point.

In the second type of test, we compare the solution of the RMPs when either the dynamic or the static version of the half-way point is used in the pricing problems. For each instance group, the following values are reported: the number of successfully solved linear relaxations and the average solution time in seconds needed in the static and dynamic version, respectively. The average is taken only over the instances that were solved by both versions.

The following subsections report the results for the different VRP variants along with the settings and instances they were obtained with.

5.4.1 VRPTW Results

We test our column-generation algorithm for the VRPTW on the 56 benchmark instances of Solomon (1987). Additionally, we use the 60 instances with 200 customers proposed by Gehring and Homberger (2002). A CPU time limit of one hour is used for all computations. Furthermore, the ng-path relaxation with a neighborhood size of ten is applied, and networks with a reduced arc set are used as pricing heuristics.

The results of the first type of test, where labeling with static and dynamic half-way points is applied with identical reduced cost, are summarized in Table 5.2. The values are aggregated according to the instance group. In addition to the results for solving the linear relaxation, we present results when subset-row inequalities (SR) are added (Jepsen *et al.*, 2008). We restrict ourselves to those SR inequalities defined on subsets with three vertices as proposed by Jepsen *et al.* (2008). To limit the number of violated inequalities added to the RMP, we use the same cut-generation strategy with identical parameters as proposed by Desaulniers *et al.* (2008).

Instance group	Generated labels			Processed labels			Time			COV		
	min	avg	max	min	avg	max	min	avg	max	min	avg	max
Solomon	0.76	0.95	1.08	0.68	0.92	1.08	0.68	0.96	1.15	0.02	0.15	0.52
G & H	0.72	0.91	1.08	0.58	0.84	1.06	0.32	0.77	1.16	0.08	0.22	1.19
all		0.93			0.88			0.86			0.19	
Solomon (SR)	0.76	0.94	1.08	0.68	0.92	1.09	0.47	0.89	1.28	0.02	0.14	0.52
G & H (SR)	0.69	0.91	1.08	0.59	0.83	1.07	0.31	0.73	1.10	0.09	0.22	1.19
all (SR)		0.92			0.87			0.81			0.18	

Table 5.2: VRPTW: Ratios for using the dynamic and static half-way point with identical reduced cost

The results for solving the linear relaxation without cuts already indicate that the dynamic half-way point is superior. Only 93 % of the labels generated in the static version are generated in the dynamic version, and only 88 % of the labels processed in the static version are processed in the dynamic version. The average computation

time of the dynamic version is reduced to 86 % in relation to the static. The reported coefficient of variation shows that the values of the dynamic half-way points vary by 19 % on average among the pricing iterations of each VRP instance. Our interpretation of this result is that a computationally convenient half-way point differs from iteration to iteration of the pricing problem. Thus, any cleverly chosen but fixed static half-way point is nonetheless inferior in most iterations.

When SR inequalities are added to the RMP, the pricing problem becomes harder because more resources are involved and labels are less comparable in terms of dominance. The ratios for generated and processed labels decrease slightly, but the computation time ratio decreases by another five percent on average.

Instance group	No. solved		Avg time [s]	
	static	dynamic	static	dynamic
Solomon	56/56	56/56	118.64	111.24
G & H	46/60	49/60	674.32	466.51
all	102/116	105/116	369.24	271.46

Table 5.3: VRPTW: Comparison of computation times using static vs. dynamic half-way points

Table 5.3 shows the results for the second type of test, i.e., the comparison between static and dynamic half-way points for solving the linear relaxation. Herein, the results are also aggregated per instance group. No significant difference can be observed when solving the Solomon instances. However, the algorithm with the dynamic half-way point is able to solve three more linear relaxations of the Gehring & Homberger instances and the solution time decreases by around 30 %. This implies that the benefit of using a dynamic half-way point increases with the size and difficulty of the SPPRC instances.

5.4.2 EVRPTW Results

The experiments for the EVRPTW are performed on the 56 instances with 100 customers introduced by Schneider *et al.* (2014). We incorporated the small modifications suggested by Desaulniers *et al.* (2016). Smaller instances are created by considering only the first 25 and 50 customers. The CPU time limit is set to one hour per instance. As for the VRPTW, the *ng*-path relaxation with a neighborhood size of ten is applied, and heuristic pricing is implemented by using reduced networks.

The values are aggregated according to the characteristics of the instances. Characteristics of the instances are the customer distribution (C=clustered, R=random, and RC=both), the number of customers per instance (25, 50, and 100), and the length of the planning horizon (Series 1 and Series 2). *Series 1* contains the instance groups $C1$, $R1$, and $RC1$ with shorter planning horizon, whereas *Series 2* comprises the remaining instances of type $C2$, $R2$, and $RC2$ with longer planning horizon.

Instance group	Generated labels			Processed labels			Time			COV		
	min	avg	max	min	avg	max	min	avg	max	min	avg	max
25	0.59	0.87	1.00	0.55	0.89	1.12	0.11	0.80	1.09	0.08	0.25	0.60
50	0.65	0.89	1.00	0.71	0.90	1.11	0.15	0.76	1.13	0.09	0.23	0.61
100	0.40	0.84	1.02	0.46	0.87	1.06	0.05	0.72	1.43	0.06	0.25	0.54
Series 1	0.59	0.90	1.00	0.64	0.93	1.12	0.29	0.91	1.13	0.06	0.27	0.53
Series 2	0.40	0.83	1.02	0.46	0.84	1.03	0.05	0.62	1.43	0.07	0.22	0.61
all		0.87			0.89			0.76			0.25	

Table 5.4: EVRPTW: Ratios for using the dynamic and static half-way point with identical reduced cost

Table 5.4 summarizes the results of the first type of test, i.e., the direct comparison per pricing problem with identical reduced costs. Overall, the pricing times are reduced through the use of the dynamic half-way point by 24 %, the number of generated and processed labels is decreased by more than ten percent on average. The instances of the second series with longer routes profit significantly more from the use of the dynamic half-way point. Surprisingly, the average COV shows no direct correlation with the effectiveness of the dynamic half-way point. However, an average COV of 25 % indicates that the dynamic adaptation of the half-way points is beneficial.

We have also run the first type of test for the linear relaxation of the master program with SR inequalities. As the ratios decrease only slightly, we do not report details. It seems that, unlike for the VRPTW, the SPPRC pricing problem of the EVRPTW is already so hard that adding SR inequalities does not significantly increase its difficulty.

Instance group	No. solved		Avg time [s]	
	static	dynamic	static	dynamic
25	55/56	56/56	5.36	2.68
50	54/56	55/56	324.50	240.46
100	41/56	42/56	692.63	517.26
Series 1	87/87	87/87	137.14	116.48
Series 2	63/81	66/81	541.80	382.08
all	150/168	153/168	305.53	227.00

Table 5.5: EVRPTW: Comparison of computation times using static vs. dynamic half-way points

Table 5.5 shows the results for the second type of test. With the dynamic half-way point, the linear relaxation of three more instances is solved within the time limit. The additional instances are in the *Series 2* showing again that EVRPTW instances with

longer routes benefit more from the dynamic half-way point determination. Moreover, the solution times are reduced by 25 % on average.

5.4.3 VRTDSP Results

The VRTDSP experiments use the 56 benchmark instances proposed by Goel (2009), which can be obtained at `http://www.telematique.eu/research/downloads`. These 100-customer instances are derived from the VRPTW benchmark of Solomon (1987). Smaller instances are created by considering only the first 25 or 50 customers. We set a CPU time limit of two hours and use the *ng*-path relaxation with a neighborhood size of ten. Limited discrepancy search (Feillet *et al.*, 2007) and a heuristic dominance procedure are applied as heuristic pricing strategies (see also Goel and Irnich, 2016). Moreover, we stop Algorithm 5.1 as soon as the sum of negative reduced-cost labels at the sink in the forward labeling and the source in the backward labeling reaches 500.

The results of the first type of test are summarized in Table 5.6. The average dynamic-to-static ratio of the generated and processed labels is nearly identical around 0.87 and 0.74, respectively, independent of the instance sizes. The average computation time reduces to 63 % for the dynamic version in relation to the static. The value of the dynamic half-way point varies by 12 % on average over the solution of a single instance. A detailed analysis shows that the pricing problem is solved faster with the static half-way point in only eleven of the 168 instances, where none of these master programs takes longer than 20 seconds. Moreover, in these eleven instances, either the static is very close to the dynamic half-way point in all pricing iterations or the solution of each single pricing problem takes less than 0.2 seconds. As in the case of the EVRPTW, adding SR inequalities to the linear relaxation has no significant impact on the dynamic-to-static ratios.

Instance group	Generated labels			Processed labels			Time			COV		
	min	avg	max	min	avg	max	min	avg	max	min	avg	max
25	0.48	0.86	1.07	0.45	0.73	0.90	0.11	0.64	1.68	0.02	0.10	0.72
50	0.75	0.87	1.02	0.59	0.75	0.95	0.32	0.67	1.02	0.02	0.14	0.50
100	0.65	0.87	0.96	0.56	0.75	0.87	0.21	0.59	0.86	0.02	0.11	0.55
all		0.87			0.74			0.63			0.12	

Table 5.6: VRDTSP: Ratios for using the dynamic and static half-way point with identical reduced cost

Table 5.7 reports the result of the second type of test showing that again labeling with a dynamic half-way point is superior to the static version. Ten additional linear relaxations are solved, and the solution of a single instance takes on average approximately half of the time.

Instance group	No. solved		Avg time [s]	
	static	dynamic	static	dynamic
25	56/56	56/56	96.80	39.11
50	51/56	54/56	702.08	388.32
100	19/56	26/56	2123.98	1023.38
all	126/168	136/168	646.60	327.92

Table 5.7: VRDTSP: Comparison of computation times using static vs. dynamic half-way points

5.5 Conclusions

In this paper, we addressed the solution of different variants of the SPPRC with the help of bidirectional dynamic programming labeling algorithms. This type of algorithm is based on the definition of a so-called half-way point separating the forward from the backward labeling. In previous bidirectional labeling algorithms, the half-way point had to be specified a priori, before the labeling procedure starts. We exploit asymmetry in input data (constraints and reduced costs) as well as asymmetry in the forward and backward labeling process (different number of labels, bounding procedures, and dominance rules of different strength) by defining forward and backward half-way points that are dynamically adjusted according to an expected workload to be spent in forward and backward labeling. These forward and backward half-way points can be interpreted as upper and lower bounds of a possible static half-way point.

We have conducted extensive computational experiments with three types of VRPs where we compared the solutions of the column-generation SPPRC pricing problems using bidirectional labeling algorithms. For the VRPTW, replacing the static by the new dynamic half-way point reduces the solution times of SPPRCs by approximately 15 to 20 %. Comparing results with and without incorporating subset-row inequalities as well as Solomon's 100 customers and Gehring & Homberger's 200 customer instances, we found that the more difficult and larger VRPTW instances benefit more from a dynamic half-way point. For the EVRPTW, which has more SPPRC resources and is asymmetric in forward and backward labeling, the dynamic half-way point implementation on average reduced computation times by approximately 25 %. Also here, more difficult and larger SPPRC instances such as the second series with 100 customers profit more (38 % speedup on average). Finally, the VRTDSP is an example of a VRP with a large number of SPPRC resources and complex REFs. The experiments have revealed that on average the computation times are reduced by 37 % (41 % for the 100 customer instances).

The general insight of the experiments is the following: the harder the SPPRC instance, the more effective is the dynamic half-way point version. The necessary modifications needed to alter a bidirectional SPPRC labeling algorithm with static into one with dynamic half-way point are very minor. Thus, a significant effect can be achieved with little implementation effort.

Acknowledgement

We would like to thank Eduardo Uchoa from Pontifícia Universidade Católica (PUC) in Rio de Janeiro for pointing us to the works (Pecin, 2014; Pecin *et al.*, 2016).

References

Baldacci, R., Mingozzi, A., and Roberti, R. (2011). New route relaxation and pricing strategies for the vehicle routing problem. *Operations Research*, **59**(5), 1269–1283.

Baldacci, R., Mingozzi, A., and Roberti, R. (2012). Recent exact algorithms for solving the vehicle routing problem under capacity and time window constraints. *European Journal of Operational Research*, **218**, 1–6.

Bentley, J. (1980). Multidimensional divide-and-conquer. *Communications of the ACM*, **23**(4), 214–229.

Bode, C. and Irnich, S. (2014). The shortest-path problem with resource constraints with (k,2)-loop elimination and its application to the capacitated arc-routing problem. *European Journal of Operational Research*, **238**(2), 415–426.

Boland, N., Dethridge, J., and Dumitrescu, I. (2006). Accelerated label setting algorithms for the elementary resource constrained shortest path problem. *Operations Research Letters*, **34**(1), 58–68.

Contardo, C., Desaulniers, G., and Lessard, F. (2015). Reaching the elementary lower bound in the vehicle routing problem with time windows. *Networks*, **65**(1), 88–99.

Cordeau, J.-F., Desaulniers, G., Desrosiers, J., Solomon, M., and Soumis, F. (2002). VRP with time windows. In P. Toth and D. Vigo, editors, *The Vehicle Routing Problem*, chapter 7, pages 155–194. Society for Industrial and Applied Mathematics, Philadelphia, PA.

Desaulniers, G., Desrosiers, J., Ioachim, I., Solomon, M., Soumis, F., and Villeneuve, D. (1998). A unified framework for deterministic time constrained vehicle routing and crew scheduling problems. In T. Crainic and G. Laporte, editors, *Fleet Management and Logistics*, chapter 3, pages 57–93. Kluwer Academic Publisher, Boston, Dordrecht, London.

Desaulniers, G., Desrosiers, J., and Solomon, M., editors (2005). *Column Generation*. Springer, New York, NY.

Desaulniers, G., Lessard, F., and Hadjar, A. (2008). Tabu search, partial elementarity, and generalized k-path inequalities for the vehicle routing problem with time windows. *Transportation Science*, **42**(3), 387–404.

Desaulniers, G., Desrosiers, J., and Spoorendonk, S. (2010). The vehicle routing problem with time windows: State-of-the-art exact solution methods. In J. J. Cochran, L. A. Cox, P. Keskinocak, J. P. Kharoufeh, and J. C. Smith, editors, *Wiley Encyclopedia of Operations Research and Management Science*, volume 8, pages 5742–5749. John Wiley & Sons, Inc.

Desaulniers, G., Madsen, O. B., and Ropke, S. (2014). The vehicle routing problem with time windows. In Toth and Vigo (2014), chapter 5, pages 119–159.

Desaulniers, G., Errico, F., Irnich, S., and Schneider, M. (2016). Exact algorithms for electric vehicle-routing problems with time windows. *Operations Research*. Forthcoming.

Desrochers, M., Desrosiers, J., and Solomon, M. (1992). A new optimization algorithm for the vehicle routing problem with time windows. *Operations Research*, **40**(2), 342–354.

Drexl, M. (2012). Rich vehicle routing in theory and practice. *Logistics Research*, **5**(1), 47–63.

Feillet, D., Dejax, P., Gendreau, M., and Guéguen, C. (2004). An exact algorithm for the elementary shortest path problem with resource constraints: Application to some vehicle routing problems. *Networks*, **44**(3), 216–229.

Feillet, D., Gendreau, M., and Rousseau, L.-M. (2007). New refinements for the solution of vehicle routing problems with branch and price. *INFOR*, **45**(4), 239–256.

Garcia, R. (2009). *Resource Constrained Shortest Paths and Extensions*. Ph.D. thesis, School of Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta, U.S.A.

Gehring, H. and Homberger, J. (2002). Parallelization of a Two-Phase metaheuristic for routing problems with time windows. *Journal of Heuristics*, **8**(3), 251–276.

Goel, A. (2009). Vehicle scheduling and routing with drivers' working hours. *Transportation Science*, **43**(1), 17–26.

Goel, A. and Irnich, S. (2016). An exact method for vehicle routing and truck driver scheduling problems. *Transportation Science*. Forthcoming.

Horváth, M. and Kis, T. (2016). Solving resource constrained shortest path problems with lp-based methods. *Computers & Operations Research*, **73**, 150 – 164.

Irnich, S. (2008). Resource extension functions: Properties, inversion, and generalization to segments. *OR Spectrum*, **30**(1), 113–148.

Irnich, S. and Desaulniers, G. (2005). Shortest path problems with resource constraints. In Desaulniers *et al.* (2005), chapter 2, pages 33–65.

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

Irnich, S., Toth, P., and Vigo, D. (2014). *The Family of Vehicle Routing Problems*, chapter 1, pages 1–33. In Toth and Vigo (2014).

Jepsen, M., Petersen, B., Spoorendonk, S., and Pisinger, D. (2008). Subset-row inequalities applied to the vehicle-routing problem with time windows. *Operations Research*, **56**(2), 497–511.

Kohl, N., Desrosiers, J., Madsen, O., Solomon, M., and Soumis, F. (1999). 2-path cuts for the vehicle routing problem with time windows. *Transportation Science*, **33**(1), 101–116.

Lozano, L. and Medaglia, A. L. (2013). On an exact method for the constrained shortest path problem. *Computers & Operations Research*, **40**(1), 378 – 384.

Pecin, D. (2014). *Exact Algorithms for the Capacitated Vehicle Routing Problem*. Ph.D. thesis, Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro.

Pecin, D., Pessoa, A., Poggi, M., and Uchoa, E. (2016). Improved branch-cut-and-price for capacitated vehicle routing. *Mathematical Programming Computation*, pages 1–40.

Poggi, M. and Uchoa, E. (2014). *New Exact Algorithms for the Capacitated Vehicle Routing Problem*, chapter 3, pages 59–86. In Toth and Vigo (2014).

Pugliese, L. D. P. and Guerriero, F. (2013). A reference point approach for the resource constrained shortest path problems. *Transportation Science*, **47**(2), 247–265.

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

Righini, G. and Salani, M. (2008). New dynamic programming algorithms for the resource constrained elementary shortest path problem. *Networks*, **51**(3), 155–170.

Schneider, M., Stenger, A., and Goeke, D. (2014). The electric vehicle-routing problem with time windows and recharging stations. *Transportation Science*, **48**(4), 500–520.

Solomon, M. (1987). Algorithms for the vehicle routing and scheduling problem with time window constraints. *Operations Research*, **35**(2), 254–265.

Tilk, C. (2016). Branch-and-price-and-cut for the vehicle routing and truck driver scheduling problem. Technical Report LM-2016-04, Chair of Logistics Management, Gutenberg School of Management and Economics, Johannes Gutenberg University Mainz, Mainz, Germany.

Toth, P. and Vigo, D., editors (2014). *Vehicle Routing*. MOS-SIAM Series on Optimization. Society for Industrial and Applied Mathematics, Philadelphia, PA.

Chapter 6

Combined Column-and-Row-Generation for the Optimal Communication Spanning Tree Problem

Christian Tilk, Stefan Irnich

Abstract

This paper considers the exact solution of the optimal communication spanning tree problem (OCSTP), which can be described as follows: Given an undirected graph with transportation costs on every edge and communication requirements for all pairs of vertices, the OCSTP seeks for a spanning tree that minimizes the sum of the communication costs between all pairs of vertices, where the communication cost of a pair of vertices is defined as their communication requirement multiplied by the transportation cost of the unique tree path that connects the two vertices. Two types of compact formulations for OCSTP were presented in the literature. The first one is a four-index model based on a path formulation. The second one is a three-index model in which a solution is an intersection of spanning trees, each rooted at a different vertex of the graph and modeled using a flow formulation for spanning tree problems. We present Dantzig-Wolfe reformulations for both compact models to be used in a combined column-and-row-generation algorithm. In the path-based reformulation, the pricing problems are simple shortest-path problems, one for each pair of vertices with a positive communication requirement. The pricing problems of the tree-based reformulation are fixed-cost network flow problems, one for each vertex of the graph. We apply different heuristic and exact methods for pricing and present optimal solutions for benchmark instances with up to 50 vertices.

6.1 Introduction

The *optimal communication spanning tree problem* (OCSTP) was first described by Hu (1974): Given an undirected graph with transportation costs on every edge and communication requirements for all pairs of vertices, the OCSTP seeks for a spanning tree that minimizes the sum of the communication costs between all pairs of vertices, where

the communication cost of a pair of vertices is defined as their communication requirement multiplied by the transportation cost of the unique tree path that connects the two vertices. Johnson *et al.* (1978) have shown that the OCSTP is $\mathcal{NP}$-hard. Besides a broad range of applications in designing telecommunication and transportation networks, the OCSTP occurs as a subproblem in some network hub location problems (see Contreras and Fernández, 2012).

Two types of compact formulations for OCSTP were presented in the literature. The first one is a four-index model based on a path formulation (also described as a multi-commodity flow model), which was introduced and used by Contreras *et al.* (2010) in a Lagrangian relaxation approach. The second is the three-index model by Fernández *et al.* (2013) in which a solution is an intersection of spanning trees, each rooted at a different vertex of the graph and modeled using a flow formulation for spanning tree problems.

The contribution of the paper at hand is the presentation of new and effective exact algorithms for the OCSTP. Our starting point is the four-index (path-based) and three-index (tree-based) models, for which we derive Dantzig-Wolfe reformulations. Both reformulations have a huge number of variables and constraints. Therefore, we develop a combined column-and-row-generation algorithm for their resolution. In the path-based reformulation, the pricing problems are simple shortest-path problems, one for each pair of vertices with a positive communication requirement. The pricing problems of the tree-based reformulation are fixed-cost network flow problems, one for each vertex of the graph.

Our goal is the comparison of both reformulations to decide which of the column-and-row-generation algorithms is more effective from a computational point of view. We think that the comparison is also interesting from a theoretical point of view: We are able to rank the linear relaxations of all four formulations (path vs. tree; compact vs. extensive (re)formulation). In the path-based reformulation, the shortest-path models used for pricing have the integrality property, while in the tree-based reformulation no integral model is available for the pricing problems because they are $\mathcal{NP}$-hard. Hence, there is the tradeoff that the former subproblems can be solved much faster while the latter generally lead to a stronger linear relaxation (Lübbecke and Desrosiers, 2005).

We combine several algorithmic techniques in order to accelerate the solution process. We use partial column generation to reduce the number of pricing problems to solve in each iteration of the column-generation algorithm (see Gamache *et al.*, 1999). Moreover, for the tree-based reformulation, we show that the solution of any single pricing problem allows us to generate columns for all other pricing problems. At the same time, this procedure provides feasible solutions and upper bounds on the OCSTP in each iteration.

In order to finally compute optimal integer solutions, both column-and-row-generation algorithms are integrated into branch-and-bound. In addition, cycle-elimination inequalities can strengthen the path-based reformulation. The resulting branch-and-price-and-cut algorithms often quickly yield integer optimal solutions or otherwise tight lower and upper bounds. The remaining integrality gap is typically small and even for 50-vertex instances often around 5 %. We present computational results for different variants of the branch-and-price-and-cut algorithms and show the usefulness of the proposed methods.

The paper is structured as follows: Section 6.2 briefly surveys exact solution algorithms for the OCSTP. The formal definition of the OCSTP, the compact four-index model of Contreras *et al.* (2010), the compact three-index model of Fernández *et al.* (2013), and a comparison of both linear relaxations are presented in Section 6.3. In Section 6.4, we describe the new extensive formulations to be solved with column-and-row-generation algorithms and compare their LP-relaxations. Moreover, we formalize the pricing problems and discuss algorithms for their effective solution. Section 6.5 reports computational results. The paper closes with conclusions drawn in Section 6.6.

6.2 Literature

There exist several heuristic approaches to solve the OCSTP, e.g., with evolutionary algorithms, see for instance (Fischer and Merz, 2007) and (Steitz and Rothlauf, 2012). Approximation algorithms were presented by Wu *et al.* (2000), Sharma (2006), and Peleg and Reshef (1998).

Only a few exact approaches have been published: Ahuja and Murty (1987) proposed a combinatorial branch-and-bound that is able to solve instances on sparse graphs with up to 40 vertices. Rothlauf (2009) investigated structural properties of optimal OCSTP solutions and heuristics exploiting this knowledge. Contreras *et al.* (2010) directly solved the four-index model for instances with up to 40 vertices and presented a Lagrangian relaxation-based algorithm to compute tight lower and upper bounds for larger instances. Recently, Fernández *et al.* (2013) introduced a three-index, flow-based formulation for the OCSTP. They also presented classes of valid inequalities and sketched some computational results obtained when the three-index formulation is directly solved with a MIP solver. Fischetti *et al.* (2002) presented a column-and-row-generation algorithm for the minimum routing cost tree problem. This problem is a special case of the OCSTP, where the communication requirement between all pairs of vertices is one (or identical). They analyzed different formulations and reported optimal solutions for instances with up to 50 vertices.

6.3 Compact Formulations

We start with the formal definition of the OCSTP. Let $G = (V, E)$ be a simple undirected graph with vertices $V = \{1, 2, \ldots, n\}$ and edges E. We define $n = |V|$ and $m = |E|$. A *transportation cost* c_{ij} is associated with each edge $\{i, j\} \in E$ and a *communication requirement* r_{ij} with each pair $(i, j) \in V \times V$. Any spanning tree $T = (V, E_T)$ with $E_T \subset E$ is a feasible solution to the OCSTP. Recall that it is necessary and sufficient that $|E_T| = n - 1$ and $V = V(E_T)$ holds. The *communication cost of a tree* T is $\sum_{(s,t)\in V\times V} r_{st} \sum_{(i,j)\in P_{st}} c_{ij}$, where P_{st} is the unique path connecting s and t in T. The objective of the OCSTP is to find a spanning tree with minimum communication cost.

For simplicity, we assume that all vertices $u \in V$ have at least some positive communication requirement, i.e., $\sum_{i \in V}(r_{iu} + r_{ui}) > 0$. Since G is undirected, there exist several possibilities to split the communication requirement between (s,t) and $(t,s) \in V \times V$. Let $o_{st} = r_{st} + r_{ts}$ be the total communication requirement between vertices s and t (in both directions). One obtains an equivalent OCSTP instance by redefining $r_{st} := (o_{st} - k_{st})/2$ and $r_{ts} := (o_{st} + k_{st})/2$ for each k_{st} with $0 \leq k_{st} \leq o_{st}$. For $k_{st} = 0$, a symmetric communication requirement $r_{st} = r_{ts}$ results. The fully asymmetric case results from setting $k_{st} = o_{st}$ for all $s,t \in V$ with $s > t$ resulting in $r_{st} = 0$ and $r_{ts} = o_{st}$. We exploit the above equivalence and use the fact that asymmetric instances are typically easier to solve, see Section 6.5.

Next we present the two compact formulations from the literature and compare their linear relaxations. Both formulations utilize the complete directed graph $D = (V, A)$ underlying G with arc set $A = \{(i,j) \in V \times V : \{i,j\} \in E\}$. Note that A contains for each arc $(i,j) \in A$ also its antiparallel counterpart $(j,i) \in A$. For each vertex $u \in V$, $\delta^+(u)$ denotes its *forward star* and $\delta^-(u)$ its *backward star* in D. Moreover, let $R := \{(s,t) \in V \times V : r_{st} > 0\}$ be the set of all pairs of vertices with a positive communication requirement.

6.3.1 Path-based Formulation

The formulation IP^{comp}_{path} by Contreras *et al.* (2010) exploits the fact that the communication requirement between a pair of vertices is satisfied by a path between them. The *communication cost* of such a path is defined as the sum of the transportation costs of the path multiplied by the communication requirement between its extremities. To fulfill the tree requirement, the overall number of arcs used in all paths must be limited to $n-1$.

Two types of variables are used in the path-based formulation. Continuous variables f_{stij} are defined for each pair $(s,t) \in R$ and each arc $(i,j) \in A$ (four indices). A positive value of f_{stij} indicates that arc $(i,j) \in A$ is used for fulfilling the communication requirement between s and t, $(s,t) \in R$. Binary variables x_{ij} are defined for each edge $\{i,j\} \in E$ and $x_{ij} = 1$ indicates that the edge is in the spanning tree. The path-based formulation reads as follows:

$$z^{comp}_{path} = \min \quad \sum_{(s,t) \in R} r_{st} \sum_{(i,j) \in A} c_{ij} f_{stij} \tag{6.1a}$$

$$\text{s.t.} \quad \sum_{(s,j) \in \delta^+(s)} f_{stsj} = 1 \qquad \forall (s,t) \in R \tag{6.1b}$$

$$\sum_{(i,j) \in \delta^+(i)} f_{stij} - \sum_{(h,i) \in \delta^-(i)} f_{sthi} = 0 \qquad \forall (s,t) \in R; i \in V \setminus \{s,t\} \tag{6.1c}$$

$$f_{stij} + f_{stji} \leq x_{ij} \qquad \forall (s,t) \in R; \{i,j\} \in E \tag{6.1d}$$

$$\sum_{\{i,j\} \in E} x_{ij} = n - 1 \tag{6.1e}$$

$$x_{ij} \in \{0,1\} \qquad \forall \{i,j\} \in E \qquad (6.1f)$$
$$f_{stij} \geq 0 \qquad \forall (i,j) \in A; (s,t) \in R \qquad (6.1g)$$

The objective (6.1a) minimizes the communication cost of the resulting tree. Constraints (6.1b) and (6.1c) are the flow-conservation constraints of the path between s and t for each positive communication requirement $r_{st} > 0$. The corresponding constraint that the flow into t is one, i.e., $\sum_{(i,t)\in\delta^-(t)} f_{stit} = 1$, is redundant and therefore omitted. Constraints (6.1d) couple the f-variables with the associated x-variables. Finally, constraint (6.1e) ensures that the flow of each communication requirement can be fulfilled by the same spanning tree. The variables and their domains are given by (6.1f) and (6.1g).

6.3.2 Tree-based Formulation

The idea of the three-index formulation IP^{comp}_{tree} by Fernández *et al.* (2013) is that each vertex $u \in V$ distributes the *commodity* u to all other vertices $V \setminus \{u\}$. The demand of vertex $t \in V$ for commodity u is r_{ut}. The flow of each commodity u induces a tree rooted at vertex u with communication cost $\sum_{t\in V\setminus\{u\}} r_{ut} \sum_{(i,j)\in P_{ut}} c_{ij}$, where P_{ut} is the tree path connecting u and t. The model seeks for a spanning tree that minimizes the sum of the communication costs of all commodities.

The three-index formulation uses three types of variables: Binary variables y_{uij} are defined for vertices $u \in V$ and arcs $(i,j) \in A$. The value $y_{uij} = 1$ indicates that arc (i,j) is used for distributing commodity u. Continuous variables f_{uij} give the amount of flow of commodity u on arc (i,j). Binary variables x_{ij} are defined for each edge $\{i,j\} \in E$ and indicate whether the edge is in the spanning tree. The compact three-index formulation is:

$$z^{comp}_{tree} = \min \quad \sum_{u\in V} \sum_{(i,j)\in A} c_{ij} f_{uij} \qquad (6.2a)$$

$$\text{s.t.} \quad \sum_{(u,j)\in\delta^+(u)} f_{uuj} = \sum_{t\in V\setminus\{u\}} r_{ut} \qquad \forall u \in V \qquad (6.2b)$$

$$\sum_{(i,j)\in\delta^+(i)} f_{uij} - \sum_{(h,i)\in\delta^-(i)} f_{uhi} = -r_{ui} \qquad \forall u \in V; i \in V \setminus \{u\} \qquad (6.2c)$$

$$f_{uij} \leq M_u y_{uij} \qquad \forall u \in V; (i,j) \in A \qquad (6.2d)$$

$$\sum_{(i,j)\in A} y_{uij} \leq n - 1 \qquad \forall u \in V \qquad (6.2e)$$

$$y_{uij} + y_{uji} \leq x_{ij} \qquad \forall u \in V; \{i,j\} \in E \qquad (6.2f)$$

$$\sum_{\{i,j\}\in E} x_{ij} = n - 1 \qquad (6.2g)$$

$$x_{ij} \in \{0,1\} \qquad \forall \{i,j\} \in E \qquad (6.2h)$$

$$y_{uij} \in \{0,1\}, f_{uij} \geq 0 \qquad \forall u \in V; (i,j) \in A \qquad (6.2i)$$

The objective (6.2a) minimizes the sum of the communication cost of all commodities. Constraints (6.2b) and (6.2c) are the flow conservation constraints of every commodity at every vertex. Constraints (6.2d) link the y-variables with the corresponding f-variables using appropriately defined big numbers $M_u > 0$, one for each $u \in V$. We have chosen $M_u := \sum_{(u,j)\in R} r_{uj}$ which makes the constraints tight. Constraints (6.2f) couple pairs of the y-variables with the associated x-variable. The cardinality constraints (6.2e) ensure that the flow of commodity u forms a tree. Finally, constraint (6.2g) together with (6.2f) ensures that the flow of each commodity can be expressed by the same spanning tree. The variables and their domains are given by (6.2h) and (6.2i).

6.3.3 Comparison of Linear Relaxations

Both formulations IP^{comp}_{path} and IP^{comp}_{tree} are valid for the OCSTP and, hence, provide identical integer objective values. The linear relaxations LP^{comp}_{path} of the path-based formulation (providing a lower bound $\underline{z}^{comp}_{path}$) is not weaker than the linear relaxation LP^{comp}_{tree} of tree-based formulation (with lower bound $\underline{z}^{comp}_{tree}$).

Proposition 1. *For any OCSTP instance, $\underline{z}^{comp}_{tree} \leq \underline{z}^{comp}_{path}$ holds.*

Proof. We show that every feasible solution of LP^{comp}_{path} can be transformed into a feasible solution of LP^{comp}_{tree} with identical communication costs. Let (x^P, f^P) be a feasible solution of LP^{comp}_{path} with objective value $\underline{z}^{comp}_{path}$. Define (x^T, y^T, f^T) by

$$\begin{aligned} x^T_{ij} &:= x^P_{ij} && \{i,j\} \in E, \\ y^T_{uij} &:= \Big(\sum_{\substack{t\in V:\\ (u,t)\in R}} f^P_{utij} r_{ut}\Big)/M_u && u \in V, (i,j) \in A \\ f^T_{uij} &:= \sum_{\substack{t\in V:\\ (u,t)\in R}} f^P_{utij} r_{ut} && u \in V, (i,j) \in A. \end{aligned}$$

Then, (x^T, y^T, f^T) is a solution of LP^{comp}_{tree} with identical objective value $\underline{z}^{comp}_{tree} = \underline{z}^{comp}_{path}$ because the objective (6.2a) is:

$$\begin{aligned} \sum_{u\in V}\sum_{(i,j)\in A} c_{ij} f^T_{uij} &= \sum_{u\in V}\sum_{(i,j)\in A} c_{ij} \sum_{\substack{t\in V:\\ (u,t)\in R}} f^P_{utij} r_{ut} \\ &= \sum_{(i,j)\in A} c_{ij} \sum_{(u,t)\in R} f^P_{utij} r_{ut} = \sum_{(u,t)\in R} r_{ut} \sum_{(i,j)\in A} c_{ij} f^P_{utij} \end{aligned}$$

where the latter term is the objective (6.1a).

Similarly, we show that inserting (x^T, y^T, f^T) into the constraints of model LP^{comp}_{tree} and transforming the expressions always results in a true statement. First, inserting (x^T, y^T, f^T) into (6.2b) yields for all $u \in V$

$$\sum_{(u,j)\in\delta^+(u)} f^T_{uuj} = \sum_{(u,t)\in R} r_{ut} \Leftrightarrow \sum_{(u,j)\in\delta^+(u)} \sum_{(u,t)\in R} f^P_{utuj} r_{ut} = \sum_{(u,t)\in R} r_{ut}$$

$$\Leftrightarrow \sum_{(u,j)\in\delta^+(u)} \sum_{(u,t)\in R} f^P_{utuj} = 1,$$

which is (6.1b) and hence true.

Next, inserting into (6.2c) gives for all $u \in V$ and all $i \in V \setminus \{u\}$:

$$\begin{aligned}
&\sum_{(i,j)\in\delta^+(i)} f^T_{uij} - \sum_{(h,i)\in\delta^-(i)} f^T_{uhi} = -r_{ui} \\
\Leftrightarrow\ &\sum_{\substack{t\in V:\\(u,t)\in R}} \sum_{(i,j)\in\delta^+(i)} f^P_{utij} r_{ut} - \sum_{\substack{t\in V:\\(u,t)\in R}} \sum_{(h,i)\in\delta^-(i)} f^P_{uthi} r_{ut} = -r_{ui} \\
\Leftrightarrow\ &\sum_{\substack{t\in V:\\(u,t)\in R}} r_{ut} \underbrace{\left(\sum_{(i,j)\in\delta^+(i)} f^P_{utij} - \sum_{(h,i)\in\delta^-(i)} f^P_{uthi} \right)}_{=(*)} = -r_{ui},
\end{aligned}$$

where the term $(*)$ is -1 for $i = t$, and zero otherwise, due to the flow conservation constraints (6.1b) and (6.1c) for paths. Hence, the true statement $-r_{ui} = -r_{ui}$ results.

Inserting (x^T, y^T, f^T) into (6.2d) yields for all $u \in V; (i,j) \in A$:

$$f^T_{uij} \le M y^T_{uij} \Leftrightarrow \sum_{\substack{t\in V:\\(u,t)\in R}} f^P_{utij} r_{ut} \le M_u \frac{\sum\limits_{\substack{t\in V:\\(u,t)\in R}} f^P_{utij} r_{ut}}{M_u} = \sum_{\substack{t\in V:\\(u,t)\in R}} f^P_{utij} r_{ut},$$

which is obviously fulfilled.

Inserting (x^T, y^T, f^T) into (6.2e) gives for all $u \in V$:

$$\sum_{(i,j)\in A} y^T_{uij} \le n-1 \Leftrightarrow \sum_{(i,j)\in A} \frac{\sum\limits_{\substack{t\in V:\\(u,t)\in R}} f^P_{utij} r_{ut}}{M_u} \le \sum_{(i,j)\in A} \frac{x^P_{ij}\left(\sum\limits_{\substack{t\in V:\\(u,t)\in R}} r_{ut}\right)}{M_u} = \sum_{(i,j)\in A} x^P_{ij} \le n-1,$$

where on the right-hand side the first inequality results from $f^P_{utij} \le x_{ij}$ (see (6.1d)), the equality in the middle uses the definition of M_u, and the last inequality is fulfilled due to (6.1e).

Finally, inserting (x^T, y^T, f^T) into the left-hand side of (6.2f) for all $\forall u \in V$ and $\{i,j\} \in E$ leads to:

$$y^T_{uij} + y^T_{uji} = \frac{\sum\limits_{\substack{t\in V:\\(u,t)\in R}} f^P_{utij} r_{ut} + \sum\limits_{\substack{t\in V:\\(u,t)\in R}} f^P_{utji} r_{ut}}{M_u} = \sum_{\substack{t\in V:\\(u,t)\in R}} \frac{\overbrace{\left(f^P_{utij} + f^P_{utji}\right)}^{\le x^P_{ij} \text{ due to (6.1d)}} r_{ut}}{M_u} \le x^P_{ij} \frac{M_u}{M_u} = x^P_{ij} = x^T_{ij},$$

which shows inequality (6.2f). This completes the proof. □

6.4 Column-and-Row Generation and Branch-and-Price-and-Cut

In this section, we first derive Dantzig-Wolfe reformulations of both models (6.1) and (6.2) and compare their LP-relaxations. Then, we present the column-and-row generation problems and algorithms for their solution. Finally, we discuss branching and cutting in the overall branch-and-price-and-cut algorithms.

6.4.1 Dantzig-Wolfe Reformulation of the Path-based Formulation

The path-based formulation (6.1) can be decomposed by communication requirements $(s,t) \in R$ so that all f_{stij}-variables for a fixed (s,t) form one block. For each $(s,t) \in R$, constraints (6.1b), (6.1c), and (6.1g) describe an s-t-path in $D = (V,A)$. Using this decomposition into blocks, the Dantzig-Wolfe reformulation replaces the f_{stij} variables by variables representing s-t-paths, keeps the x_{ij}-variables and the constraints (6.1e)–(6.1f) in the master problem, and reformulates (6.1d) with the new path variables. Let $w(s,t)$ be the number of s-t-paths in D so that they can be indexed by $p \in P(s,t) := \{1,2,\ldots,w(s,t)\}$. Hence, there is one continuous variable λ^p_{st} for $p \in P(s,t)$, i.e, for every possible s-t-path. The set of these paths is given by $\{(f^p_{stij}) : p \in P(s,t)\}$. Herein, f^p_{stij} indicates if the path $p \in P(s,t)$ uses arc $(i,j) \in A$ or not. Moreover, let c^p_{st} be the communication cost of the pth path. Then, the *integer master problem* of the path-based formulation (in the following denoted by IP^{ext}_{path}) is:

$$z^{ext}_{path} = \min \quad \sum_{(s,t)\in R} \sum_{p\in P(s,t)} c^p_{st}\lambda^p_{st} \tag{6.3a}$$

$$\text{s.t.} \quad \sum_{p\in P(s,t)} (f^p_{stij} + f^p_{stji})\lambda^p_{st} \leq x_{ij} \qquad \forall (s,t) \in R; \{i,j\} \in E \tag{6.3b}$$

$$\sum_{\{i,j\}\in E} x_{ij} = n-1 \tag{6.3c}$$

$$x_{ij} \in \{0,1\} \qquad \forall \{i,j\} \in E \tag{6.3d}$$

$$\sum_{p\in P(s,t)} \lambda^p_{st} = 1 \qquad \forall (s,t) \in R \tag{6.3e}$$

$$\lambda^p_{st} \geq 0 \qquad \forall (s,t) \in R; p \in P(s,t) \tag{6.3f}$$

The objective (6.3a) minimizes the communication costs of paths for all communication requirements $(s,t) \in R$. Constraints (6.3b)–(6.3d) are (identical) reformulations of (6.1d)–(6.1f). Finally, (6.3e) are the convexity constraints forcing the selection of exactly one s-t-path for each $(s,t) \in R$, while (6.3f) describe the domain of the path variables.

The reformulated model (6.3) is identical to the model presented by Fischetti *et al.* (2002) for the minimum routing-cost-tree problem except that the coefficients c^p_{st} here model communication costs for generally non-unary communication requirements.

6.4.2 Dantzig-Wolfe Reformulation of the Tree-based Formulation

The tree-based formulation (6.2) can be decomposed by commodities $u \in V$ so that the variables y_{uij} and f_{uij} for a fixed u form a block. For each $u \in V$, the constraints (6.2b)–(6.2e) and (6.2i) describe a directed spanning tree rooted at vertex u together with flows satisfying the communication requirements for commodity u. Using this definition of blocks, the Dantzig-Wolfe reformulation replaces the variables y_{uij} and f_{uij} by variables representing the spanning trees with associated flows and reformulates constraints (6.2f)–(6.2h) in these new variables. Let $\ell(u)$ be the number of directed spanning trees rooted at u. Moreover, we define the index set $Q(u) := \{1, 2, \ldots, \ell(u)\}$ so that the set of all spanning trees rooted at u with flows can be described by $\{(f^q_{uij}, y^q_{uij}) : q \in Q(u)\}$. Herein, y^q_{uij} denotes if the tree $q \in Q(u)$ contains arc $(i,j) \in A$ or not and f^q_{uij} specifies the amount of flow on arc (i,j) for the tree $q \in Q(u)$. Let the communication cost of the qth spanning tree be c^q_u. The *integer master problem* of the tree-based formulation (in the following denoted by IP^{ext}_{tree}) is then:

$$z^{ext}_{tree} = \min \quad \sum_{u \in V} \sum_{q \in Q(u)} c^q_u \lambda^q_u \tag{6.4a}$$

$$\text{s.t.} \quad \sum_{q \in Q(u)} (y^q_{uij} + y^q_{uji}) \lambda^q_u \leq x_{ij} \qquad \forall u \in V; \{i,j\} \in E \tag{6.4b}$$

$$\sum_{\{i,j\} \in E} x_{ij} = n - 1 \tag{6.4c}$$

$$x_{ij} \in \{0,1\} \qquad \{i,j\} \in E \tag{6.4d}$$

$$\sum_{q \in Q(u)} \lambda^q_u = 1 \qquad \forall u \in V \tag{6.4e}$$

$$\lambda^q_u \geq 0 \qquad \forall u \in V; q \in Q(u) \tag{6.4f}$$

The objective (6.4a) calls for the minimization of the overall communication costs of all commodities. Constraints (6.4b)–(6.4d) are (identical) reformulations of the constraints (6.2f)–(6.2h). The convexity constraints (6.4e) require the selection of exactly one spanning tree for each commodity. The domain of the spanning tree variables is given by (6.4f). Note that the big numbers M_u are removed from the reformulation and will be handled in the pricing problems.

6.4.3 Comparison of the LP-Relaxation

As in the case of the compact formulations, both extensive formulations provide identical integer objective values, but the lower bounds $\underline{z}^{ext}_{path}$ and $\underline{z}^{ext}_{tree}$, respectively, provided by solving their LP-relaxations may differ.

Proposition 2. *For any OCSTP instance,* $\underline{z}^{ext}_{path} \leq \underline{z}^{ext}_{tree}$ *holds.*

Proof. Let (λ^q, x^q) be a feasible solution of LP^{ext}_{tree}. We have to prove that there exists an equivalent solution (λ^p, x^p) to LP^{ext}_{path}, i.e., one with the identical costs.

For the sake of convenience, we write $p \in q$ for $p \in P(s,t)$ and $q \in Q(s)$ if and only if $y^q_{sij} = 1$ for all $(i,j) \in p$ (the s-t path p is contained in the tree q routed at s). Note that by definition of c^q_s for any $q \in Q(s)$ we have

$$c^q_s = \sum_{\substack{t \in V \setminus \{s\}: \\ (s,t) \in R}} \sum_{\substack{p \in P(s,t): \\ p \in q}} c^p_{st}.$$

Define (λ^p, x^p) by

$$x^p_{ij} := x^q_{ij} \qquad \{i,j\} \in E, \tag{6.5a}$$

$$\lambda^p_{st} := \sum_{\substack{q \in Q(s): \\ p \in q}} \lambda^q_s \qquad (s,t) \in R : p \in P(s,t). \tag{6.5b}$$

It follows

$$\begin{aligned}
\sum_{(s,t) \in R} \sum_{p \in P(s,t)} c^p_{st} \lambda^p_{st} &= \sum_{(s,t) \in R} \sum_{p \in P(s,t)} c^p_{st} \sum_{\substack{q \in Q(s): \\ p \in q}} \lambda^q_s \\
&= \sum_{s \in V} \sum_{q \in Q(s)} \sum_{\substack{t \in V \setminus \{s\}: \\ (s,t) \in R}} \sum_{\substack{p \in P(s,t): \\ p \in q}} c^p_{st} \lambda^q_s \\
&= \sum_{s \in V} \sum_{q \in Q(s)} \left(\sum_{\substack{t \in V \setminus \{s\}: \\ (s,t) \in R}} \sum_{\substack{p \in P(s,t): \\ p \in q}} c^p_{st} \right) \lambda^q_s = \sum_{s \in V} \sum_{q \in Q(s)} c^q_s \lambda^q_s
\end{aligned}$$

where the first line has the objective (6.3a) on the left-hand side, the right-hand side results from (6.5b), the second line is a rearrangement of the terms, and the third line results from the definition of c^q_s leading to the objective (6.4a).

Inserting λ^p_{st} into the left-hand-side of (6.3b) gives for all $(s,t) \in R$ and $\{i,j\} \in E$:

$$\begin{aligned}
\sum_{p \in P(s,t)} (f^p_{stij} + f^p_{stji}) \lambda^p_{st} &= \sum_{p \in P(s,t)} (f^p_{stij} + f^p_{stji}) \sum_{\substack{q \in Q(s): \\ p \in q}} \lambda^q_s \\
&= \sum_{q \in Q(s)} \sum_{\substack{p \in P(s,t): \\ p \in q}} (f^p_{stij} + f^p_{stji}) \lambda^q_s \leq \sum_{q \in Q(s)} \sum_{\substack{p \in P(s,t): \\ p \in q}} (y^q_{sij} + y^q_{sji}) \lambda^q_s \leq x^q_{ij} = x^p_{ij}
\end{aligned}$$

i.e., the validity of (6.3b). Note that the first equality is the definition (6.5b), the second equality is a rearrangement of terms, the first inequality results from the fact that $f^p_{stij} \leq y^q_{sij}$ for all $p \in P(s,t)$, $q \in Q(s)$ with $p \in q$, the second inequality results from (6.4b) and the fact that (λ^q, x^q) is a feasible solution for (6.4), and the last equality is valid by definition (6.5a).

The proof that the remaining constraints (6.3c) and (6.3e) hold is straightforward. □

Proposition 3. *For any OCSTP instance,* $\underline{z}^{comp}_{tree} \leq \underline{z}^{comp}_{path} = \underline{z}^{ext}_{path} \leq \underline{z}^{ext}_{tree}$ *holds.*

Proof. Since the path formulation of the pricing problem of LP^{ext}_{path} has the integrality property, it follows $\underline{z}^{comp}_{path} = \underline{z}^{ext}_{path}$ (Lübbecke and Desrosiers, 2005). The proof now follows immediately from Propositions 1 and 2. □

6.4.4 Combined Column-and-Row Generation

For solving the mixed-integer programs (6.3) and (6.4) we employ a branch-and-price-and-cut algorithm (Lübbecke and Desrosiers, 2005; Desaulniers *et al.*, 2005), meaning that first integrality is relaxed, the respective linear relaxations are then solved with combined column-and-row generation algorithms, and integrality is finally established by integrating the procedure into a branch-and-bound scheme with an optional cutting-plane procedure.

In the following, the linear relaxations of formulations (6.3) and (6.4) are denoted as the *master programs* of the path-based and tree-based formulation, respectively. They do not contain the integer constraints (6.3d) and (6.4d). Moreover, the *restricted master problems* for the path-based formulation (RMP-Path) and the tree-based formulation (RMP-Tree) have a reduced set of variables and constraints. The RMP-Path contains a subset of the path variables λ^p_{st} as well as a subset of the $|E||R|$ coupling constraints (6.3b). RMP-Path is initialized with paths that are solutions to the s-t-shortest-path problems for all $(s,t) \in R$. Similarly, the RMP-Tree contains a reduced set of the tree variables λ^q_u as well as a subset of the $|E||V|$ coupling constraints (6.4b). We initialize RMP-Tree with trees that are solutions to the minimum spanning tree problem and to s-to-all shortest-path problems for all $s \in V$. For each tree, we add a column per commodity $u \in V$. The cost of such a column can be easily computed by solving a modified u-to-all shortest-path problem.

We start with the description of the combined column-and-row generation algorithms while branching and cutting is discussed later in Section 6.4.5.

Column Generation for the Path-based Reformulation

Let $\pi_{stij} \leq 0$ be the dual prices of the constraints (6.3b) of RMP-Path, and let $\mu_{st} \in \mathbb{R}$ be the dual prices of the convexity constraints (6.3e). There is a pricing problem for each pair $(s,t) \in R$ asking for a negative reduced-cost s-t-path:

$$
\begin{aligned}
\min \quad & \sum_{(i,j)\in A} (r_{st}c_{ij} - \pi_{stij}) f_{stij} - \mu_{st} \\
\text{s.t.} \quad & \sum_{(s,j)\in\delta^+(s)} f_{stsj} = 1 \\
& \sum_{(i,j)\in\delta^+(i)} f_{stij} - \sum_{(j,i)\in\delta^-(i)} f_{stji} = 0 && \forall i \in V \setminus \{s,t\} \\
& f_{stij} \geq 0 && \forall (i,j) \in A
\end{aligned}
$$

This is the network-flow formulation of the s-t-shortest-path problem on $D = (V, A)$ with arc costs $\tilde{c}_{ij} := (r_{st}c_{ij} - \pi_{stij})$. Due to the non-negativity of the coefficients $\tilde{c}_{ij}$, the pricing problem can be easily solved with Dijkstra's algorithm (Dijkstra, 1959). We can generate more than one negative reduced-cost column per iteration by using the following modification. Every time vertex t is labeled, we check if the corresponding path has negative reduced cost and if so, we add the corresponding variable to RMP-Path.

Column Generation for the Tree-based Reformulation

Let $\pi_{uij} \leq 0$ be the dual prices of the constraints (6.4b) of the RMP-Tree, and let $\mu_u \in \mathbb{R}$ be the dual prices of the convexity constraints (6.4e). There is one pricing problem for each commodity $u \in V$ asking for a negative-reduced cost spanning tree with flows:

$$\min \quad \sum_{(i,j)\in A} c_{ij} f_{uij} - \sum_{(i,j)\in A} \pi_{uij} y_{uij} - \mu_u \tag{6.6a}$$

$$\text{s.t.} \quad \sum_{(u,j)\in\delta^+(u)} f_{uuj} = \sum_{h\in V\setminus\{u\}} r_{uh} \tag{6.6b}$$

$$\sum_{(h,j)\in\delta^+(h)} f_{uhj} - \sum_{(i,h)\in\delta^-(h)} f_{uih} = -r_{uh} \qquad \forall h \in V \setminus \{u\} \tag{6.6c}$$

$$f_{uij} \leq M_u y_{uij} \qquad \forall (i,j) \in A \tag{6.6d}$$

$$\sum_{(i,j)\in A} y_{uij} \leq n - 1 \tag{6.6e}$$

$$y_{uij} \in \{0,1\}, f_{uij} \geq 0 \qquad \forall (i,j) \in A \tag{6.6f}$$

This problem is a *fixed-cost network flow problem* (FCNFP). Its objective (6.6a) is to minimize the sum of fixed costs and flow costs. The decision that an arc (i,j) is used, indicated by $y_{uij} = 1$, results in a fixed cost $-\pi_{uij} \geq 0$, while flow along that arc, given by the value of f_{uij}, imposes flow costs $c_{ij}f_{uij}$. Constraints (6.6b) and (6.6c) guarantee that the supply of commodity u flows from vertex u to every other vertex. Consistency between flow and arc variables is ensured by (6.6d) using the same big number M_u constant as in Section 6.3.2. Constraints (6.6e) limit the number of arcs in the spanning tree and (6.6f) describe the domains of the variables.

The pricing problem (6.6) is guaranteed to generate valid spanning trees only if all vertices $i \in V \setminus \{u\}$ have a positive demand $r_{ui} > 0$. Otherwise, less than $|V| - 1$ arcs may be needed to reach all positive-demand vertices possibly creating undirected cycles. The following inequalities can be added to exclude such non-tree solutions:

$$\sum_{(h,i)\in\delta^-(i)} y_{uhi} \leq 1 \qquad i \in V \setminus \{u\}$$

The FCNFP is $\mathcal{NP}$-hard as it generalizes the Steiner tree problem (Garey and Johnson, 1979). Many heuristic and exact methods have been developed for the FCNFP (Hewitt *et al.*, 2010).

Heuristic solution of the FCNFP We decided to apply a *dynamic slope scaling procedure* (DSSP, Kim and Pardalos, 1999) to heuristically solve the pricing problem. DSSP completely removes the binary variables from the model by approximating costs of an optimal FCNFP solution with new variable costs $\widehat{c}_{uij}$ that simultaneously reflect the original fixed costs $-\pi_{uij}$ and variable costs c_{ij}. The coefficients $\widehat{c}_{uij}$ are approximated by solving successive u-to-many *minimum-cost flow* (MCF) problems and iteratively updating the coefficients depending on the computed values. The u-to-many MCF problems can be solved quickly with CPLEX. Each solution provides a feasible solution of the FCNFP and therefore each iteration may generate a negative reduced-cost tree with flows.

Exact solution of the FCNFP We use the branch-and-cut algorithm of Ortega and Wolsey (2003) for the exact solution of FCNFP. Ortega and Wolsey (2003) introduce *dicut inequalities* and their variants together with several heuristics for their separation. Standard branching on the y_{uij}-variables is applied. Every time the branch-and-cut finds a feasible integer solution, we add the corresponding variable to RMP-Tree if it has negative reduced cost.

Moreover, it can occur that a tree solution contains leaf vertices with demand zero. (A leaf is a vertex with degree one.) We modify such a solution by pruning zero-demand leaf vertices (note that (6.6e) is an inequality). This avoids the generation of equivalent columns with equal cost. Pruning is also used in some heuristics for the Steiner tree problem (Voß, 1992).

Overall pricing strategy In each iteration of the RMP-Tree, one pricing problem per commodity results. As they are $\mathcal{NP}$-hard, solving all of them requires a large amount of computation time. However, it suffices to terminate pricing when one negative reduced-cost solution is found. This method is known as partial column generation and was previously applied, e.g., to large-scale aircrew rostering problems (Gamache *et al.*, 1999). Our implementation of partial column generation computes priorities for the commodities $u \in V$ to determine the order in which the pricing problems in each iteration are examined. The priorities are updated depending on the number of columns generated in the previous iterations.

Moreover, a tree computed for a specific commodity u may at the same time be a solution to the pricing problem of every other commodity. We exploit this fact using the following heuristic: First, non-spanning trees are extended to spanning trees using the original edge costs c. This can be done in several ways (we check all): Use a minimum spanning-tree algorithm starting with the given tree. Alternatively, for each $j \in V \setminus \{u\}$, shrink the tree into a single super-vertex and compute a shortest-path tree rooted at vertex j. Second, for each commodity $i \in V \setminus \{u\}$ prune zero-demand leaf vertices as

explained before. As a by-product of this procedure, we obtain an upper bound for the OCSTP by simply summing up the communication costs for all commodities.

Row Generation

RMP-Path and RMP-Tree are typically highly degenerate linear programs. The reason is that the path-based formulation (6.3) consists of $\mathcal{O}(|E||R|)$ constraints while an integer basic solution comprises no more than $|R| + (|V| - 1)$ positive variables. The tree-based formulation has $\mathcal{O}(|E||V|)$ constraints, but an integer basic solution consists of no more than $|V| + (|V| - 1)$ positive variables. To overcome degeneracy problems, we generate the coupling constraints (6.3b) and (6.4b) of RMP-Path and RMP-Tree dynamically. Fischetti *et al.* (2002) use a similar row-generation method to solve the minimum routing-cost-tree problem with a path-based formulation.

In both reformulations, we initialize the RMP with no coupling constraints. For each edge $\{i, j\} \in E$, violated coupling constraints (6.3b) and (6.4b) can be identified by inspection. We add violated coupling constraints only if there are no more negative reduced-cost columns in the current RMP. Hence, in our implementation of combined column-and-row generation, a valid lower bound for the OCSTP results whenever no negative reduced-cost columns are found. In Section 6.5, we quantify the positive impact of a dynamic row generation on the basis of a computational study.

6.4.5 Branching and Cutting

Branching on the edge variables $x_{ij} \in \{0, 1\}$ ensures integrality in both reformulations (6.3) and (6.4). We use the following variable-selection strategy inspired by Ahuja and Murty (1987). It can be observed that most of the time the lower bound of the zero-branch is weaker than the lower bound of the one-branch. Therefore, one should try to maximize the lower bound of the zero-branch. More precisely, let z_{st} be the length of a shortest s-t-path. A trivial lower bound on the cost of the OCSTP can be computed by summing up z_{st} multiplied with the communication requirement r_{st} over all pairs $(s, t) \in R$, i.e., $\sum_{(s,t)\in R} z_{st} r_{st}$. Now, a lower bound of a specific node of the branch-and-bound tree is defined as $\sum_{(s,t)\in R} z_{st} r_{st}$ such that the shortest-path corresponding to z_{st} respects all previous branching decisions. Hence, among all edges $\{i, j\}$ with fractional value $\bar{x}_{ij}$, we choose the one that maximizes $\sum_{(s,t)\in R} z_{st} r_{st}$ such that z_{st} is the length of a shortest s-t-path that respects all previous branching decision and the tentative branching decision $x_{ij} = 0$. Pretests have revealed that such a rule is often able to well balance the branch-and-bound tree and on average outperforms standard strategies like the fractional value closest to a fixed value between 0 and 1. We break ties by preferring edges $\{i, j\}$ with higher cost c_{ij}.

Solving the path-based formulation (6.3) requires massive branching. We therefore decided to use strong branching: Every time a branching decision must be made, we choose up to five candidate edges with fractional values $\bar{x}_{ij}$ that maximize the lower bound defined above. The two son nodes of each candidate edge $\{i, j\}$ are solved and their lower bounds $LB^1_{\{i,j\}}$ and $LB^2_{\{i,j\}}$ are stored. Among the candidate edges, we

choose the one that maximizes $\min\{LB^1_{\{i,j\}}, LB^2_{\{i,j\}}\}$. In order to compute valid upper bounds in the path-based formulation, we compute for all nodes s at every node of the search-tree a s-to-all shortest path tree and a minimum spanning tree that respect all previous branching decisions. These trees are feasible solutions to the OCSTP and their communication costs can be computed easily.

In order to strengthen the master program bounds for the path-based formulation (6.3), we use the following *cycle-elimination inequalities* (Padberg and Wolsey, 1983):

$$\sum_{\{i,j\}\in E(S)} x_{ij} \leq |S| - 1 \qquad \forall \varnothing \neq S \subsetneq V$$

Herein, $E(S)$ is the set of all edges with both endpoints in S. Violated cycle-elimination inequalities can be separated solving maximum-flow problems (see Padberg and Wolsey, 1983). These inequalities are valid but redundant for the tree-based formulation (6.4).

6.5 Computational Results

This section presents the computational results for both branch-and-price-and-cut algorithms for the OCSTP. All computations are performed on a standard PC with an Intel(R) Core(TM) i7-5930k 3.5 GHz processor and 64 GB of main memory. Algorithms are coded in C++ and compiled into 64-bit single-thread code with MS Visual Studio 2013. The callable library of CPLEX 12.6 is used to iteratively re-optimize both RMPs in the combined column-and-row-generation algorithm as well as to solve the heuristic and exact MCF and FCNFP pricing problems of the tree-based formulation. We test our algorithms on the set of benchmark instances from Contreras *et al.* (2010). There are 20 instances divided into five groups each containing four instances with 10, 20, 30, 40, and 50 vertices, respectively. Distances and demand are uniformly distributed in [0,100] and approximately 50 % of the vertex pairs have a positive communication requirement.

For all runs, we set a hard time limit of 7 200 seconds. Pre-tests have shown that the following strategies are beneficial:

(1) Define the communication requirements in the most asymmetric way in both formulations.

For the path-based formulation:

(2) Use the dual simplex algorithm for reoptimizing RMP-Path.

(3) Do not apply partial column generation. Shortest-path computations using the Dijkstra algorithm are already very fast.

(4) Use cycle-elimination inequalities in the entire search tree.

(5) Remove columns as soon as the ratio columns to rows exceeds 1.1 to avoid too long computations times for solving the master program. Columns with the largest positive reduced-cost are removed.

(6) Use strong branching with five candidate edges in the entire search tree.

For the tree-based formulation:

(7) Use the barrier optimizer of CPLEX to reoptimize RMP-Tree. The dual prices are stabilized and oscillate less compared to using the primal or dual simplex algorithm.

(8) Use partial column generation and terminate pricing as soon as a negative reduced-cost column is found. Solving the pricing problem exactly is by far the most time-consuming component.

(9) Reuse negative reduced-cost trees of one commodity for all other commodities and use them to compute upper bounds.

6.5.1 Impact of Dynamic Row Generation

Tables 6.1 and 6.2 show the impact of dynamic row generation for both formulations (6.3) and (6.4). The table entries have the following meaning:

$\#rows$	the average number of rows present at the last RMP solved
T_{RMP}	the average time for solving one iteration of RMP-Path/RMP-Tree in seconds
gap	the average gap in percent between the best lower and upper bound found
T	the average time in seconds needed to solve the instance to optimality
$\#solved$	the number of instances solved to optimality

	all rows					dynamic row generation				
Group	$\#rows$	T_{RMP}	gap	T	$\#solved$	$\#rows$	T_{RMP}	gap	T	$\#solved$
`Contreras10`	1 094	0.01	0.00	0.08	4/4	187	0.01	0.00	0.05	4/4
`Contreras20`	16 429	0.02	0.00	4.55	4/4	1 573	0.01	0.00	2.08	4/4
`Contreras30`	106 282	0.54	0.00	370.69	4/4	6 440	0.02	0.00	27.44	4/4
`Contreras40`	308 508	3.49	2.85	5 714.76	1/4	16 500	0.25	1.02	3 273.75	3/4
`Contreras50`	773 616	33.92	7.78	7 200.00	0/4	31 115	0.88	3.79	6 977.81	1/4
All	241 186	7.59	2.13	2 658.02	13/20	11 163	0.23	0.96	2 056.23	16/20

Table 6.1: Impact of applying dynamic row generation in the path-based reformulation (6.3)

Table 6.1 quantifies the benefit of using dynamic row generation in the path-based reformulation. On average only 4.6 % of all coupling constraints are present in the RMP resulting in a reduction of the computation time from 7.59 to 0.23 seconds on average for one iteration of the RMP. The average gap at the end of the optimization is reduced by more than 1.1 %.

	all rows					dynamic row generation				
Group	$\#rows$	T_{RMP}	gap	T	$\#solved$	$\#rows$	T_{RMP}	gap	T	$\#solved$
Contreras10	312	0.01	0.00	8.79	4/4	120	0.01	0.00	24.93	4/4
Contreras20	2 962	0.12	0.00	927.59	4/4	635	0.04	0.00	637.94	4/4
Contreras30	11 446	2.55	8.32	7 200.00	0/4	1 380	0.28	2.45	7 200.00	0/4
Contreras40	26 555	6.59	13.25	7 200.00	0/4	2 160	0.78	6.43	7 200.00	0/4
Contreras50	53 945	22.93	38.42	7 200.00	0/4	3 025	2.19	11.58	7 200.00	0/4
All	19 044	6.44	12.00	4 507.28	8/20	1464	0.66	4.09	4 452.58	8/20

Table 6.2: Impact of applying dynamic row generation in the tree-based reformulation (6.4)

In the tree-based reformulation, the impact of row generation is smaller as can be seen from Table 6.2. On average there are only 7.7 % of all coupling constraints present when applying dynamic row generation and the average computation time for solving RMP-Tree decreases by almost factor 10 (from 6.44 to 0.66 seconds). Therefore, the overall solution time also decreases, but since most instances cannot be solved to proven optimality, the reduction in average computation time is small. The average gap decreases by around 8 %. The impact of row generation increases significantly for larger instances in both reformulations.

Table 6.3 compares the solutions of the two reformulations (6.3) and (6.4) when dynamic row generation is applied. The table entries have the following meaning:

gap_{root} the average gap at the root node in percent

T_{root} the time to solve the root node in seconds

gap_{closed} the percentage of the root gap closed at the end of the branch-and-cut-and-price

$\#N$ the number of branch-and-bound nodes solved

$\#PP$ the number of pricing problems solved

	path-based formulation (6.3)					tree-based formulation (6.4)				
Group	gap_{root}	T_{root}	gap_{closed}	$\#N$	$\#PP$	gap_{root}	T_{root}	gap_{closed}	$\#N$	$\#PP$
Contreras10	0.27	0.02	100.00	2	962	0.07	4.90	100.00	2	774
Contreras20	2.38	0.16	100.00	9	23 150	0.68	241.28	100.00	10	1 941
Contreras30	2.41	1.72	100.00	14	133 391	2.45*	7 200.00	0.00	0	3 233
Contreras40	5.06	4.99	90.66	321	2 551 881	6.43*	7 200.00	0.00	0	2 861
Contreras50	7.58	22.11	59.60	168	3 463 866	11.58*	7 200.00	0.00	0	2 890
All	3.54	5.80	90.05	103	1 234 650	4.24	4369.23	40.00	2	2 340

Table 6.3: Comparison of both reformulations solved with dynamic row generation

The tree-based formulation is not able to solve the root node of a single instance when the number of vertices exceeds 20. The values marked with * in the table show the gap between the best Lagrangian lower bound and the best upper bound found at

the root node. As clear from Proposition 2, the root lower bound of the tree-based reformulation dominates the root lower bound of the path-based formulation, but the solution times of the tree-based formulation are on average more than 750 times larger. Therefore, the path-based formulation is able to close more than 90 % of the root gap by branching and cutting, while the tree-based formulation is not even able to solve the root node for instances with 30 or more vertices. The last two columns indicate that many more shortest-path pricing problems can be solved within the time limit compared to FCNFP pricing problems. In turn, more branch-and-bound nodes can be solved in the path-based reformulation leading to smaller optimality gaps at the end.

6.5.2 Comparison of all four Formulations

In this section, we compare the two compact and extensive formulations. In addition to the instances from (Contreras *et al.*, 2010), we use other benchmark instances for the OCSTP from the literature that we have obtained from Franz Rothlauf (Johannes Gutenberg University Mainz, Germany). These instances can be grouped into three classes: First, five instances, attributed to `Raidl` with 10, 20, 50, 75, and 100 vertices, respectively. Distances and demands are uniformly distributed in [0, 100]. Second, three instances from Palmer (1994) with 6, 12, and 24 vertices, respectively. The vertices correspond to cities in the United States of America, and the distances between cities are obtained from a database. The demands between nodes are inversely proportional to their distances. Third, three instances from Berry *et al.* (1995), one with six vertices and two with 35 vertices. The instance labeled `35u` has the same communication requirements as instance `Berry35` but all edge costs are equal to one. The two compact formulations are solved with the callable library of CPLEX 12.6. CPLEX was allowed to use its standard cuts (default). The two extensive formulations were solved with branch-and-price-(and-cut) and dynamic row generation as in the subsection before.

Table 6.4 compares all four formulations, compact and extensive as well as path-based and tree-based, respectively. The value *Opt* denotes the cost of an optimal solution found if any of the algorithms was able to terminate in time. Values marked with an * are those instances that are solved to optimality for the first time (as far as we know). *gap* denotes the gap between the computed lower bound and the best known solution, and T is the solution time in seconds. For this comparison, we compute the gap relative to the best known (heuristic) solution because CPLEX was not able to find competitive upper bounds when solving the compact formulations for instances with 50 or more vertices. The best known solution values for all instances that were not solved to optimality are taken from the work of Contreras *et al.* (2010).

The compact path-based formulation is not able to solve instances with more than 50 vertices due to memory limitations. The results show that the extensive path-based formulation (6.3) produces consistently better results than the other formulations: Solution times are on average more than 50 % smaller and three new optimal solutions could be found although their values coincide with the upper bounds computed by (Contreras *et al.*, 2010).

		compact path-based		compact tree-based		extensive path-based		extensive tree-based	
Instance	Opt	*gap*	*T*	*gap*	*T*	*gap*	*T*	*gap*	*T*
Contreras10a	71 156	0.00	0.12	0.00	0.72	0.00	0.06	0.00	4.07
Contreras10b	38 059	0.00	0.07	0.00	0.27	0.00	0.03	0.00	1.15
Contreras10c	29 113	0.00	0.08	0.00	0.17	0.00	0.03	0.00	0.16
Contreras10d	39 197	0.00	0.07	0.00	0.42	0.00	0.08	0.00	1.53
Contreras20a	89 474	0.00	6.65	0.00	291.64	0.00	3.30	0.00	1 334.21
Contreras20b	96 333	0.00	1.80	0.00	66.16	0.00	2.44	0.00	335.85
Contreras20c	102 505	0.00	4.13	0.00	257.90	0.00	2.30	0.00	285.68
Contreras20d	87 452	0.00	1.20	0.00	20.97	0.00	0.29	0.00	58.78
Contreras30a	228 247	0.00	75.41	2.40	7 200.00	0.00	24.75	0.50	7 200.00
Contreras30b	249 607	0.00	113.94	2.52	7 200.00	0.00	30.57	1.39	7 200.00
Contreras30c	209 062	0.00	245.92	2.08	7 200.00	0.00	22.94	1.35	7 200.00
Contreras30d	219 170	0.00	180.38	2.51	7 200.00	0.00	31.52	1.06	7 200.00
Contreras40a	350 542	0.00	316.80	2.71	7 200.00	0.00	43.43	0.88	7 200.00
Contreras40b	292 047*	1.46	7 200.00	7.11	7 200.00	0.00	2 511.97	4.23	7 200.00
Contreras40c	287 198*	1.10	7 200.00	6.96	7 200.00	0.00	3 339.61	4.09	7 200.00
Contreras40d		8.21	7 200.00	11.39	7 200.00	3.54	7 200.00	10.37	7 200.00
Contreras50a	458 881*	4.07	7 200.00	9.46	7 200.00	0.00	6 311.23	6.75	7 200.00
Contreras50b		7.41	7 200.00	12.34	7 200.00	3.16	7 200.00	10.66	7 200.00
Contreras50c		7.89	7 200.00	14.29	7 200.00	4.67	7 200.00	11.79	7 200.00
Contreras50a		7.18	7 200.00	12.25	7 200.00	4.81	7 200.00	10.23	7 200.00
Raidl10	53 674	0.00	0.09	0.00	0.18	0.00	0.03	0.00	1.05
Raidl20	157 570	0.00	5.94	0.00	99.44	0.00	3.09	0.00	125.72
Raidl50		6.95	7 200.00	17.59	7 200.00	5.02	7 200.00	23.29	7 200.00
Raidl75		-	-	27.48	7 200.00	12.41	7 200.00	28.87	7 200.00
Raidl100		-	-	28.17	7 200.00	12.07	7 200.00	74.70	7 200.00
Palmer6	693 180	0.00	0.06	0.00	0.10	0.00	0.03	0.00	0.27
Palmer12	3 428 510	0.00	2.81	0.00	20.86	0.00	7.25	0.00	1471.34
Palmer24	1 086 660	0.00	0.17	0.00	4.71	0.00	0.02	0.00	0.44
berry6	534	0.00	0.08	0.00	0.08	0.00	0.03	0.00	0.14
berry35	16 915	0.00	0.64	0.00	123.51	0.00	0.13	0.00	16.15
berry35u		12.92	7 200.00	13.34	7 200.00	10.02	7 200.00	12.78	7 200.00
Solved		20/31		15/31		23/31		15/31	

Table 6.4: Comparison of compact and extensive formulations

6.6 Conclusions

This paper has presented a path-based and a tree-based extensive formulation of the OCSTP. Due to their large number of variables and constraints, both Dantzig-Wolfe reformulations are solved with combined column-and-row generation. Branch-and-bound finally ensures integrality of solutions.

We have shown that the root lower bound of the compact path-based formulation dominates the root lower bound of the compact tree-based formulation, while it is the other way around for the two extensive formulations. Our findings concerning a reasonable design of combined column-and-row-generation algorithms are the following: Dynamic row generation significantly accelerates the solution process. Partial pricing is advantageous for the tree-based but not for the path-based reformulation. For the former pricing problem, partial pricing includes the use of FCNFP heuristics as well as prioritizing the pricing problems (there is one for each vertex/commodity) and stopping pricing whenever a negative reduced cost tree with flows is found. Other important components of the algorithm for the tree-based formulation are the computation of upper bounds from trees computed in the pricing problems and the exchange of trees between pricing problems of different commodities.

In summary, the combined column-and-row-generation algorithm for the path-based reformulation is superior on the benchmark instances although its linear relaxation bound is dominated by the one of the tree-based formulation. The decisive point is that the speed, in which simple shortest-path pricing problems can be solved in the path-based approach, overcompensates the weaker linear relaxation. The tree-based combined column-and-row-generation approach suffers too much from the time-consuming FCNFP pricing problems, which finally have to be solved by branch-and-cut. Overall, the extensive path-based formulation is able to solve instances with up to 50 vertices to optimality, while increasing the number of vertices leads to optimality gaps that can easily reach 5 % and more. Moreover, the extensive path-based formulation has provided three new optimal solutions.

References

Ahuja, R. and Murty, V. (1987). Exact and heuristic algorithms for the optimum communication spanning tree problem. *Transportation Science*, **21**(3), 163–170.

Berry, L., Murtagh, B., and McMahon, G. (1995). Applications of a genetic-based algorithm for optimal design of tree-structured communication networks. In *Proceedings of the Regional Teletraffic Engineering Conference of the International Teletraffic Congress*, pages 361–370.

Contreras, I. and Fernández, E. (2012). General network design: A unified view of combined location and network design problems. *European Journal of Operational Research*, **219**(3), 680–697.

Contreras, I., Fernández, E., and Marín, A. (2010). Lagrangean bounds for the optimum communication spanning tree problem. *Top*, **18**(1), 140–157.

Desaulniers, G., Desrosiers, J., and Solomon, M., editors (2005). *Column Generation*. Springer, New York, NY.

Dijkstra, E. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik*, **1**, 269–271.

Fernández, E., Luna-Mota, C., Hildenbrandt, A., Reinelt, G., and Wiesberg, S. (2013). A flow formulation for the optimum communication spanning tree. *Electronic Notes in Discrete Mathematics*, **41**(0), 85 – 92.

Fischer, T. and Merz, P. (2007). A memetic algorithm for the optimum communication spanning tree problem. In *Hybrid Metaheuristics*, pages 170–184. Springer.

Fischetti, M., Lancia, G., and Serafini, P. (2002). Exact algorithms for minimum routing cost trees. *Networks*, **39**(3), 161–173.

Gamache, M., Soumis, F., and Marquis, G. (1999). A column generation approach for large-scale aircrew rostering problems. *Operations Research*, **47**(2), 247–263.

Garey, M. and Johnson, D. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman, San Francisco.

Hewitt, M., Nemhauser, G. L., and Savelsbergh, M. W. (2010). Combining exact and heuristic approaches for the capacitated fixed-charge network flow problem. *INFORMS Journal on Computing*, **22**(2), 314–325.

Hu, T. C. (1974). Optimum communication spanning trees. *SIAM Journal on Computing*, **3**(3), 188–195.

Johnson, D. S., Lenstra, J. K., and Rinnooy Kan, A. (1978). The complexity of the network design problem. *Networks*, **8**(4), 279–285.

Kim, D. and Pardalos, P. M. (1999). A solution approach to the fixed charge network flow problem using a dynamic slope scaling procedure. *Operations Research Letters*, **24**(4), 195–203.

Lübbecke, M. and Desrosiers, J. (2005). Selected topics in column generation. *Operations Research*, **53**(6), 1007–1023.

Ortega, F. and Wolsey, L. A. (2003). A branch-and-cut algorithm for the single-commodity, uncapacitated, fixed-charge network flow problem. *Networks*, **41**(3), 143–158.

Padberg, M. W. and Wolsey, L. A. (1983). Trees and cuts. In C. Berge, D. Bresson, P. Camion, J. Maurras, and F. Sterboul, editors, *Combinatorial MathematicsProceedings of the International Colloquium on Graph Theory and Combinatorics*, volume 75 of *North-Holland Mathematics Studies*, pages 511 – 517. North-Holland.

Palmer, C. C. (1994). *An approach to a problem in network design using genetic algorithms*. Ph.D. thesis, Citeseer.

Peleg, D. and Reshef, E. (1998). Deterministic polylog approximation for minimum communication spanning trees. In K. Larsen, S. Skyum, and G. Winskel, editors, *Automata, Languages and Programming*, volume 1443 of *Lecture Notes in Computer Science*, pages 670–681. Springer Berlin Heidelberg.

Rothlauf, F. (2009). On optimal solutions for the optimal communication spanning tree problem. *Operations Research*, **57**(2), 413–425.

Sharma, P. (2006). Algorithms for the optimum communication spanning tree problem. *Annals of Operations Research*, **143**(1), 203–209.

Steitz, W. and Rothlauf, F. (2012). Using penalties instead of rewards: Solving OCST problems with guided local search. *Swarm and Evolutionary Computation*, **3**, 46–53.

Voß, S. (1992). Steiner's problem in graphs: heuristic methods. *Discrete Applied Mathematics*, **40**(1), 45–72.

Wu, B. Y., Chao, K.-M., and Tang, C. Y. (2000). Approximation algorithms for some optimum communication spanning tree problems. *Discrete Applied Mathematics*, **102**(3), 245–266.

Chapter 7

Conclusion

The main goal of this thesis was to contribute to the development of new column-generation approaches in the field of combinatorial optimization. Here, we give a chapter-wise summary, restate the main results, and give concluding remarks.

Chapter 2 has addressed the minimum tour duration problem (MTDP), which is a variant of the traveling salesman problem with time windows where the objective is the minimization of the time between the departure at the start and the arrival at the destination. Many different algorithmic components for a DP-based approach tailored to the MTDP have been presented, such as relaxed and exact forward and backward DPs, a VND, a dynamic *ng* neighborhood augmentation procedure, and a penalty method based on subgradient and column-generation algorithms. We see one main contribution in the consistent way that resources and REFs are defined for both forward and backward DP including relaxations. This consistency (see Section 2.3) is the theoretic background for the later algorithm design.

We have presented a detailed computational study to identify and parameterize those components that can be combined efficiently in an overall computational setup. Our findings are the following: First, for bounding purposes, a combined $ngL.2res$ relaxation provides an excellent tradeoff between the strength of the resulting bounds and the computational effort. No other relaxation was found competitive with $ngL.2res$. Second, an algorithm penalizing those routes that are non-elementary is often more effective than enlarging the *ng*-neighborhoods. Third, with optimized penalties and afterwards carefully dynamically augmented *ng*-neighborhoods, excellent lower bounds on the MTDP can be achieved. Our computational analysis shows that these lower bounds often suffice to either close the gap or to make the exact DP solution with a bounding procedure relatively easy. Fourth, upper bounds produced with the Balas-Simonetti neighborhood-based VND are generally tight. However, using very tight tentative upper bounds for the exact DP and increasing them step-wise seems computationally more advantageous. Summarizing the presented computational results, the proposed DP-based approach can solve more than 90 % of the instances with up to 125 nodes from the standard benchmark sets to proven optimality.

Chapter 3 has investigated the exact solution of the active-passive vehicle-routing problem (APVRP) by means of a branch-and-price-and-cut method. The APVRP supports a flexible coupling of transport resources in order to achieve an efficient resource utilization and high-quality transport solutions. Therefore, synchronization constraints for the operations and the movement of active and passive vehicles are needed and have to be incorporated in the branch-price-and-cut-algorithm. Hence, we have introduced

an extended network that introduces linear vertex costs in the SPPRC pricing problem, thus significantly complicating its solution. A bidirectional labeling algorithm and a refinement of the *ng*-path relaxation were applied to solve the pricing problem. Therein, we introduce a sophisticated procedure for merging forward and backward labels. Computational experiments show that our method delivers improved results for the APVRP benchmark instances introduced by Meisel and Kopfer (2014). For a newly created benchmark set, it is capable of solving instances with 76 tasks, 4 active, and 8 passive vehicles to optimality.

In its current form, the APVRP already covers a number of relevant applications from the areas of distribution logistics, health care management, the security industry and others. Nevertheless, there are several further features of real-world problems that are not yet supported by the presented model. This includes, for example, the possibility of a temporary drop-off of passive vehicles at intermediate locations (e.g., public parking lots) that neither belong to the set of initial or final locations nor to the set of pickup and delivery locations. Such drop-off locations could reduce the detours needed by active vehicles for the exchange of passive vehicles. Moreover, in many practical applications, vehicles with a capacity of more than one are used, and performing load transshipments between such vehicles is quite common in practice. These features and their corresponding synchronization requirements represent practically relevant and mathematically non-trivial extensions of the APVRP and will be a subject of our future research.

In Chapter 4, we have investigated a branch-and-price-and-cut algorithm for the vehicle routing and truck driver scheduling problem (VRTDSP) with U.S. hours of service regulations. We presented consistent backward resource extension functions in order to build a bidirectional labeling with a sophisticated merge procedure. Different techniques, e.g., the *ng*-path relaxation and limited discrepancy search, were used to speed up the solution process of the pricing problem. In addition, valid inequalities for the vehicle routing problem with time windows were adapted to strengthen the linear relaxation.

We have presented a detailed computational study to analyze the different algorithmic components. Our findings are the following: First, using a dynamic half-way point reduces the number of extended labels significantly resulting in a huge gain in terms of computation time. Second, choosing a moderate *ng*-neighborhood size of ten provides an excellent tradeoff between the strength of the resulting bounds and the computational effort. Third, a careful choice of the parameters is necessary when applying valid inequalities. Furthermore, when combining different classes of valid inequalities, the choice of the classes and the order of applying them are crucial. The best setting found was applying 2-path, subset-row inequalities, and the dynamic neighborhood extension in that order. The computational experiments with this setting show that our method delivers improved results for the VRTDSP benchmark instances introduced by Goel (2009). Our algorithm is able to solve all 25 customer instances in less than half an hour to optimality. In addition, nearly 80 % of the 50 customer instances were solved to optimality in two hours of computation time.

In Chapter 5, we have addressed the solution of different variants of the SPPRC with the help of bidirectional dynamic programming labeling algorithms. This type of

algorithm is based on the definition of a so-called half-way point separating the forward from the backward labeling. In previous bidirectional labeling algorithms, it had to be specified a priori before the labeling procedure starts. We exploit asymmetry in input data (constraints and reduced costs) as well as asymmetry in the forward and backward labeling process (different number of resources, bounding procedures and dominance rules of different strength) by defining forward and backward half-way points that are dynamically adjusted according to an expected workload to be spent in forward and backward labeling.

We have conducted computational experiments with three types of VRPs, where we compared the solutions of the column-generation SPPRC pricing problems that were solved with bidirectional labeling algorithms either with a static or the new dynamic half-way point. When using a dynamic half-way point, the solution times could be reduced by 15 to 40 % on average depending on the SPPRC variant. The general insight of the experiments is the following: The harder the SPPRC instance, the more effective is the dynamic half-way point version. The necessary modifications needed to alter a bidirectional SPPRC labeling algorithm with static into one with dynamic half-way point is very minor. Thus, a significant effect can be achieved with little implementation effort.

Chapter 6 has presented a path-based and a tree-based extensive formulation of the optimum communication spanning tree problem. Both Dantzig-Wolfe reformulations are solved with combined column-and-row generation due to their large number of variables and constraints. Branch-and-bound finally ensures integrality of solutions.

Our computational experiments have shown that dynamic row generation significantly accelerates the solution process. Moreover, due to the fact that the pricing problem of the tree-based formulation is $\mathcal{NP}$-hard, partial pricing is advantageous. In the path-based reformulation, this is not the case because simple shortest-path pricing problems can be solved very fast. Other important components of the algorithm for the tree-based formulation are the computation of upper bounds from trees computed in the pricing problems and the exchange of trees between pricing problems of different commodities. In summary, the combined column-and-row-generation algorithm for the path-based reformulation is superior although its linear relaxation bound is dominated by the one of the tree-based formulation. The tree-based combined column-and-row-generation approach suffers too much from the time-consuming FCNFP pricing problems, which are finally solved by branch-and-cut. Overall, the path-based formulation is able to solve instances with up to 40 vertices to optimality, while increasing the number of vertices leads to optimality gaps that can easily reach 5 % and more.

References

Afifi, S., Dang, D.-C., and Moukrim, A. (2015). Heuristic solutions for the vehicle routing problem with time windows and synchronized visits. *Optimization Letters, in press.*

Ahuja, R. and Murty, V. (1987). Exact and heuristic algorithms for the optimum communication spanning tree problem. *Transportation Science*, **21**(3), 163–170.

Andersson, H., Duesund, J. M., and Fagerholt, K. (2011). Ship routing and scheduling with cargo coupling and synchronization constraints. *Computers & Industrial Engineering*, **61**(4), 1107–1116.

Ascheuer, N. (1995). *Hamiltonian Path Problems in the On-Line Optimization of Flexible Manufacturing Systems.* Ph.D. thesis, Technische Universität Berlin.

Ascheuer, N., Fischetti, M., and Grötschel, M. (2001). Solving the asymmetric traveling salesmann problem with time windows by branch-and-cut. *Mathematical Programming, Series A*, **90**(3), 475–506.

Baker, E. (1983). An exact algorithm for the time-constrained traveling salesman problem. *Operations Research*, **31**, 938–945.

Balas, E. (1999). New classes of efficiently solvable generalized traveling salesman problems. *Annals of Operations Research*, **86**, 529–558.

Balas, E. and Simonetti, N. (2001). Linear time dynamic-programming algorithms for new classes of restricted TSPs: A computational study. *INFORMS Journal on Computing*, **13**(1), 56–75.

Baldacci, R., Mingozzi, A., and Roberti, R. (2011). New route relaxation and pricing strategies for the vehicle routing problem. *Operations Research*, **59**(5), 1269–1283.

Baldacci, R., Mingozzi, A., and Roberti, R. (2012a). New state-space relaxations for solving the traveling salesman problem with time windows. *INFORMS Journal on Computing*, **24**(3), 356–371.

Baldacci, R., Mingozzi, A., and Roberti, R. (2012b). Recent exact algorithms for solving the vehicle routing problem under capacity and time window constraints. *European Journal of Operational Research*, **218**, 1–6.

Barnhart, C., Johnson, E., Nemhauser, G., Savelsbergh, M., and Vance, P. (1998). Branch-and-price: Column generation for solving huge integer programs. *Operations Research*, **46**(3), 316–329.

Bartodziej, P., Derigs, U., Malcherek, D., and Vogel, U. (2009). Models and algorithms for solving combined vehicle and crew scheduling problems with rest constraints: An application to road feeder service planning in air cargo transportation. *OR Spectrum*, **31**, 405–429.

Bentley, J. (1980). Multidimensional divide-and-conquer. *Communications of the ACM*, **23**(4), 214–229.

Berry, L., Murtagh, B., and McMahon, G. (1995). Applications of a genetic-based algorithm for optimal design of tree-structured communication networks. In *Proceedings of the Regional Teletraffic Engineering Conference of the International Teletraffic Congress*, pages 361–370.

Bode, C. and Irnich, S. (2014). The shortest-path problem with resource constraints with (k,2)-loop elimination and its application to the capacitated arc-routing problem. *European Journal of Operational Research*, **238**(2), 415–426.

Bode, C. and Irnich, S. (2015). In-depth analysis of pricing problem relaxations for the capacitated arc-routing problem. *Transportation Science*, **49**(2), 369–383.

Boland, N., Dethridge, J., and Dumitrescu, I. (2006). Accelerated label setting algorithms for the elementary resource constrained shortest path problem. *Operations Research Letters*, **34**(1), 58–68.

Bredström, D. and Rönnqvist, M. (2008). Combined vehicle routing and scheduling with temporal precedence and synchronization constraints. *European Journal of Operational Research*, **191**(1), 19–31.

Buijs, P., Vis, I. F., and Carlo, H. J. (2014). Synchronization in cross-docking networks: A research classification and framework. *European Journal of Operational Research*, **239**(3), 593–608.

Ceselli, A., Righini, G., and Salani, M. (2009). A column generation algorithm for a rich vehicle-routing problem. *Transportation Science*, **43**(1), 56–69.

Cheung, R. K., Shi, N., Powell, W.-B., and Simao, H. P. (2008). An attribute-decision model for cross-border drayage problem. *Transportation Research Part E: Logistics and Transportation Review*, **44**(2), 217–234.

Christiansen, M. and Nygreen, B. (1998). A method for solving ship routing problems with inventory constraints. *Annals of Operations Research*, **81**(0), 357–378.

Christofides, N., Mingozzi, A., and Toth, P. (1981). State-space relaxation procedures for the computation of bounds to routing problems. *Networks*, **11**, 145–164.

Contardo, C., Cordeau, J.-F., and Gendron, B. (2014). An exact algorithm based on cut-and-column generation for the capacitated location-routing problem. *INFORMS Journal on Computing*, **26**(1), 88–102. (Forthcoming.).

Contardo, C., Desaulniers, G., and Lessard, F. (2015). Reaching the elementary lower bound in the vehicle routing problem with time windows. *Networks*, **65**(1), 88–99.

Contreras, I. and Fernández, E. (2012). General network design: A unified view of combined location and network design problems. *European Journal of Operational Research*, **219**(3), 680–697.

Contreras, I., Fernández, E., and Marín, A. (2010). Lagrangean bounds for the optimum communication spanning tree problem. *Top*, **18**(1), 140–157.

Cordeau, J.-F., Desaulniers, G., Desrosiers, J., Solomon, M., and Soumis, F. (2002). VRP with time windows. In P. Toth and D. Vigo, editors, *The Vehicle Routing Problem*, chapter 7, pages 155–194. Society for Industrial and Applied Mathematics, Philadelphia, PA.

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

Dash, S., Günlük, O., Lodi, A., and Tramontani, A. (2012). A time bucket formulation for the traveling salesman problem with time windows. *INFORMS Journal on Computing*, **24**(1), 132–147.

Derigs, U., Kurowsky, R., and Vogel, U. (2011). Solving a real-world vehicle routing problem with multiple use of tractors and trailers and EU-regulations for drivers arising in air cargo road feeder services. *European Journal of Operational Research*, **213**, 309–319.

Desaulniers, G., Desrosiers, J., Ioachim, I., Solomon, M., Soumis, F., and Villeneuve, D. (1998). A unified framework for deterministic time constrained vehicle routing and crew scheduling problems. In T. Crainic and G. Laporte, editors, *Fleet Management and Logistics*, chapter 3, pages 57–93. Kluwer Academic Publisher, Boston, Dordrecht, London.

Desaulniers, G., Desrosiers, J., and Solomon, M., editors (2005). *Column Generation*. Springer, New York, NY.

Desaulniers, G., Lessard, F., and Hadjar, A. (2008). Tabu search, partial elementarity, and generalized k-path inequalities for the vehicle routing problem with time windows. *Transportation Science*, **42**(3), 387–404.

Desaulniers, G., Desrosiers, J., and Spoorendonk, S. (2010). The vehicle routing problem with time windows: State-of-the-art exact solution methods. In J. J. Cochran, L. A. Cox, P. Keskinocak, J. P. Kharoufeh, and J. C. Smith, editors, *Wiley Encyclopedia of Operations Research and Management Science*, volume 8, pages 5742–5749. John Wiley & Sons, Inc.

Desaulniers, G., Madsen, O. B., and Ropke, S. (2014). The vehicle routing problem with time windows. In Toth and Vigo (2014), chapter 5, pages 119–159.

Desaulniers, G., Errico, F., Irnich, S., and Schneider, M. (2016). Exact algorithms for electric vehicle-routing problems with time windows. *Operations Research*. Forthcoming.

Desrochers, M. (1986). *La Fabrication d'Horaires de Travail pour les Conducteurs d'Autobus par une Méthode de Génération de Colonnes*. Ph.D. thesis, Centre de recherche sur les Transports, Université de Montréal, Montréal, Canada. Publication #470, in French.

Desrochers, M., Desrosiers, J., and Solomon, M. (1992). A new optimization algorithm for the vehicle routing problem with time windows. *Operations Research*, **40**(2), 342–354.

Desrosiers, J., Dumas, Y., Solomon, M., and Soumis, F. (1995). Time constrained routing and scheduling. In M. Ball, T. Magnanti, C. Monma, and G. Nemhauser, editors, *Network Routing*, volume 8 of *Handbooks in Operations Research and Management Science*, chapter 2, pages 35–139. Elsevier, Amsterdam.

Dijkstra, E. (1959). A note on two problems in connexion with graphs. *Numerische Mathematik*, **1**, 269–271.

Dohn, A., Kolind, E., and Clausen, J. (2009). The manpower allocation problem with time windows and job-teaming constraints: A branch-and-price approach. *Computers & Operations Research*, **36**, 1145–1157.

Dohn, A., Rasmussen, M., and Larsen, J. (2011). The vehicle routing problem with time windows and temporal dependencies. *Networks*, **58**, 273–289.

Drexl, M. (2007). *On Some Generalized Routing Problems*. Ph.D. thesis, Faculty of Business and Economics, RWTH Aachen University.

Drexl, M. (2012a). Rich vehicle routing in theory and practice. *Logistics Research*, **5**(1), 47–63.

Drexl, M. (2012b). Synchronization in vehicle routing—a survey of VRPs with multiple synchronization constraints. *Transportation Science*, **46**(3), 297–316.

Drexl, M. (2013). Applications of the vehicle routing problem with trailers and transshipments. *European Journal of Operational Research*, **227**(2), 275–283.

Drexl, M. (2014). Branch-and-cut algorithms for the vehicle routing problem with trailers and transshipments. *Networks*, **63**(1), 119–133.

Drexl, M. and Prescott-Gagnon, E. (2010). Labelling algorithms for the elementary shortest path problem with resource constraints considering eu drivers' rules. *Logistics Research*, **2**(2), 79–96.

Drexl, M., Rieck, J., Sigl, T., and Press, B. (2013). Simultaneous vehicle and crew routing and scheduling for partial- and full-load long-distance road transport. *BuR - Business Research*, **6**(2), 242–264.

Dror, M. (1994). Note on the complexity of the shortest path models for column generation in VRPTW. *Operations Research*, **42**(5), 977–978.

Dumas, Y., Desrosiers, J., Gelinas, E., and Solomon, M. (1995). An optimal algorithm for the traveling salesman problem with time windows. *Operations Research*, **43**(2), 367–371.

European Union (2006). Regulation (EC) No 561/2006 of the European Parliament and of the Council of 15 March 2006 on the harmonisation of certain social legislation relating to road transport and amending Council Regulations (EEC) No 3821/85 and (EC) No 2135/98 and repealing Council Regulation (EEC) No 3820/85. Official Journal of the European Union L 102, 11.04.2006.

Favaretto, D., Moretti, E., and Pellegrini, P. (2006). An ant colony system approach for variants of the traveling salesman problem with time windows. *Journal of information and optimization sciences*, **27**(1), 35.

Federal Motor Carrier Safety Administration (2011). Hours of service of drivers. Federal Register Vol. 76, No. 248, pages 81134 - 81188.

Feillet, D., Dejax, P., Gendreau, M., and Guéguen, C. (2004). An exact algorithm for the elementary shortest path problem with resource constraints: Application to some vehicle routing problems. *Networks*, **44**(3), 216–229.

Feillet, D., Gendreau, M., and Rousseau, L.-M. (2007). New refinements for the solution of vehicle routing problems with branch and price. *INFOR*, **45**(4), 239–256.

Fernández, E., Luna-Mota, C., Hildenbrandt, A., Reinelt, G., and Wiesberg, S. (2013). A flow formulation for the optimum communication spanning tree. *Electronic Notes in Discrete Mathematics*, **41**(0), 85 – 92.

Fischer, T. and Merz, P. (2007). A memetic algorithm for the optimum communication spanning tree problem. In *Hybrid Metaheuristics*, pages 170–184. Springer.

Fischetti, M., Lancia, G., and Serafini, P. (2002). Exact algorithms for minimum routing cost trees. *Networks*, **39**(3), 161–173.

Gamache, M., Soumis, F., and Marquis, G. (1999). A column generation approach for large-scale aircrew rostering problems. *Operations Research*, **47**(2), 247–263.

Garcia, R. (2009). *Resource Constrained Shortest Paths and Extensions*. Ph.D. thesis, School of Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta, U.S.A.

Garey, M. and Johnson, D. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman, San Francisco.

Gehring, H. and Homberger, J. (2002). Parallelization of a Two-Phase metaheuristic for routing problems with time windows. *Journal of Heuristics*, **8**(3), 251–276.

Gendreau, M., Hertz, A., Laporte, G., and Stan, M. (1998). A generalized insertion heuristics for the traveling salesman problem with time windows. *Operations Research*, **43**(3), 330–335.

Goel, A. (2009). Vehicle scheduling and routing with drivers' working hours. *Transportation Science*, **43**(1), 17–26.

Goel, A. (2010). Truck driver scheduling in the european union. *Transportation Science*, **44**(4), 429–441.

Goel, A. (2014). Hours of service regulations in the United States and the 2013 rule change. *Transport Policy*, **33**, 48–55.

Goel, A. and Irnich, S. (2014). An exact method for vehicle routing and truck driver scheduling problems. Technical Report No. 33, Jacobs University, School of Engineering and Science, Bremen, Germany.

Goel, A. and Irnich, S. (2016). An exact method for vehicle routing and truck driver scheduling problems. *Transportation Science*. Forthcoming.

Goel, A. and Vidal, T. (2014). Hours of service regulations in road freight transport: An optimization-based international assessment. *Transportation Science*, **48**, 391–412.

Hachemi, N. E., Gendreau, M., and Rousseau, L.-M. (2013). A heuristic to solve the synchronized log-truck scheduling problem. *Computers & Operations Research*, **40**(3), 666–673.

Hewitt, M., Nemhauser, G. L., and Savelsbergh, M. W. (2010). Combining exact and heuristic approaches for the capacitated fixed-charge network flow problem. *INFORMS Journal on Computing*, **22**(2), 314–325.

Hollis, B., Forbes, M., and Douglas, B. (2006). Vehicle routing and crew scheduling for metropolitan mail distribution at Australia Post. *European Journal of Operational Research*, **173**(1), 133–150.

Horváth, M. and Kis, T. (2016). Solving resource constrained shortest path problems with lp-based methods. *Computers & Operations Research*, **73**, 150 – 164.

Hu, T. C. (1974). Optimum communication spanning trees. *SIAM Journal on Computing*, **3**(3), 188–195.

Huber, S. and Geiger, M. (2014). Swap body vehicle routing problem: A heuristic solution approach. In R. González-Ramírez, F. Schulte, S. Voß, and J. Ceroni Díaz, editors, *Computational Logistics*, volume 8760 of *Lecture Notes in Computer Science*, pages 16–30. Springer International Publishing.

Ioachim, I., Gélinas, S., Desrosiers, J., and Soumis, F. (1998). A dynamic programming algorithm for the shortest path problem with time windows and linear node costs. *Networks*, **31**, 193–204.

Irnich, S. (2008). Resource extension functions: Properties, inversion, and generalization to segments. *OR Spectrum*, **30**(1), 113–148.

Irnich, S. and Desaulniers, G. (2005). Shortest path problems with resource constraints. In Desaulniers *et al.* (2005), chapter 2, pages 33–65.

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

Irnich, S., Toth, P., and Vigo, D. (2014). *The Family of Vehicle Routing Problems*, chapter 1, pages 1–33. In Toth and Vigo (2014).

Jans, R. (2010). Classification of Dantzig-Wolfe reformulations for binary mixed integer programming problems. *European Journal of Operational Research*, **204**(2), 251–254.

Jepsen, M., Petersen, B., Spoorendonk, S., and Pisinger, D. (2008). Subset-row inequalities applied to the vehicle-routing problem with time windows. *Operations Research*, **56**(2), 497–511.

Johnson, D. S., Lenstra, J. K., and Rinnooy Kan, A. (1978). The complexity of the network design problem. *Networks*, **8**(4), 279–285.

Kara, I., Koc, O. N., Altıparmak, F., and Dengiz, B. (2013). New integer linear programming formulation for the traveling salesman problem with time windows: minimizing tour duration with waiting times. *Optimization*, **62**(10), 1309–1319.

Kergosien, Y., Lenté, C., Piton, D., and Billaut, J.-C. (2011). A tabu search heuristic for the dynamic transportation of patients between care units. *European Journal of Operational Research*, **214**(2), 442–452.

Kergosien, Y., Lenté, C., Billaut, J.-C., and Perrin, S. (2013). Metaheuristic algorithms for solving two interconnected vehicle routing problems in a hospital complex. *Computers & Operations Research*, **40**(10), 2508 – 2518.

Kim, B.-I., Koo, J., and Park, J. (2010). The combined manpower-vehicle routing problem for multi-staged services. *Expert Systems with Applications*, **37**(12), 8424–8431.

Kim, D. and Pardalos, P. M. (1999). A solution approach to the fixed charge network flow problem using a dynamic slope scaling procedure. *Operations Research Letters*, **24**(4), 195–203.

Kindervater, G. and Savelsbergh, M. (1997). Vehicle routing: Handling edge exchanges. In E. Aarts and J. Lenstra, editors, *Local Search in Combinatorial Optimization*, chapter 10, pages 337–360. Wiley, Chichester.

Kohl, N., Desrosiers, J., Madsen, O., Solomon, M., and Soumis, F. (1999). 2-path cuts for the vehicle routing problem with time windows. *Transportation Science*, **33**(1), 101–116.

Kok, A., Hans, E., and Schutten, J.M.J.and Zijm, W. (2010). A dynamic programming heuristic for vehicle routing with time-dependent travel times and required breaks. *Flexible Services and Manufacturing Journal*, **22**, 83–108.

Kopfer, H. and Meyer, C. M. (2009). A model for the traveling salesman problem including the ec regulations on driving hours. In B. Fleischmann, K.-H. Borgwardt, R. Klein, and A. Tuma, editors, *Operations Research Proceedings 2008: Selected Papers of the Annual International Conference of the German Operations Research Society (GOR) University of Augsburg, September 3-5, 2008*, pages 289–294. Springer Berlin Heidelberg, Berlin, Heidelberg.

Labadie, N., Prins, C., and Yang, Y. (2014). Iterated local search for a vehicle routing problem with synchronization constraints. In *ICORES 2014 - Proceedings of the 3rd International Conference on Operations Research and Enterprise Systems, Angers, Loire Valley, France, March 6-8, 2014*, pages 257–263.

Lahyani, R., Khemakhem, M., and Semet, F. (2015). Rich vehicle routing problems: From a taxonomy to a definition. *European Journal of Operational Research*, **241**(1), 1–14.

Langevin, A., Desrochers, M., Desrosiers, J., Gélinas, S., and Soumis, F. (1993). A two-commodity flow formulation for the traveling salesman and the makespan problems with time windows. *Networks*, **23**, 631–640.

Laporte, G. (2016). Scheduling issues in vehicle routing. *Annals of Operations Research*, **236**(2), 463–474.

Li, J.-Q. (2009). A computational study of bi-directional dynamic programming for the traveling salesman problem with time windows. Technical report, Working paper, University of California, Berkeley, Berkeley.

Liberatore, F., Righini, G., and Salani, M. (2011). A column generation algorithm for the vehicle routing problem with soft time windows. *4OR*, **9**(1), 49–82.

Lozano, L. and Medaglia, A. L. (2013). On an exact method for the constrained shortest path problem. *Computers & Operations Research*, **40**(1), 378 – 384.

Lübbecke, M. and Desrosiers, J. (2005). Selected topics in column generation. *Operations Research*, **53**(6), 1007–1023.

Magnanti, T. and Wolsey, L. (1995). Optimal trees. In M. Ball, T. Magnanti, C. Monma, and G. Nemhauser, editors, *Network Models*, volume 7 of *Handbooks in Operations Research and Management Science*, chapter 9, pages 503–615. Elsevier, Amsterdam.

Mankowska, D. S., Meisel, F., and Bierwirth, C. (2013). The home health care routing and scheduling problem with interdependent services. *Health Care Management Science*, **17**(1), 15–30.

Meisel, F. and Kopfer, H. (2014). Synchronized routing of active and passive means of transport. *OR Spectrum*, **36**(2), 297–322.

Meyer, C. M. and Kopfer, H. (2008). Restrictions for the operational transportation planning by regulations on drivers' working hours. In A. Bortfeldt, J. Homberger, H. Kopfer, G. Pankratz, and R. Strangmeier, editors, *Intelligent Decision Support: Current Challenges and Approaches*, pages 177–186. Gabler, Wiesbaden.

Mingozzi, A., Bianco, L., and Ricciardelli, S. (1997). Dynamic programming strategies for the traveling salesman problem with time window and precedence constraints. *Operations Research*, **45**(3), 365–377.

Morais, V. W., Mateus, G. R., and Noronha, T. F. (2014). Iterated local search heuristics for the vehicle routing problem with cross-docking. *Expert Systems with Applications*, **41**(16), 7495–7506.

Mues, C. and Pickl, S. (2005). Transshipment and time windows in vehicle routing. In *Proceedings of the 8th International Symposium on Parallel Architectures, Algorithms and Networks*, pages 113–119, Piscataway. IEEE.

Ohlmann, J. and Thomas, B. (2007). A Compressed-Annealing heuristic for the traveling salesman problem with time windows. *INFORMS Journal on Computing*, **19**(1), 80–90.

Ortega, F. and Wolsey, L. A. (2003). A branch-and-cut algorithm for the single-commodity, uncapacitated, fixed-charge network flow problem. *Networks*, **41**(3), 143–158.

Padberg, M. W. and Wolsey, L. A. (1983). Trees and cuts. In C. Berge, D. Bresson, P. Camion, J. Maurras, and F. Sterboul, editors, *Combinatorial MathematicsProceedings of the International Colloquium on Graph Theory and Combinatorics*, volume 75 of *North-Holland Mathematics Studies*, pages 511 – 517. North-Holland.

Palmer, C. C. (1994). *An approach to a problem in network design using genetic algorithms*. Ph.D. thesis, Citeseer.

Pecin, D. (2014). *Exact Algorithms for the Capacitated Vehicle Routing Problem*. Ph.D. thesis, Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro.

Pecin, D., Pessoa, A., Poggi, M., and Uchoa, E. (2016). Improved branch-cut-and-price for capacitated vehicle routing. *Mathematical Programming Computation*, pages 1–40.

Peleg, D. and Reshef, E. (1998). Deterministic polylog approximation for minimum communication spanning trees. In K. Larsen, S. Skyum, and G. Winskel, editors, *Automata, Languages and Programming*, volume 1443 of *Lecture Notes in Computer Science*, pages 670–681. Springer Berlin Heidelberg.

Poggi, M. and Uchoa, E. (2014). *New Exact Algorithms for the Capacitated Vehicle Routing Problem*, chapter 3, pages 59–86. In Toth and Vigo (2014).

Potvin, J.-Y. and Bengio, S. (1996). The vehicle routing problem with time windows Part II: Genetic search. *INFORMS Journal on Computing*, **8**(2), 165–172.

Prescott-Gagnon, E., Desaulniers, G., Drexl, M., and Rousseau, L.-M. (2010). European driver rules in vehicle routing with time windows. *Transportation Science*, **44**(4), 455–473.

Pugliese, L. D. P. and Guerriero, F. (2013). A reference point approach for the resource constrained shortest path problems. *Transportation Science*, **47**(2), 247–265.

Rancourt, M.-E. and Paquette, J. (2014). Multicriteria optimization of a long-haul routing and scheduling problem. *Journal of Multi-Criteria Decision Analysis*, **21**(5-6), 239–255. MCDA-12-0037.R1.

Rancourt, M.-E., Cordeau, J.-F., and Laporte, G. (2013). Long-haul vehicle routing and scheduling with working hour rules. *Transportation Science*, **47**(1), 81–107.

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

Righini, G. and Salani, M. (2008). New dynamic programming algorithms for the resource constrained elementary shortest path problem. *Networks*, **51**(3), 155–170.

Roberti, R. and Mingozzi, A. (2014). Dynamic ng-path relaxation for the delivery man problem. *Transportation Science*, **48**(3), 413–424.

Rothlauf, F. (2009). On optimal solutions for the optimal communication spanning tree problem. *Operations Research*, **57**(2), 413–425.

Salani, M. (2005). *Branch-and-price algorithms for Vehicle Routing Problems*. Ph.D. dissertation, Università degli Studi di Milano, Facoltà die Scienze Matematiche, Fisiche e Naturali, Dipartimento di Tecnologie dell'Informatione, Milan, Italy.

Salazar-Aguilar, M. A., Langevin, A., and Laporte, G. (2013). The synchronized arc and node routing problem: Application to road marking. *Computers & Operations Research*, **40**(7), 1708–1715.

Savelsbergh, M. (1992). The vehicle routing problem with time windows: Minimizing route duration. *Operations Research Society of America*, **4**(2), 146–154.

Savelsbergh, M. W. P. and Sol, M. (1998). DRIVE: dynamic routing of independent vehicles. *Operations Research*, **46**, 474–490.

Schneider, M., Stenger, A., and Goeke, D. (2014). The electric vehicle-routing problem with time windows and recharging stations. *Transportation Science*, **48**(4), 500–520.

Sharma, P. (2006). Algorithms for the optimum communication spanning tree problem. *Annals of Operations Research*, **143**(1), 203–209.

Simonetti, N. and Balas, E. (1996). Implementation of a linear time algorithm for certain generalized traveling salesman problems. In W. Cunningham, S. McCormick, and M. Queyranne, editors, *Integer Programming and Combinatorial Optimization*, volume 1084 of *Lecture Notes in Computer Science*, pages 316–329. Springer Berlin Heidelberg.

Smilowitz, K. (2006). Multi-resource routing with flexible tasks: An application in drayage operations. *IIE Transactions*, **38**(7), 555–568.

Solomon, M. (1987). Algorithms for the vehicle routing and scheduling problem with time window constraints. *Operations Research*, **35**(2), 254–265.

Spliet, R. and Gabor, A. (2014). The time window assignment vehicle routing problem. *Transportation Science*.

Steitz, W. and Rothlauf, F. (2012). Using penalties instead of rewards: Solving OCST problems with guided local search. *Swarm and Evolutionary Computation*, **3**, 46–53.

Tilk, C. (2016). Branch-and-price-and-cut for the vehicle routing and truck driver scheduling problem. Technical Report LM-2016-04, Chair of Logistics Management, Gutenberg School of Management and Economics, Johannes Gutenberg University Mainz, Mainz, Germany.

Tilk, C. and Irnich, S. (2016). Dynamic programming for the minimum tour duration problem. *Transportation Science*. Forthcoming.

Tilk, C., Rothenbächer, A.-K., Gschwind, T., and Irnich, S. (2016). Asymmetry helps: Dynamic half-way points for solving shortest path problems with resource constraints faster. Technical Report LM-2016-05, Chair of Logistics Management, Gutenberg School of Management and Economics, Johannes Gutenberg University Mainz, Mainz, Germany.

Toth, P. and Vigo, D., editors (2014). *Vehicle Routing*. MOS-SIAM Series on Optimization. Society for Industrial and Applied Mathematics, Philadelphia, PA.

Vanderbeck, F. (2005). Implementing mixed integer column generation. In Desaulniers *et al.* (2005), chapter 12, pages 331–358.

Vidal, T., Crainic, T. G., Gendreau, M., and Prins, C. (2011). A unifying view on timing problems and algorithms. Technical Report 2011-43, CIRRELT, Montréal, Canada.

Vidal, T., Crainic, T. G., Gendreau, M., and Prins, C. (2013). A hybrid genetic algorithm with adaptive diversity management for a large class of vehicle routing problems with time-windows. *Computers & Operations Research*, **40**(1), 475–489.

Voß, S. (1992). Steiner's problem in graphs: heuristic methods. *Discrete Applied Mathematics*, **40**(1), 45–72.

Wu, B. Y., Chao, K.-M., and Tang, C. Y. (2000). Approximation algorithms for some optimum communication spanning tree problems. *Discrete Applied Mathematics*, **102**(3), 245–266.

Xu, H., Chen, Z., Rajagopal, S., and Arunapuram, S. (2003). Solving a practical pickup and delivery problem. *Transportation Science*, **37**(3), 347–364.

Xue, Z., Zhang, C., Lin, W.-H., Miao, L., and Yang, P. (2014). A tabu search heuristic for the local container drayage problem under a new operation mode. *Transportation Research Part E: Logistics and Transportation Review*, **62**, 136–150.

Zäpfel, G. and Bögl, M. (2008). Multi-period vehicle routing and crew scheduling with outsourcing options. *International Journal of Production Economics*, **113**, 980–996.

Zhang, R., Yun, W. Y., and Kopfer, H. (2010). Heuristic-based truck scheduling for inland container transportation. *OR Spectrum*, **32**(3), 787–808.

Zhang, R., Yun, W. Y., and Kopfer, H. (2013). Multi-size container transportation by truck: modeling and optimization. *Flexible Services and Manufacturing Journal*, **27**(2-3), 403–430.

Zhang, R., Lu, J.-C., and Wang, D. (2014). Container drayage problem with flexible orders and its near real-time solution strategies. *Transportation Research Part E: Logistics and Transportation Review*, **61**, 235–251.