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

Christian Tilk[a], Ann-Kathrin Rothenbächer[a], Timo Gschwind[a], Stefan Irnich[a]

[a] *Chair of Logistics Management, Gutenberg School of Management and Economics, Johannes Gutenberg University Mainz, Jakob-Welder-Weg 9, D-55128 Mainz, Germany.*

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 effect that the number of labels increases strongly with the path length. Bidirectional DP has become a quasi-standard for solving SPPRCs. 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 and introduce 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 workload.

Key words: Shortest path problem with resource constraints (SPPRC), bidirectional labeling algorithms

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 more than 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, 2012; 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

Email addresses: tilk@uni-mainz.de (Christian Tilk), anrothen@uni-mainz.de (Ann-Kathrin Rothenbächer), gschwind@uni-mainz.de (Timo Gschwind), irnich@uni-mainz.de (Stefan Irnich)

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).

Still, for the exact solution of SPPRCs, 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 start 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 path $P' = (P, j) = (s, \ldots, i, j)$ is feasible, an if so,
(4) store it so that it 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 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 impossible 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, extended, and needed 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. 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 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.

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. 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 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 3. The impact of using a dynamic half-way point is computationally evaluated in Section 4. Finally, concluding remarks are given in Section 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 start vertex $s \in V$, the destination 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 let $v(F_i) = i$. Then, for any arc $(i, j) \in A$ and any 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 let $v(B_j) = j$. Then, for any arc $(i, j) \in A$ and any 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 $[L, U]$.

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 1 shows a generic version of a labeling algorithm for solving a 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 callDominanceAlgorithm(), and the function getNextDirection() that controls whether the next label to extend is a forward or a backward label.

We comment on Algorithm 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. Initial labels F and B are created for the trivial forward and 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 $F \neq null \wedge B = null$, and $backward$ if $F = null \wedge B \neq null$, while in case of $F, B \neq null$ both directions are possible. In Steps 5 and 11, it is guaranteed that only labels respecting the half-way criterion are processed. 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 deposition of the new label F_j so that it can later be retrieved via getNextForwardLabel(). The

Algorithm 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 ≠ null or B ≠ null do
3      direction := getNextDirection();
4      if direction = forward then
5          if F^res ≤ H_F then
6              for (i, j) ∈ A with i = v(F) do
7                  Propagate F to vertex j with REF f_ij;
8          H_B := max{H_B, min{F^res, H_F}};
9          F := getNextForwardLabel();
10     else
11         if B^res > H_B then
12             for (i, j) ∈ A with j = v(B) do
13                 Propagate B to vertex i with REF b_ij;
14         H_F := min{H_F, max{B^res, H_B}};
15         B := getNextBackwardLabel();
16     if callDominanceAlgorithm() then
17         Apply dominance rules;
```

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 depending on the value of the monotone resource of the currently processed label. For the version with the dynamic half-way point, the update is only correct if the order in which labels are processed is monotone as explained in the next paragraphs.

In Steps 9 and 15, we assume that all labels are managed with a data structure that allows the efficient retrieval of the next label to process. 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. If no more forward/backward label is unprocessed, the function getNextForwardLabel()/getNextBackwardLabel() returns the value *null*.

In Steps 16 and 17, the invocation of a dominance algorithm can be delayed to some convenient point in time. In particular, the determination of dominated labels, every time a new feasible label is created, may not be efficient. The function callDominanceAlgorithm() may call a dominance algorithm only when there is the chance that the next label to process is dominated.

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. Moreover, the backward half-way point H_B can only be increased. This can happen if 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). Thus, depending on the initial values of H_F and H_B, the resulting Algorithm 1 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 in 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 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, already extended labels would fail the half-way test (Step 5 or 11) later in the algorithm meaning that redundant labels are created. 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 beyond H_B/H_F have already been extended. In the course of Algorithm 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. However, the difference to a classical bidirectional labeling algorithm is that the half-way point was 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 1.

An ideal strategy to choose a direction would be one in which the overall number of processed forward and backward labels is minimal. 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.

Pre-test have shown that on average the last strategy outperforms the two others.

When Algorithm 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.

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 differs significantly.

3.1. Vehicle Routing Problem with Time Windows (VRPTW)

The VRPTW is an extension of the classical 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 (Desaulniers *et al.*, 2014; Baldacci *et al.*, 2012; Desaulniers *et al.*, 2010; Cordeau *et al.*, 2002). Different branch-and-price-and-cut algorithms were proposed for the VRPTW in the literature (e.g., Desaulniers *et al.*, 2008; Irnich and Villeneuve, 2006; Kohl *et al.*, 1999; Desrochers *et al.*, 1992).

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 start 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). 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 destination depot t to the start depot 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 start depot s and b_t is the latest arrival time at the destination depot 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.

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.

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 start depot 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 start depot s, the destination depot 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 j;

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.

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. They consider U.S. hours of service regulations and 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 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 1: Parameters imposed by the new U.S. hours of service regulations (Table 1 in Goel and Irnich, 2016)

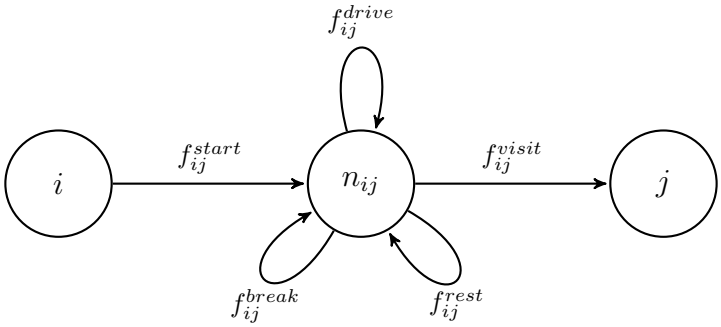

Figure 1: Subnetwork for arc $(i, j) \in A$ (similar to Figure 1 in Tilk, 2016)

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 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 1. 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 start depot 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;

$F_i^{el|R}$: 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 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.

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.

We run two different types of tests. In the first 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. 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. 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.

In the second type of test, we solve every instance of the SPPRC pricing problem with both the static and the dynamic half-way point in each iteration of the linear relaxation of the master program. Since both versions are here 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.

The following subsections report the results for the different VRP variants along with the settings and instances they were obtained with.

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.

Table 2 shows the results for the first type of test, i.e., the comparison between static and dynamic half-way points for solving the linear relaxation. The values are aggregated according to the 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.

The results of the second type of test, where labeling with static and dynamic half-way points is applied with identical reduced cost, are summarized in Table 3. Herein, the results are also aggregated per instance

Instance group	No. solved		Avg time	
	static	dynamic	static	dynamic
Solomon	56	56	118.64	111.24
Gehring & Homberger	46	49	674.32	466.51
all	102	105	369.24	271.46

Table 2: VRPTW: Comparison of computation times using static vs. dynamic half-way points

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).

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	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
Gehring & Homberger	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
Gehring & Homberger (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 3: VRPTW: Ratios for using the dynamic and static half-way point with identical reduced cost

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.

Table 4 shows the results for the first type of test. 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	No. solved		Avg time	
	static	dynamic	static	dynamic
25	55	56	5.36	2.68
50	54	55	324.50	240.46
100	41	42	692.63	517.26
Series 1	87	87	137.14	116.48
Series 2	63	66	541.80	382.08
all	150	153	305.53	227.00

Table 4: EVRPTW: Comparison of computation times using static vs. dynamic half-way points

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* meaning 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.

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: EVRPTW: Ratios for using the dynamic and static half-way point with identical reduced cost

Table 5 summarizes the results of the second 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. Again, 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 second 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.

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 1 as soon as 500 negative reduced-cost routes have been found.

Instance group	No. solved		Avg time	
	static	dynamic	static	dynamic
25	56	56	96.80	39.11
50	51	54	702.08	388.32
100	19	26	2123.98	1023.38
all	126	136	646.60	327.92

Table 6: VRDTSP: Comparison of computation times using static vs. dynamic half-way points

Table 6 reports the result of the first 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.

The results of the second type of test are summarized in Table 7. 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. Moreover, 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. 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 7: VRDTSP: Ratios for using the dynamic and static half-way point with identical reduced cost

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 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 is very minor. Thus, a significant effect can be achieved with little implementation effort.

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.

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., 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., 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*, chapter 5, pages 119–159. In Toth and Vigo (2014).

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.

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.

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 G. Desaulniers, J. Desrosiers, and M. Solomon, editors, *Column Generation*, chapter 2, pages 33–65. Springer, New York, NY.

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.

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).

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.

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.