Resource-Window Reduction by Reduced Costs in Path-based Formulations for Routing and Scheduling Problems

Nicola Bianchessi*,[a,b], Timo Gschwind[c], Stefan Irnich[b]

[a] *Department of Computer Science "Giovanni degli Antoni", Università degli Studi di Milano, Via Celoria 18, 20133 Milan, Italy.*
[b] *Chair of Logistics Management, Gutenberg School of Management and Economics, Johannes Gutenberg University Mainz, Jakob-Welder-Weg 9, 55128 Mainz, Germany.*
[c] *Chair of Logistics, Department of Business Studies and Economics, Technical University of Kaiserlautern, Gottlieb-Daimler-Straße 42, 67663 Kaiserlautern, Germany.*

Abstract

Many routing and scheduling problems are modeled through variables that represent paths (routes, schedules, etc.). For such extensive formulations, branch-price-and-cut (BPC) algorithms constitute nowadays the leading exact solution technique and, most of the time, the pricing problem is a shortest-path problem with resource constraints that can be solved by a dynamic-programming labeling algorithm. For this setting, variable-fixing techniques based on the reduced costs of the paths have been proposed with the aim of eliminating arcs from the underlying network and speeding up the solution process of the pricing problem as well as of the overall BPC algorithm. For an efficient variable fixation, bidirectional labeling must be possible. We move one step forward and show how the reduced costs of paths can also be exploited to reduce the resource windows for many types of resources including the time resource and a load-related resource. This can be achieved without modifying the pricing problem network and altering the structure of the pricing problem itself. Moreover, different resources can be considered simultaneously. A straightforward reduction of the resource windows associated with the vertices of the network can tighten them, but this reduction does not translate into savings in computation times. On the contrary, the reduction of the resource windows is effective when distinct forward and backward resource windows are defined for each arc, and reduced independently based on the traversal direction of the arc itself. Moreover, an arc can be eliminated when one of its arc-specific resource windows becomes empty, and the explicit use of variable-fixing techniques can be avoided. Computational results obtained for benchmark instances of the vehicle routing problem with time windows show that the overall computation times of the BPC algorithm can be significantly reduced compared to a fully-fledged BPC algorithm using variable-fixing techniques.

Key words: integer programming; column generation; variable fixing; resource windows reduction; vehicle routing

1. Introduction

Branch-price-and-cut (BPC) algorithms constitute the leading exact methodology for solving many *vehicle-routing problems* (VRPs, Toth and Vigo, 2014) and vehicle- and crew-scheduling problems (Huisman *et al.*, 2005). A BPC algorithm is a branch-and-bound algorithm in which the dual bounds (lower bounds, we assume cost minimization) are computed by column generation and cuts are added dynamically to strengthen the linear relaxations (Costa *et al.*, 2019). Column generation alternates between solving a

*Corresponding author.
Email addresses: nicola.bianchessi@unimi.it (Nicola Bianchessi), gschwind@wiwi.uni-kl.de (Timo Gschwind), irnich@uni-mainz.de (Stefan Irnich)

restricted master problem (RMP), which is the linear relaxation of a restricted version of an *extensive* model comprising only a subset of the variables, and solving a pricing problem that generates missing variables. For most applications mentioned above, the pricing problem is an elementary *shortest-path problem with resource constraints* (SPPRC) that can be solved by a dynamic-programming labeling algorithm (see, Irnich and Desaulniers, 2005). Often, the largest share of the overall computational effort lies in the solution of these SPPRCs. Accordingly, research on problem-specific SPPRC labeling algorithms is still very active. Complementary to this, conceptual and algorithmic improvements for general BPC algorithms have been presented over the years. Indeed, the leading BPC algorithms are highly comprehensive and include several algorithmic components to accelerate the solution process. The probably most sophisticated and powerful BPC implementations have been developed for the *capacitated VRP* (CVRP, Pecin *et al.*, 2017a) and the *VRP with time windows* (VRPTW, Pecin *et al.*, 2017b), and various other VRP variants (Pessoa *et al.*, 2020) including algorithmic components such as bidirectional labeling, *ng*-route relaxation, limited-memory subset-row inequalities, route enumeration, and variable fixing.

The focus of this paper is on refinements of variable-fixing techniques using reduced costs. Variable fixing refers to a collection of techniques to identify variables of a mathematical optimization problem that must assume a unique value in every feasible or optimal solution to the problem. The following proposition seems to be folklore:

Proposition 1. *Let UB be an upper bound on the optimal value of the minimization problem M, and let π be a dual solution to the linear relaxation of M providing a lower bound $LB(\pi)$. If an integer variable $x \geq 0$ with reduced cost $\tilde{c}_x(\pi)$ in the linear relaxation fulfills $\tilde{c}_x(\pi) > UB - LB(\pi)$, then $x = 0$ in every optimal solution to M, i.e., x can be eliminated.*

Variables known to assume $x = 0$ can be eliminated completely (*variable elimination*). Nemhauser and Wolsey (1988, Proposition 2.1, page 389) consider bounded integer variables x with $L \leq x \leq U$ and highlight that those that are non-basic in the linear relaxation, in which case $x = L$ or $x = U$ holds, can be set to the respective value if $\tilde{c}_x(\pi) > UB - LB(\pi)$ or $-\tilde{c}_x(\pi) > UB - LB(\pi)$, respectively. To be more precise, Nemhauser and Wolsey only state that there exists an optimal solution with $x = L$ (or $x = U$), but it seems to be common knowledge that this is also a necessary property in the sense that a solution with $x \neq L$ (or $x \neq U$, respectively) cannot be optimal. In this case, x can be replaced by the constant value L (or U) (*variable fixing*).

In this work, we focus on variable elimination but we anyway speak of variable fixing. For the sake of simplicity, we assume a (possibly mixed) integer linear minimization problem M in the following. The application of Proposition 1 in BPC algorithms is not straightforward. After eliminating a variable, the re-generation of the same variable by the pricing problem has to be forbidden. Forbidding the re-generation of one or several variables changes the structure of the pricing problem. In routing and scheduling applications, where the variables typically represent paths (routes, schedules, etc.) and the pricing subproblem is solved by a dynamic-programming labeling algorithm, the modified pricing problem that forbids the re-generation is typically difficult (Villeneuve and Desaulniers, 2005). The idea of Irnich *et al.* (2010) was to not directly apply the proposition to individual variables but to consider sets of variables, i.e., paths, defined by some property. If all variables with a certain property can be eliminated, the consequence is that this property does not hold for optimal solutions. Two fundamental questions arise in this context:

(Q1) Which properties are helpful to accelerate the BPC algorithm?

(Q2) How can it be proved efficiently that all variables fulfilling a given property can be eliminated?

Since pricing typically consumes a large share of the total run time of a BPC algorithm for a variant of the VRP, the answer to question (Q1) is obvious: Variable fixing must use a property that finally simplifies the pricing. Irnich *et al.* (2010) argued that a structurally identical but simpler-to-solve pricing problem is one with fewer arcs. Accordingly, the properties that were considered are the existence of a given arc (i, j) in the path. If all variables for paths containing arc (i, j) have a sufficiently large reduced cost, the arc (i, j) is redundant and can therefore be eliminated from the pricing network. It was shown that, with tight dual bounds, this technique can reduce the pricing network by more than 90 percent leaving only a kind of "core" pricing problem.

The answer to question (Q2) is more technical as it refers to the solution approach used for solving the pricing subproblem. Ideally, the elimination should result from information already available from the pricing process. For variables representing paths and the solution of the pricing problem with dynamic-programming labeling, we exploit the non-dominated labels. However, we are going to see that variable elimination does not come for free. It is a by-product of bidirectional labeling, but additional effort is required to compute property-related reduced costs. As a result, we observe a tradeoff between the frequency of elimination trials and its actual impact on simplifying the pricing.

The contributions of this work can be summarized as follows:

- We consider properties related to the resource consumption of paths. We show that reduced costs of paths visiting a given vertex i can be exploited to reduce the resource windows of i for many types of resources including time and a load-related resource.
- We further show that arc-specific properties that simultaneously consider a given arc (i, j) and two resource windows at the endpoints i and j lead to more effective resource-window reduction procedures. The idea to consider combinations of arcs (i, j) and specific resource values is a fundamental ingredient of the arc-load network model for the CVRP used in (Pecin *et al.*, 2017a). While the arc-load network of Pecin *et al.* is tailored to the CVRP (one resource only), our arc-specific properties can easily be used also in rich and complex VRP variants that may be defined using a larger number of attributes/resources (e.g., Desaulniers *et al.*, 2016; Altman *et al.*, 2021).
- We perform a computational study on the Solomon benchmark instances of the VRPTW showing that the overall computation times of the BPC algorihtm can be significantly reduced with the help of arc-specific resource-window reductions. This is still true in comparison to a fully-fledged BPC algorithm implementation using arc fixing. In particular, more difficult VRPTW instances characterized by long computation times benefit more.

The remainder of this work is structured as follows: In Section 2, we lay the theoretical and algorithmic foundations of resource-window reduction. The formal definition of the VRPTW and the description of the BPC algorithm follow in Section 3. In Section 4, we present the computational study that shows the effect of different types of resource-window reduction on the performance of the BPC algorithm. Final conclusions are drawn in Section 5.

2. Theory of Resource-Window Reduction

In this section, we formally define the prerequisites of variable fixing and explain how reduced costs of paths can be used to tighten resource windows for resources that are used to describe feasible paths. The starting point is the definition of the master program which comprises at least variables for paths (representing routes or schedules) and task-fulfillment constraints. Let $\mathcal{P}$ denote the set of all resource feasible paths. The following set-partitioning type of master program has integer variables λ_p, one for each path $p \in \mathcal{P}$ indicating how often p is used in the solution. The cost of path $p \in \mathcal{P}$ is denoted c_p. Moreover, we assume that a given set K of tasks needs to be fulfilled (by exactly one path). To this end, let the integer coefficient a_{kp} indicate whether (or how often) task $k \in K$ is performed by path $p \in \mathcal{P}$. The master program reads as follows:

$$\min \sum_{p \in \mathcal{P}} c_p \lambda_p \tag{1a}$$

$$\text{s.t.} \sum_{p \in \mathcal{P}} a_{kp} \lambda_p = 1 \qquad \forall k \in K \tag{1b}$$

$$\lambda_p \geq 0 \quad \text{integer} \tag{1c}$$

The objective (1a) is the minimization of the total costs of all routes or schedules. Constraints (1b) enforce the fulfillment of every task. The domains of the variables are stated in (1c).

Besides these basic ingredients, the master program may contain further variables and constraints. The unified framework of Desaulniers *et al.* (1998) gives an impression of such extensions. In particular, additional constraints can either model requirements such as fleet-size, balancing, or service-level constraints or

constraints that are added to strengthen the linear relaxation of the master program. For the latter, the literature (see, e.g., Costa *et al.*, 2019) distinguishes between robust constraints, e.g., capacity cuts, and non-robust cuts, e.g., subset-row inequalities (Jepsen *et al.*, 2008).

For the following, the vanilla version of the master program given by (1) suffices for exactly describing our new methods. Column generation iteratively solves the linear relaxation ($\lambda_p \geq 0$ and $\lambda_p \in \mathbb{R}$) of RMPs, in which only a subset $\mathcal{P}' \subset \mathcal{P}$ of the paths and associated variables $\lambda_p, p \in \mathcal{P}'$ are present. We assume that a dual feasible solution π to an RMP is given. With $\tilde{c}_p(\pi)$ denoting the reduced cost of a path $p \in \mathcal{P}$, the pricing problem is defined as $z(\pi) = \min_{p \in \mathcal{P}} \tilde{c}_p(\pi)$. Any path $p \in \mathcal{P}$, with negative reduced cost $\tilde{c}_p(\pi)$, is a solution to the pricing problem. It is found as an origin-destination-path in a problem-specific network $D = (V, A)$ with vertex set V and arc set A fulfilling additional constraints. For an overview of most common resource constraints we refer to (Desaulniers *et al.*, 1998; Irnich, 2008). If $z(\pi) \geq 0$, the solution of the RMP is the solution to the linear relaxation of (1), and its objective value is the associated dual bound $LB(\pi)$.

Next, we formalize the idea of properties defining subsets of paths. We use the symbol $\mathcal{P}[prop]$ to refer the subset of paths $\mathcal{P}$ that fulfill a given property *prop*. For example, two important properties are:

[ij]: For an arc $(i, j) \in A$, the path includes the arc (i, j) at least once;

[hij]: For two arcs $(h, i), (i, j) \in A$, the path includes the sequence (h, i, j) at least once;

For these and other properties *prop*, we define the short-hand notation

$$\tilde{c}[prop](\pi) = \min_{p \in \mathcal{P}[prop]} \tilde{c}_p(\pi) \qquad \text{and} \qquad LB[prop](\pi) = LB(\pi) + \tilde{c}[prop](\pi) \tag{2}$$

to refer to the minimum reduced cost and the associated *property-specific lower bound* $LB[prop](\pi)$.

For the property $[ij]$ and the given arc $(i, j) \in A$, Proposition 1 allows the following arc-fixing: If the property-specific lower bound $LB[ij](\pi)$ fulfills $LB[ij](\pi) > UB$, then an optimal solution cannot contain any of the paths $p \in \mathcal{P}[ij]$. As a result, the arc $(i, j) \in A$ can be eliminated from (V, A) as first suggested by Irnich *et al.* (2010).

We next explain how to efficiently find all arcs (i, j) that can be eliminated, i.e., we provide an answer to question (Q2). To this end, bidirectional labeling (Righini and Salani, 2006; Tilk *et al.*, 2017) must be possible. Describing the prerequisites of bidirectional labeling requires some additional notation and assumptions that we introduce now. Solutions to the pricing problem are o-o'-path $p = (v_0 = o, v_1, \ldots, v_\ell = o')$ with given source and sink vertices o and $v_\ell = o'$. Algorithm 1 provides a high-level synopsis of a mono-directional forward labeling algorithm. Forward labeling starts with an initial forward label at vertex $v_0 = o$ characterized by attributes T_o in Step 1. In Step 4, so-called *resource extension functions* (REFs) f_{ij} for each arc $(i, j) \in A$ are used to propagate the attributes of partial paths through the network (V, A). The values T_i of the attributes of a partial path reaching vertex i are stored in a *label*. Labels do not only store attributes but typically also include a pointer to the predecessor label (if existent) and may contain further auxiliary data to support fast dominance and fast label extensions. Nevertheless, we interchangeably use attributes T and labels F in REFs and in direct comparisons of attribute values. This slight abuse of notation helps to keep the description simple and intuitive.

For an arc $(i, j) \in A$ and a label $F \in \mathcal{F}_i$ for the partial path p_F ending at the tail vertex i, the REF f_{ij} computes the resulting values $f_{ij}(F)$ at the head vertex j. If the associated partial path (p_F, j) is infeasible, we assume that the result of $f_{ij}(F)$ is ∞. Moreover, all REFs properly transfer the infeasibility information by guaranteeing $f_{ij}(\infty) = \infty$. Note that it is implementation dependent when and how often a dominance algorithm is invoked in Step 9. Well-performing implementations use a test in Step 8 that postpones the application of the dominance algorithm as long as possible, but guarantees that no label that has already been extended is later eliminated by dominance.

Backward labeling starts with an initial backward label at vertex $v_\ell = o'$ characterized by attributes $T_{o'}$. Likewise, backward REFs b_{ij} for each arc $(i, j) \in A$ are used to propagate the attributes of partial paths through the network (V, A), but in reverse direction, i.e., from j to i. For a backward label $B \in \mathcal{B}_j$ for the partial path p_B ending at the head vertex j, the REF b_{ij} computes the resulting label $b_{ij}(B)$ at the tail vertex i. Also here, we write $b_{ij}(B) = \infty$ to signal infeasibility, and $b_{ij}(\infty) = \infty$ is ensured.

Algorithm 1: Forward Labeling Algorithm for SPPRCs

Input: Digraph (V, A), source vertex $o \in V$, and forward REFs f_{ij} for $(i, j) \in A$

$F \leftarrow F(T_o)$;
while $F \neq null$ **do**
  for $(i, j) \in A$ *with* $i = v(F)$ **do**
    $F' \leftarrow f_{ij}(F)$;
    if $F' < \infty$ **then**
      Store F' in $\mathcal{F}_j$;
  $F \leftarrow$ getNextForwardLabel();
  if callDominanceAlgorithm() **then**
    Apply dominance ;

Output: Labels $\mathcal{F}_i$ for all $i \in V$

Algorithm 2: Backward Labeling Algorithm for SPPRCs

Input: Digraph (V, A), sink vertex $o' \in V$, and backward REFs b_{ij} for $(i, j) \in A$

$B \leftarrow B(T_{o'})$;
while $B \neq null$ **do**
  for $(i, j) \in A$ *with* $j = v(B)$ **do**
    $B' \leftarrow b_{ij}(B)$;
    if $B' < \infty$ **then**
      Store B' in $\mathcal{B}_i$;
  $B \leftarrow$ getNextBackwardLabel();
  if callDominanceAlgorithm() **then**
    Apply dominance ;

Output: Labels $\mathcal{B}_i$ for all $i \in V$

Algorithm 2 summarizes the mono-directional backward labeling algorithm. Note the perfect symmetry between Algorithms 1 and 2.

Bidirectional labeling additionally requires a monotone resource like the resource *time*, i.e., one that is non-decreasing in forward propagation and non-increasing in backward propagation. A convenient value in the "middle" of the resource's domain, a so called *half-way point* H then restricts the forward propagation to labels F with $F^{res} \leq H$ and backward labels B with $B^{res} > H$; for details we refer to (Righini and Salani, 2006; Tilk *et al.*, 2017).

For arc-fixing and resource-window reduction, the half-way point is irrelevant. In contrast, arc-fixing and resource-window reduction techniques assume that the pricing problem for the dual price π has been solved completely and twice, once in forward direction and once in backward direction. We further assume that, as a result of this labeling processes, the sets of all non-dominated forward labels $\mathcal{F}_i$ and all non-dominated backward labels $\mathcal{B}_i$ are available for each vertex $i \in V$.

Under these assumptions, we describe how forward labels are joined with appropriate backward labels. For a forward label $F \in \mathcal{F}_i$ and a backward label $B \in \mathcal{B}_i$ associated with vertex i, let p_F and p_B be the associated partial paths. Each bidirectional labeling algorithm has a *merge operator* m that checks the feasibility of the concatenated path (p_F, p_B) and computes its reduced cost. It is convenient to define $m(F, B)$ as the resulting reduced cost of path (p_F, p_B) if it is feasible, and $+\infty$ otherwise. Moreover, we guarantee $m(\infty, B) = m(F, \infty) = \infty$.

As suggested by Irnich *et al.*, the value $\tilde{c}[ij](\pi)$ can be effectively computed for all arcs $(i, j) \in A$. With the formal definition of the REFs f_{ij}, b_{ij}, and the merge operator m, it is:

$$\tilde{c}[ij](\pi) = \min_{\substack{F \in \mathcal{F}_i, \\ B \in \mathcal{B}_j}} m(f_{ij}(F), B) = \min_{\substack{F \in \mathcal{F}_i, \\ B \in \mathcal{B}_j}} m(F, b_{ij}(B)) \tag{3}$$

Eqs. (3) show that one can choose in which way the value $\tilde{c}[ij](\pi)$ is computed, either with attributes $f_{ij}(F)$ and B referring to the tail vertex j, or with attributes F and $b_{ij}(B)$ referring to the head vertex i.

The recent work of Desaulniers *et al.* (2020) considers another extension of arc fixing considering sequences of two consecutive arcs, i.e., property $[hij]$ for given arcs $(h, i), (i, j) \in A$. The associated paths traverse the vertices h, i, and j consecutively. If $LB[hij](\pi) > UB$ holds, then Proposition 1 shows that no optimal path exists in which the arcs (h, i) and (i, j) are traversed in direct sequence. In this case, no direct arc elimination is possible. However, Desaulniers *et al.* have shown that a so-called *two-arc fixing* can be incorporated into the forward and backward labeling algorithms. Since the associated labeling process is more technical and we do not rely on two-arc fixing in the following, we omit a detailed explanation. Instead, we study properties related to resource windows implied by time-window constraints and load constraints.

2.1. Time-Window Constraints

Time-window constraints are very important in vehicle routing and crew scheduling. For each vertex $i \in V$, let $[e_i, \ell_i]$ denote its time window, i.e., the time interval in which i can be visited or in which the service

at i can feasibly start. Moreover, let τ_{ij} be the time consumed when directly traveling from i to j for each arc $(i,j) \in A$. The standard assumption is that a potential service duration at vertex i is included in the travel time τ_{ij}, and that waiting before serving j is possible. The latter can happen if $e_i + \tau_{ij} < e_j$. The reduction of time windows goes beyond classical, feasibility-based time-window techniques as presented in (Desrochers *et al.*, 1992) and used in almost all subsequent works.

In the following, we use three symbols for resource-related information: F and B denote the attribute vectors of the labels $F \in \mathcal{F}_i$ and $B \in \mathcal{B}_i$ that result from the forward and backward labelling process at a vertex $i \in V$, respectively, and T_i for an arbitrary attribute vector. (We often leave out the index i if the context is clear.)

If we want to compute $\tilde{c}[prop](\pi)$ for some time-window related property *prop* in a similar way as in (3), we must clarify the relationship between arbitrary paths with property *prop* and the forward and backward labels $F \in \mathcal{F}_i$ and $B \in \mathcal{B}_i$. The feasibility of a path regarding time-window constraints is typically tested by computing an as-early-as-possible schedule in forward labeling and by computing an as-late-as-possible schedule in backward labeling. The standard forward and backward resource extension functions that are, e.g., used in the VRPTW, propagate a time attribute T^{time} in the following way:

$$T_j^{time} = f_{ij}^{time}(T_i) = \max\{e_j, T_i^{time} + \tau_{ij}\} \qquad \text{feasible if} \quad T_j^{time} \leq \ell_j \tag{4a}$$

$$T_i^{time} = b_{ij}^{time}(T_j) = \min\{\ell_i, T_j^{time} - \tau_{ij}\} \qquad \text{feasible if} \quad T_i^{time} \geq e_i \tag{4b}$$

We study three properties of resource-feasible paths $p \in \mathcal{P}$ related to a vertex $i \in V$ and an arbitrary point in time $t \in \mathbb{R}$ with the intention to manipulate the time window $[e_i, \ell_i]$.

$[T_i^{time} < t]$: The path can service/visit vertex i before t; if i occurs several times in path p, at least one of the services must start strictly before time t;

$[T_i^{time} > t]$: Likewise with a service start/visit at vertex i at a time strictly after time t;

$[T_i^{time} = t]$: Likewise with a service/visit at vertex i starting exactly at time t;

The standard less-or-equal dominance regarding the as-early-as-possible schedule for forward labels implies that we can only make verified statements about service starts before a given point in time t. Conversely, backward labels can only be used to make verified statements for service starts after time t. Verified statements about exact service start times then result from the combination of the two types of labels. This is summarized in the following proposition.

Proposition 2. *Let a dual feasible solution π be given. We further assume that a complete forward and backward labeling has been performed producing forward label sets $\mathcal{F}_i$ and backward label sets $\mathcal{B}_i$ for all vertices $i \in V$. Then, the reduced cost of the properties $[T_i^{time} < t]$, $[T_i^{time} > t]$, and $[T_i^{time} = t]$, can be computed as follows:*

$$\tilde{c}[T_i^{time} < t](\pi) = \min_{\substack{F \in \mathcal{F}_i : F^{time} < t \\ B \in \mathcal{B}_i}} m(F,B) \tag{5a}$$

$$\tilde{c}[T_i^{time} > t](\pi) = \min_{\substack{F \in \mathcal{F}_i, \\ B \in \mathcal{B}_i : B^{time} > t}} m(F,B) \tag{5b}$$

$$\tilde{c}[T_i^{time} = t](\pi) = \min_{\substack{F \in \mathcal{F}_i : F^{time} \leq t \\ B \in \mathcal{B}_i : B^{time} \geq t}} m(F,B) \tag{5c}$$

Note: *Similar equalities as (5a) and (5b) are valid when $<$ is replaced by $\leq$ and $>$ is replaced by $\geq$.*

Proof. We first prove equality (5a). Let p be an optimal o-o'-path visiting vertex i with resource value T_i at i fulfilling $T_i^{time} < t$, i.e., p has minimal reduced cost in the left-hand-side of (5a). This path decomposes into $p = (p_F, p_B)$, where p_F is the corresponding o-i-path and p_B the corresponding i-o'-path. Let T_i' be the attribute vector of label F' that results from propagating forward the initial attributes T_o along p_F. Likewise, let T_i'' be the attribute vector of label B'' that results from backward propagating the initial attributes $T_{o'}$ along p_B. Then, the merge produces the minimum reduced cost $m(F', B'') = \tilde{c}_p(\pi)$ and $T_i'^{time} \leq T_i^{time} \leq T_i''^{time}$ holds. We now know that $T_i'^{time} \leq T_i^{time} < t$ is fulfilled. (However, nothing is

known about the relationship between $T_i''^{time}$ and t.) Due to the dominance principle, there exists a label $F \in \mathcal{F}_i$ identical to or dominating F' and a label $B \in \mathcal{B}_i$ identical to or dominating B''. Since domination and merge must be compatible, the relationship

$$m(F, B) \leq m(F', B'') = \tilde{c}_p(\pi) = \tilde{c}[T_i^{time} < t](\pi)$$

holds. Finally, the dominance principle also guarantees the first inequality in $F^{time} \leq T_i'^{time} \leq T_i^{time} < t$, which establishes equality in the above chain and proves the statement.

Equality (5b) follows from (5a) with exchanged roles of forward and backward labels.

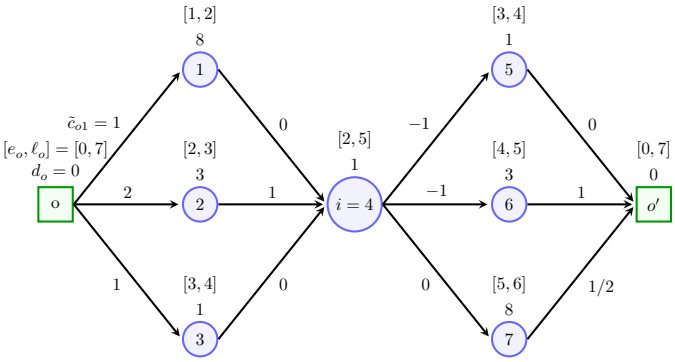
Equality (5c) can be proved similarly with the same techniques. For the sake of brevity, we omit the formal proof. □

It is now straightforward to use the properties $[T_i^{time} < t]$ and $[T_i^{time} > t]$ to perform a reduced cost-based time-window reduction.

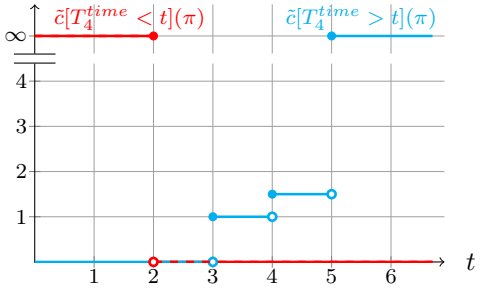
Proposition 3. *Let a dual feasible solution π be given. The following two rules are valid:*
(i) If $LB[T_i^{time} < t](\pi) > UB$, the time window can be reduced to $[t, \ell_i] \subset [e_i, \ell_i]$.
(ii) If $LB[T_i^{time} > t](\pi) > UB$, the time window can be reduced to $[e_i, t] \subset [e_i, \ell_i]$.

Proof. This is a consequence of Propositions 1 and 2. □

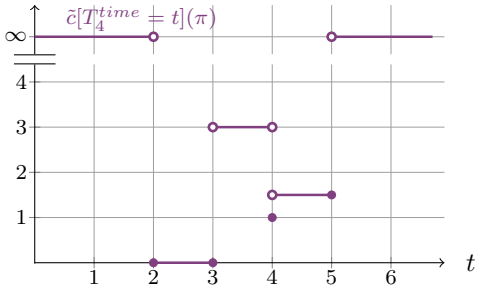
Example 1. *We consider the network (V, A) depicted in Figure 1a with twelve arcs and nine vertices, where $o = 0$ is the origin and $o' = 8$ is the destination. Three resources are relevant:* time, load, *and reduced cost* rdc. *It is assumed that all travel times are identical with $\tau_{ij} = 1$, while reduced costs $\tilde{c}_{ij}$ are shown next to each arc $(i, j) \in A$. Moreover, demands d_i are depicted next to each vertex. Time is constrained by time windows $[e_i, \ell_i]$ at each vertex and load by a global vehicle capacity $Q = 10$.*



(a) Network (V, A) with reduced costs on arcs and time windows and demands on vertices; all travel times are 1.



(b) Function $t \mapsto \tilde{c}[T_4^{time} < t](\pi)$ (in red) and function $t \mapsto \tilde{c}[T_4^{time} > t](\pi)$ (in cyan)



(c) Function $t \mapsto \tilde{c}[T_4^{time} = t](\pi)$

Figure 1: Example for time-window reduction; focus is on vertex $i = 4$.

There are nine different o-o′-paths in the network. Five of them are feasible (see Table 1a) and four are infeasible (see Table 1b). We focus now on vertex $i = 4$ in the middle of the network. Since the set of all

feasible paths $\mathcal{P}$ is explicitly known, we can compute for each $p \in \mathcal{P}$ the possible start times T_4^{time} of service at $i = 4$ and the reduced cost $\tilde{c}_p(\pi)$. With this data, the computation of the functions $t \mapsto \tilde{c}[T_4^{time}(\pi) < t]$, $t \mapsto \tilde{c}[T_4^{time}(\pi) > t]$, and $t \mapsto \tilde{c}[T_4^{time}(\pi) = t]$ is straightforward. The functions are depicted in Figures 1b and 1c.

Feasible path $p \in \mathcal{P}$	Possible values T_4^{time}	Reduced cost $\tilde{c}_p(\pi)$
$(0, 1, 4, 5, 8)$	$[2, 3]$	0
$(0, 2, 4, 5, 8)$	$[3, 3]$	2
$(0, 2, 4, 6, 8)$	$[3, 4]$	3
$(0, 3, 4, 6, 8)$	$[4, 4]$	1
$(0, 3, 4, 7, 8)$	$[4, 5]$	1.5

(a) Feasible paths

Infeasible path	Reason for infeasibility
$(0, 1, 4, 6, 8)$	load $\sum d_i = 12$
$(0, 1, 4, 7, 8)$	load $\sum d_i = 17$
$(0, 2, 4, 7, 8)$	load $\sum d_i = 12$
$(0, 3, 4, 5, 8)$	time window at vertex 5

(b) Infeasible paths

Table 1: Feasible and infeasible paths in Example 1.

According to Proposition 3(ii), the time window $[e_4, \ell_4] = [2, 5]$ can be reduced depending on the values $LB(\pi)$ and UB, i.e., depending on the integrality gap $UB - LB(\pi)$. Indeed, the values $t \mapsto \tilde{c}[T_4^{time}(\pi) > t]$ allow the following reductions:

- *if $UB - LB(\pi) < 1.5$, then the new time window is $[e_4, \ell_4] = [2, 4]$;*
- *if $UB - LB(\pi) < 1$, then the new time window is $[e_4, \ell_4] = [2, 3]$.*

No time-window reduction is possible with $t \mapsto \tilde{c}[T_4^{time}(\pi) < t]$.

Finally, the values $t \mapsto \tilde{c}[T_4^{time}(\pi) = t]$ are non-monotone for $t \in \mathbb{R}$. The interesting case is that of an integrality gap $UB - LB(\pi)$ that is smaller than 2. Optimal resource values of T_4^{time} must fall into the union of two non-overlapping intervals $[2, 3] \cup [4, 5]$. This could be implemented by replacing normal time windows by multiple time windows.

As visible in Example 1, the reduced cost functions and herewith the lower bound functions for the properties $[T_4^{time}(\pi) < t]$, $[T_4^{time}(\pi) > t]$, and $[T_4^{time}(\pi) = t]$ have several characteristics that are summarized in the following proposition. We will exploit these characteristics in the BPC algorithms.

Proposition 4. *The lower bound functions defined by (2) have the following properties.*

(i) $LB[T_i^{time} < t](\pi)$ is a non-increasing, piecewise constant function; it is continuous from the left;
(ii) $LB[T_i^{time} > t](\pi)$ is a non-decreasing, piecewise constant function; it is continuous from the right;
(iii) $LB[T_i^{time} = t](\pi)$ is a piecewise constant function; it is not necessarily monotone;
(iv) $LB[T_i^{time} = t](\pi) \geq \max\{LB[T_i^{time} \leq t](\pi), LB[T_i^{time} \geq t](\pi)\}$ holds true; strictly $>$ is possible.

Proof. The monotonicity and continuity (from one side) of $LB[T_i^{time} < t]$ and $LB[T_i^{time} > t]$ results from the fact that the set of paths fulfilling $[T_i^{time} < t]$ increases and fulfilling $[T_i^{time} > t]$ decreases with larger values of $t \in \mathbb{R}$. Moreover, the number of labels is finite. Hence, formulas (5a) and (5b) imply that also the number of different values assumed by the two functions is finite. This proves part (i) and (ii).

That $LB[T_i^{time} = t](\pi)$ is not necessarily monotone can be seen in Example 1, i.e., part (iii) is shown.

Since $[T_i^{time} = t]$ describes a subset of the paths described by $[T_i^{time} \leq t]$ and $[T_i^{time} \geq t]$, the inequality $\geq$ in part (iv) is straightforward. Note also that the difference between $LB[T_i^{time} < t](\pi)$ and $LB[T_i^{time} \leq t](\pi)$ as well as between $LB[T_i^{time} > t](\pi)$ and $LB[T_i^{time} \geq t](\pi)$ is just the values at the jump discontinuities. Example 1 therefore shows that $LB[T_i^{time} = t](\pi)$ is strictly greater than $\max\{LB[T_i^{time} \leq t](\pi), LB[T_i^{time} \geq t](\pi)\}$ on the open interval $(3, 4)$. This completes the proof. □

We will exploit these properties later in Section 2.5 when we consider the effective representation and storage of lower bound functions.

Arc-specific Properties. It is possible to combine the property $[ij]$ for an arbitrary arc $(i,j) \in A$ with properties that refer to the resource windows at the tail vertex i or the head vertex j. We refer to such properties as *arc-specific properties* in the following. We here present the arc-specific properties for the resource *time*.

Let us assume that an arc $(i,j) \in A$ is given. For the tail vertex i, property $[T_i^{time} = t, ij]$ refers to all paths that include arc (i,j) and allow a start of service at vertex i at time t followed by the traversal of arc (i,j). For the tail vertex j, property $[ij, T_j^{time} = t]$, describes all paths that traverse arc (i,j) and allow a start of service at time t at vertex j. The traversal must occur directly before the considered visit of vertex j. Obviously, analogue definitions are possible for $T_i^{time} < t$ and $T_i^{time} > t$.

To handle arc-specific properties, we must be able to efficiently compute the associated reduced cost. The following proposition provides a viable computation.

Proposition 5. *Let a dual feasible solution π be given. We further assume that a complete forward and backward labeling has been performed producing forward label sets $\mathcal{F}_i$ and backward label sets $\mathcal{B}_i$ for all vertices $i \in V$. Then, the reduced cost of the properties $[T_i^{time} < t, ij]$, $[T_i^{time} > t, ij]$, $[ij, T_j^{time} < t]$, and $[ij, T_j^{time} > t]$, can be computed as follows:*

$$\tilde{c}[T_i^{time} < t, ij](\pi) = \min_{\substack{F \in \mathcal{F}_i : F^{time} < t \\ B \in \mathcal{B}_j}} m(F, b_{ij}(B)) \tag{6a}$$

$$\tilde{c}[T_i^{time} > t, ij](\pi) = \min_{\substack{F \in \mathcal{F}_i, \\ B \in \mathcal{B}_j : b_{ij}^{time}(B) > t}} m(F, b_{ij}(B)) \tag{6b}$$

$$\tilde{c}[ij, T_j^{time} < t](\pi) = \min_{\substack{F \in \mathcal{F}_i : f_{ij}^{time}(F) < t \\ B \in \mathcal{B}_j}} m(f_{ij}(F), B) \tag{6c}$$

$$\tilde{c}[ij, T_j^{time} > t](\pi) = \min_{\substack{F \in \mathcal{F}_i, \\ B \in \mathcal{B}_j : B^{time} > t}} m(f_{ij}(F), B) \tag{6d}$$

Proof. Note the analogy to eq. (3) for reduced cost computation when arc (i,j) is included in a path. With this analogy, the proof of this proposition is similar to the one presented for Proposition 2. □

With Proposition 5 and the corresponding lower bound functions defined by eq. (2), it is already clear how to efficiently prove that path with properties $[T_i^{time} < t, ij]$, $[T_i^{time} > t, ij]$, $[ij, T_j^{time} < t]$, and $[ij, T_j^{time} > t]$ can be eliminated. This answer question (2) before question (1).

The open point is how the labeling process can be accelerated by considering the four properties, i.e., question (1). For this purpose, we suggest to replace the vertex-specific time windows $[e_i, \ell_i]$ for all $i \in V$ by arc-specific time windows. More precisely, we define for each arc (i,j) two time windows $[e_i^{ij}, \ell_i^{ij}]$ and $[e_j^{ij}, \ell_j^{ij}]$, where the first refers to the tail vertex i and the second to the head vertex j (the vertex is indicated by the lower index).

Initially, the new time windows are identical to the original time windows, i.e., $[e_i^{ij}, \ell_i^{ij}] = [e_i, \ell_i]$ and $[e_j^{ij}, \ell_j^{ij}] = [e_j, \ell_j]$ for all $(i,j) \in A$. We can now redefine the time-related parts of forward and backward REFs with the four arc-specific attributes $e_i^{ij}, e_j^{ij}, \ell_i^{ij}$, and ℓ_j^{ij} in the following way:

$$T_j^{time} = f_{ij}^{time}(T_i) = \max\{e_j^{ij}, T_i^{time} + \tau_{ij}\} \qquad \text{feasible if} \quad T_j^{time} \leq \ell_j^{ij} \tag{7a}$$

$$T_i^{time} = b_{ij}^{time}(T_j) = \min\{\ell_i^{ij}, T_j^{time} - \tau_{ij}\} \qquad \text{feasible if} \quad T_i^{time} \geq e_i^{ij} \tag{7b}$$

Eqs. (7) replace the standard time-window REFs (4). Such REFs are not new but have been mentioned in (Irnich and Desaulniers, 2005). Note that the computational effort for computing (7) is identical to the one for computing (4).

Proposition 6. *Let a dual feasible solution π be given. The following four rules are valid for all $t \in \mathbb{R}$:*
(i) If $LB[T_i^{time} < t, ij](\pi) > UB$, the value e_i^{ij} can be updated to $\max\{e_i^{ij}, t\}$.

(ii) If $LB[T_i^{time} > t, ij](\pi) > UB$, *the value* ℓ_i^{ij} *can be updated to* $\min\{t, \ell_i^{ij}\}$.
(iii) If $LB[ij, T_j^{time} < t](\pi) > UB$, *the value* e_j^{ij} *can be updated to* $\max\{e_j^{ij}, t\}$.
(iv) If $LB[ij, T_j^{time} > t](\pi) > UB$, *the value* ℓ_j^{ij} *can be updated to* $\min\{t, \ell_j^{ij}\}$.

Proof. This is a direct consequence of Propositions 1 and 5. □

2.2. Capacity Constraints

Capacity constraints are at the core of VRPs, e.g., the only constraints in the CVRP. They are defined by a vehicle capacity Q and non-negative demands d_j at all vertices $j \in V$. There is no demand at the origin and destination, i.e., $d_o = d_{o'} = 0$. The demand accumulated along the route must not exceed the capacity Q. Standard REFs for capacity constraints start with an initial load of 0 for the forward and backward labeling and propagate the load-attributes as follows:

$$T_j^{load} = f_{ij}^{load}(T_i) = T_i^{load} + d_j \qquad \text{feasible if} \quad T_j^{load} \le Q$$
$$T_i^{load} = b_{ij}^{load}(T_j) = T_j^{load} + d_i \qquad \text{feasible if} \quad T_i^{load} \le Q$$

These standard REFs have the clear disadvantage (for our purposes) that forward and backward load values do not share identical resource windows. This can be easily be corrected by (a) using the residual capacity for the backward labeling and (b) reducing the residual capacity by the demand d_j of vertex $j \in V$ along the backward propagation over an arc (i, j). With these assumptions, we can define the load-related resource $l\widetilde{oad}$ and the corresponding window $[d_j, Q]$ for each vertex $j \in V$. The left-hand-side (lower bound) of the window is the minimal accumulated load while the right-hand-side (upper bound) is the residual capacity at the vertex. Accordingly, we define the following properties for all $p \in \mathcal{P}$, vertices $j \in V$, and values $q \in \mathbb{R}$:

$[T_j^{l\widetilde{oad}} < q]$: The path can service/visit vertex j with a $l\widetilde{oad}$ value smaller than q; if j occurs several times in path p, at least one of the services must have accumulated $l\widetilde{oad}$ value strictly smaller than q;
$[T_j^{l\widetilde{oad}} > q]$: Likewise with a service/visit at vertex j with a $l\widetilde{oad}$ value strictly greater than q;
$[T_j^{l\widetilde{oad}} = q]$: Likewise with a service/visit at vertex j with a $l\widetilde{oad}$ value equal to q.

In analogy to the time-window case, we consider arc-specific properties $[T_i^{l\widetilde{oad}} < q, ij]$, $[T_i^{l\widetilde{oad}} > q, ij]$, $[ij, T_j^{l\widetilde{oad}} < q]$, and $[ij, T_j^{l\widetilde{oad}} > q]$ for propagating over an arc (i, j). Note that for these four properties *prop* the reduced cost $\tilde{c}[prop]$ can be computed exactly in the same way as described in Proposition 5 by replacing *time* with $l\widetilde{oad}$ (the upper index).

Even more, an analogue result as given in Proposition 6 can be transferred to the $l\widetilde{oad}$ resource case. To this end, we define arc-specific resource windows for the load-related resource. The left-hand-side and the right-hand-side of the windows are respectively the minimal accumulated load and the residual capacity at the *respective* vertex of the arc. Formally, the $l\widetilde{oad}$ resource windows at the tail vertex i and head vertex j are $[d_i^{ij}, Q_i^{ij}]$ and $[d_j^{ij}, Q_j^{ij}]$, respectively. They can be initialized with the values

$$[d_i^{ij}, Q_i^{ij}] := [d_i, Q - d_j] \qquad \text{and} \qquad [d_j^{ij}, Q_j^{ij}] := [d_i + d_j, Q].$$

Their propagation is performed in the following way:

$$T_j^{l\widetilde{oad}} = f_{ij}^{l\widetilde{oad}}(T_i) = \max\{d_j^{ij}, T_i^{l\widetilde{oad}} + d_j\} \qquad \text{feasible if} \quad T_j^{l\widetilde{oad}} \le Q_j^{ij} \tag{8a}$$
$$T_i^{l\widetilde{oad}} = b_{ij}^{l\widetilde{oad}}(T_j) = \min\{Q_i^{ij}, T_j^{l\widetilde{oad}} - d_j\} \qquad \text{feasible if} \quad T_i^{l\widetilde{oad}} \ge d_i^{ij} \tag{8b}$$

Note the similarity between (7) and (8). As a consequence, after replacing *time* with $l\widetilde{oad}$, the equivalent of Proposition 6 can be stated as follows:

Proposition 7. *Let a dual feasible solution* π *be given. The following four rules are valid for all* $q \in \mathbb{R}$*:*
(i) If $LB[T_i^{l\widetilde{oad}} < q, ij](\pi) > UB$, *the value* d_i^{ij} *can be updated to* $\max\{d_i^{ij}, q\}$.
(ii) If $LB[T_i^{l\widetilde{oad}} > q, ij](\pi) > UB$, *the value* Q_i^{ij} *can be updated to* $\min\{q, Q_i^{ij}\}$.
(iii) If $LB[ij, T_j^{l\widetilde{oad}} < q](\pi) > UB$, *the value* d_j^{ij} *can be updated to* $\max\{d_j^{ij}, q\}$.
(iv) If $LB[ij, T_j^{l\widetilde{oad}} > q](\pi) > UB$, *the value* Q_j^{ij} *can be updated to* $\min\{q, Q_j^{ij}\}$.

2.3. Possible Generalization

The previous subsections have shown that resource-window reduction for the resource time and the load-related resource are very similar, at least after a rather straightforward re-definition of attributes in the latter case. The fundamental requirement is that forward and backward propagation must use identical resource windows. Typically, the forward labeling provides lower bounds on the attribute values, while backward labeling provides upper bounds on the attribute values. This could also be reverse, in which case the roles of upper and lower bounds must be swapped for obtaining comparable results.

Up to now, we have only presented examples in which the resources were completely independent. An example of a interdependency between the time and the *state of charge* (SOC) occurs in the *electric VRPTW* (EVRPTW, Desaulniers *et al.*, 2016) with partial recharging, in which the arrival (forward) at the next vertex may have to be delayed because of an insufficient SOC requiring additional time for recharging. Time-window reduction with reduced cost is however directly applicable to the EVRPTW, since forward and backward labeling still consider the as-early-as-possible and the as-late-as-possible schedule, respectively. The generalization to the case of truly interdependent resources is also possible and we sketch two other examples.

In the VRP with simultaneous deliveries and pickups (Min, 1989), each customer $i \in V \setminus \{o, o'\}$ has to be visited exactly once. The service consists of a delivery of a non-negative quantity d_i as well as a pickup of a non-negative quantity p_i. At no point in time, the maximum load onboard the vehicle must exceed the given capacity Q. Forward labeling propagates two attributes, the accumulated quantity picked up (*pick*) on the partial path and the maximum load onboard (*ml*), at any point in the partial path. For the actual propagation along an arc $(i, j) \in A$, the *ml*-value depends on both *pick* and the *ml*-value before traversing the arc (i, j). Irnich (2008) has shown how to define backward REFs consistent with the forward labeling, i.e., forward and backward attributes share identical resource windows: The attributes in the backward case are the residual capacity with respect to the maximum load and the accumulated quantity delivered. Comparison of forward and backward attributes is asymmetric, i.e., accumulated pickup (forward) or delivery (backward) quantities are both compared against attributes that consider the (residual) maximum load onboard. Even if attribute propagation is interdependent, individual resource windows are given for the two attributes at all vertices $i \in V$. These resource windows can be reduced in the same way as suggested by Propositions 3, 6, and 7.

A second example is the computation of a route with minimum duration (travel time plus waiting time) in the time-window context. The minimum duration is relevant, e.g., to compute a duration-dependent salary for the driver. Tilk and Irnich (2016) have defined three attributes (in the forward labeling they represent the earliest time, the minimal duration, and latest possible start time at the origin) to model the *minimum tour duration problem* (MTDP). Moreover, they have defined resource windows as well as forward and backward REFs for the MTDP with the property that forward and backward attributes are mutual lower and upper bounds. Also here, resource-window reduction is applicable for all three resources: The earliest (forward) times and latest (backward) times can be handled like the time attributes of the VRPTW. As in the first example, the two other attributes have interdependent propagation rules but allow the application of resource-window reduction techniques.

2.4. Different Dual Prices

When solving a branch-and-bound node, the use of cutting planes in the BPC algorithm offers the possibility to iteratively apply resource-window reduction by reduced costs. We assume that $\ell \geq 1$ rounds of separation have been performed delivering dual feasible solutions $\pi_1, \pi_2, \ldots, \pi_\ell$. The associated lower bounds then fulfill $LB(\pi_1) \leq LB(\pi_2) \leq \cdots \leq LB(\pi_\ell)$, except for labeling algorithms using selective pricing, see (Desaulniers *et al.*, 2019).

Even for monotone dual bounds, the lower bound functions $LB[prop](\pi_1), LB[prop](\pi_2), \ldots, LB[prop](\pi_\ell)$ for a property $[prop]$ have no obvious relationship. For example, neither the relation $LB[T^{time} < t](\pi_k) \leq LB[T^{time} < t](\pi_j)$ nor $LB[T^{time} > t](\pi_k) \leq LB[T^{time} > t](\pi_j)$ holds in general for $1 \leq k < j \leq \ell$.

What seems like a poor characteristic at first glance, turns out to be a strong point for resource-window reduction. We can accumulate independent information from different dual solutions and use the maximum

$$LB^*[prop] = \max_{p=1,\dots,\ell} LB[prop](\pi_p). \tag{9}$$

The maximum $LB^*[prop]$ is again a monotone, piecewise constant lower bound function whenever the individual functions $LB[prop](\pi_p)$ have that characteristic for all $p \in \{1,\dots,\ell\}$. Instead of storing ℓ functions separately, only one function $LB^*[prop]$ per property *prop* can be stored and incrementally updated.

Storing $LB^*[prop]$ is useful, since the best known solution and upper bound UB may improve (become smaller) in the course of the BPC algorithm. In this case, $LB^*[prop]$ can be compared against the new UB to further reduce resource windows. This often happens at points much later than when the lower bound functions were computed.

2.5. Storing Lower Bound Functions

We next consider the effective representation and storage of the lower bound functions $LB^*[prop]$ for the properties like $[T_i^{time} < t]$, $[T_i^{time} > t]$, $[T_i^{l\widetilde{oa}d} < q]$, and $[T_i^{l\widetilde{oa}d} > q]$ as well as those for the respective arc-specific properties. As a maximum (see eq. (9)) of monotone functions (see Proposition 4(i) and (ii)) they are also monotone, which is not generally true for properties like $[T_i^{time} = t]$ and $[T_i^{l\widetilde{oa}d} = q]$.

The representation can be further simplified: Only values below $UB + 1$ are relevant. Furthermore, we typically have objective functions that guarantees integer objective values for all feasible solutions. For the comparison of $LB^*[prop]$ against UB, we can therefore replace arbitrary continuous values in $LB^*[prop]$ by values

$$LB^+[prop] = \begin{cases} UB + \varepsilon, & \text{if } LB^*[prop](\pi) > UB \\ \lceil LB^*[prop] \rceil - 1 + \varepsilon, & \text{otherwise} \end{cases} \tag{10}$$

where $\varepsilon > 0$ is a small constant (smaller than 1, close to zero). The functions $LB^+[prop]$ have not more than $UB - LB$ jump discontinuities, where $LB = \max_{p=1,\dots,\ell} LB(\pi_p)$. It means that they can be stored efficiently requiring not more space than proportional to the optimality gap $UB - LB$.

Example 2. *(cont'd from Example 1) To keep the example simple, we assume that only one round of fixing has been performed with the dual solution π. We have seen that inside the regular time window $[e_4, \ell_4] = [2, 5]$, the function $t \mapsto \tilde{c}[T_4^{time} > t](\pi)$ has three constant pieces with value 0 for $t \in (2, 3)$, value 1 for $t \in [3, 4)$, and value 1.5 for $t \in [4, 5)$. Note that the lower bound function $LB[T_4^{time} > t](\pi)$ is structurally identical, but shifted upwards by $LB(\pi)$.*

If, e.g., $LB = LB(\pi) = 10.2$, then $LB^+[T_4^{time} > t](\pi)$ has only two pieces. On the first piece for $t \in [2, 3)$, the value of $LB^+[T_4^{time} > t](\pi)$ is $\lceil 10.2+0 \rceil + \varepsilon - 1 = 10 + \varepsilon$. On the second piece for $t \in [3, 4) \cup [4, 5) = [3, 5)$, the value of $LB^+[T_4^{time} > t](\pi)$ is $\lceil 10.2 + 1 \rceil + \varepsilon - 1 = \lceil 10.2 + 1.5 \rceil + \varepsilon - 1 = 11 + \varepsilon$.

2.6. Interdependencies

Summarizing the results of the previous subsections, we see that resource-window reduction comes in two natures: We distinguish between *standard* resource windows (such as time windows $[e_i, \ell_i]$ and load-related windows $[d_i, Q]$) and *arc-specific* resource windows as used in the REFs (7) and (8). We now discuss the mutual interdependencies among the following components of the BPC algorithm:

(A) arc fixing
(B) resource-window reduction
(C) pricing subproblem (label extension, feasibility, dominance)
(D) master problem (RMP) and its dual solution π
(E) new best solution (upper bound UB)

Impact of Arc Fixing. Arc fixing (A) reduces of the arc set A, which has a direct impact on the pricing subproblem (C) as well as on the RMP (D). For pricing, a reduced arc set immediate lowers the computational effort in the label extension step. Moreover, it can be expected that less labels are generated so that the dominance algorithm must compare relatively fewer labels. For the RMP, any variable λ_p corresponding to a route p traversing any fixed arc can be eliminated from the RMP. This lowers the computational effort for the re-optimization of the RMP in every subsequent iteration. Note that at the time of running the arc-fixing algorithm, the λ_p variables that can be fixed are non-basic variables (assuming that an implementation of the simplex algorithm is used), but these variables may otherwise return into another basis at some later point in time of the solution process. Arc fixing can therefore produce tighter bounds so that there is also a (minor) impact on the branch-and-bound process.

Impact of Resource-Window Reduction. We consider resource-window reduction (B) in the standard and arc-specific case: In the standard case, a resource-window reduction alters the standard REFs as defined in (4) for time windows. Tighter feasibility constraints directly lead to fewer feasible labels generated in the labeling process (C). Moreover, tightened resource windows tend to produce better comparable labels. For example, two labels at a vertex i resulting from arrival times T_i^{time} and $T_i'^{time}$ become mutually comparable when the earliest service time e_i can be tightened to any value greater than or equal to $\max\{T_i^{time}, T_i'^{time}\}$. In addition, tightened resource windows allow the identification of additional *unreachable vertices* for a given label. In the elementary SPPRC, unreachable vertices allow to re-interpret the visiting counters and herewith to substantially improve the dominance between labels (Feillet *et al.*, 2004). All this reduces the number of newly generated labels and subsequent dominance comparisons, and makes the latter more effective. In addition, we can identify path variables λ_p in the RMP for which the path p is infeasible with respect to the tightened resource windows and eliminate these variables from the RMP (D).

In the arc-specific case, the same effects occur. In particular, in order to identify path variables λ_p in the RMP for which the path p is infeasible regarding the new REFs (7) and (8), we proceed as follows: The initial attribute vector at the origin vector o can be propagated with the REFs in forward direction along the arcs of the path p, always checking the feasibility conditions given in (7a) and (8a). If only one condition for one of the arcs $(i, j) \in A(p)$ and attributes fails, the path variables λ_p can be permanently eliminated from the RMP. The test can also be performed in backward direction with backward REFs (7b) and (8b), starting with the final attribute vector at the destination vertex o'.

Notably enough, in the arc-specific case, interesting additional effects can be observed. Note first that, by means of resource-window reduction, the resource windows $[e_i, \ell_i]$ and $[d_i, Q]$ of a vertex i can only become empty if the vertex is optional (examples are intermediate facilities such as replenishing points and recharging stations for electric vehicles). On the other hand, arc-specific resource window can always become empty so that the associated arc $(i, j) \in A$ can be eliminated. The latter effect is arc fixing (A) via resource-window reduction.

Impact of Master Problem and its Dual Solution. The solution of the master program (D) delivers the dual solution π and the lower bound $LB(\pi)$. A better lower bound $LB(\pi)$ gives a smaller optimality gap $UB - LB(\pi)$. Even if lower bound functions $LB[prop](\pi)$ are not systematically improving with increasing $LB(\pi)$, the trend is that smaller optimality gaps allow more fixations. This includes additional arc fixations and reductions of standard as well as arc-specific resource windows, i.e., (A) and (B).

Impact of New Best Solution. A new best solution (E) decreases the value UB, which in turn reduces the optimality gap, allowing again arc fixations and reductions of standard as well as arc-specific resource windows, i.e., (A) and (B).

3. Branch-Price-and-Cut Algorithm for the VRPTW

In this section, we briefly recall the definition of the VRPTW and sketch the algorithmic components of the BPC algorithm that we use for solving VRPTW instances. The implementation is almost identical

to the one used in (Desaulniers *et al.*, 2020), and we highlight the very few modifications in the following subsections.

The VRPTW can be defined over a directed graph $D = (V, A)$. The vertices V represent the customers N and two copies o and o' of the depot used to model the starting point and ending point of each route. Customers $i \in N$ are characterized by a demand d_i, a service duration s_i, and a time window $[e_i, \ell_i]$. The latter describes the possible service start times. For convenience, we define $d_o = d_{o'} = s_o = s_{o'} = 0$. The arcs $(i, j) \in A$ represent the possible travel connections consuming the travel time t_{ij} and inducing the routing cost c_{ij}. We assume that service durations and travel times are aggregated into a joint service and travel time $\tau_{ij} = t_{ij} + s_i$. The vehicle fleet is homogeneous and not restricted. All vehicles have the identical capacity Q.

A feasible route $(i_o, i_1, \ldots, i_p)$ is an elementary o-o'-path with $p \geq 2, i_0 = o$, and $i_p = o'$ satisfying the time-window and capacity constraints. The route fulfills the time-window constraints if there exists a schedule $(T_0^{time}, T_1^{time}, \ldots, T_p^{time}) \in \mathbb{R}^{p+1}$ with $T_j^{time} \in [e_{i_j}, \ell_{i_j}]$ for all $j \in \{0, 1, \ldots, p\}$ and $T_j^{time} + \tau_{i_j, i_{j+1}} \leq T_{j+1}^{time}$ for all $j \in \{0, 1, \ldots, p-1\}$. The route fulfills the capacity constraint if $\sum_{j=0}^{p} d_{i_j} \leq Q$.

The cost of a route is given by the sum $\sum_{j=0}^{p-1} c_{i_j, i_{j+1}}$. A set of feasible routes is a feasible solution to the VRPTW if all routes together visit and serve all customers $i \in N$ exactly once. A feasible solution is optimal if it minimizes the total cost of the routes.

3.1. Labeling

Forward and backward labeling for the elementary SPRRC of the VRPTW can be done using four types of attributes, i.e., T_i^{rdc} for the accumulated reduced cost of a partial path ending at vertex i, T_i^{time} for the time (as-early-as-possible or as-late-as-possible schedule at i, see Section 2.1), $T_i^{\widetilde{load}}$ for the load-related resource (accumulated load and residual capacity, see Section 2.2), and binary attributes $T^{visit,v}$ for all customers $v \in N$. The latter binary attributes $T_i^{visit,v}$ indicate whether customer v has already been visited on the partial path that ends at vertex i.

We use all well-established acceleration techniques. First, to improve the dominance, possibly additional unreachable vertices v that cannot be feasibly reached because of the *current* resource level $T_i^{time}/T_i^{\widetilde{load}}$ are included into $T_i^{visit,v}$ (Feillet *et al.*, 2004). Second, since solving the fully elementary version of the SPPRC is computationally difficult, we solve the *ng*-path relaxation Baldacci *et al.* (2011). Our implementation uses neighborhoods of size 10. As pointed out in Section 2, bidirectional labeling is indispensable for efficient arc fixing and resource-window reduction. Therefore, we solve the pricing subproblems with bidirectional labeling in the version with a dynamic half-way point (Tilk *et al.*, 2017). On average, dynamic half-way points further reduce the average computation time of bidirectional labeling, because they better balance the workload between forward and backward labeling and, thus, reduce the combinatorial explosion. Furthermore, we rely on a bucket-based implementation using a one-dimensional bucketing on the *time* resource a suggested in (Sadykov *et al.*, 2021). Following ideas in (Desaulniers *et al.*, 2008), we use pricing heuristics a.k.a. partial pricing. To this end, arc-reduced networks are constructed having a minimum of $k = 2, 5, 10$, and 15 arcs, respectively, that enter and exit each customer node.

3.2. Cutting and Branching

The linear relaxation of the master program can be strengthen with robust and non-robust cuts, where the latter require the addition of further attributes to the attributes (see Section 3.1) because they cannot be implemented by only modifying the reduced cost of arcs. In order to better assess the impacts of arc fixing and resource-window reduction, we considered just one family of non-robust cuts. In particular, we use *limited-memory subset-row inequalities* (lm-SRIs, Pecin *et al.*, 2017a) for subsets S of rows with $|S| = 3$. In this case, one binary attribute $T_i^{SRI,S}$ must be added for each active subset-row inequality (SRI) to the subproblem. Compared to the original SRI of Jepsen *et al.* (2008), lm-SRIs are computationally less costly because they reduce the number of cases in which a cost penalty must be added to a potentially dominating label in the SPPRC dominance algorithm. Violated limited-memory subset-row inequalities are computed and inserted only at the root node of the branch-and-bound tree. We consider inequalities as violated when the violation exceeds 0.05.

Branching in a BCP algorithm is required to finally ensure integer solutions. We explore the branch-and-bound search tree with a best-bound first node selection strategy to foster small search trees and high-quality lower bounds. As in many other implementations, we apply a two-level branching scheme in which the first level considers the number of vehicles in use and the second level the flow over arcs. In both levels, we always select a branching variable whose fractional value is the closest to 0.5. Two child nodes are created by bounding the branching variable from above (below) by the rounded-up (rounded-down) value. Branching on the number of vehicles is implemented by adding a linear constraint to the restricted master program, while branching on an arc is directly enforced by forbidding and removing one or several arcs from the underlying digraph over which the pricing problem is solved. Moreover, routes that use forbidden arcs are removed from the RMP.

3.3. Arc Fixing and Resource-Window Reduction

In principle, arc fixing and resource-window reduction could be applied after every round of solving the linear relaxation of the master program and at every branch-and-bound node. However, a detailed bookkeeping of the lower bound functions would be required after branching, because all lower bounds are only valid for the respective subtree. To keep the computational setting simple, we only compute restricted arc sets and the lower bound functions at the root node of the branch-and-bound tree. Several rounds of adding violated valid inequalities (see Section 3.2) allow us to compute the refined functions $LB^+[prop]$ as defined by eq. (10).

We now give some more details about the arc fixing and resource-window reduction computation in a round, i.e., for a given feasible dual price π. As can be seen by comparing eq. (3) with eqs. (6) (and likewise for other resources), the computational effort for arc fixing is lower than that for arc-specific resource-window reduction. Hence, we always employ arc fixing before resource-window reduction in settings that use both techniques. When some arcs are already eliminated, the computational effort for arc-specific resource-window reduction becomes smaller.

Moreover, the computation of all undominated forward and backward labels can be accelerated by using the labels from the bidirectional labeling with a half-way point. Indeed, we warm-start the complete labeling with the undominated labels that have not been extended because of the halfway condition. Note further that forward labels can be used to bound backward labels and vice versa. Therefore, we try to estimate in which direction the complete labeling with warm start is simpler. The dynamic HWP gives an indication (we use *time* as the monotone resource): if it is right (left) to the middle $(e_o + \ell_{o'})/2$ of the time horizon, forward (backward) labeling is expected to be simpler. The labeling in the simpler direction is performed before the other direction that can afterwards benefit from bounding.

4. Computational Results

All algorithms are implemented in `C++` using the callable library of `CPLEX` 12.7.0 and compiled into 64-bit single-thread release code with Microsoft Visual Studio 2017. The computational study is performed on a 64-bit Microsoft Windows 10 computer with an Intel® Core™ i7-6700K clocked at 4.00 GHz and with 32 GB of RAM. In all calls to `CPLEX`, default values of all parameters are kept except for setting the number of available threads to one. Moreover, the restricted master programs are re-optimized with the primal simplex algorithm of `CPLEX`.

We tested the algorithms on the Solomon's benchmark set which consists of 56 instances with 100 customers (see `http://w.cba.neu.edu/~msolomon/problems.htm`). The instances are classified according to the customer distribution as random (R), clustered (C), and mixed (RC). Moreover, Series 1 with the subsets R1, C1, and RC1 has tight, and Series 2 with the subsets R2, C2, and RC2 has loose constraints. By considering only the first 25 or 50 customers, one gets another two subsets of 56 smaller instances each. For all 168 instances, optimal solutions are known (see the online supplement of He *et al.*, 2019).

We used the established preprocessing rules that are described in (Desrochers *et al.*, 1992) to reduce the resource windows for $\widetilde{load}$ and *time* .

Our BPC algorithm with arc fixing solves all but two instances (R208.100 and R211.100) to optimality within the time limit of 7200 seconds. The linear relaxation of the master program already provides integer

optimal solution for 86 instances. We do not consider these instances for analyzing the impact of arc fixing and resource-window reduction, as the result would otherwise be biased. We restrict our analysis to the remaining 80 VRPTW instances that were also selected in (Desaulniers *et al.*, 2020).

4.1. Assessment of Resource-Window Reduction

Arc fixing and resource-window reduction requires the computation of an upper bound UB. We use the same type of initialization as several other works (Pecin *et al.*, 2017a,b; Pessoa *et al.*, 2018; Desaulniers *et al.*, 2020): Assuming that high-quality metaheuristics can compute an optimal solution (with objective value *opt*) quickly, the upper bound can be initially set to $UB = opt + 1$. The task of the BPC algorithm is then to systematically improve the tree lower bound LB so that it reaches *opt* to actually find at least one optimal solution in the tree.

4.1.1. Reduction of the Standard Resource Windows

We first compare different computational *settings* of the BPC algorithm that apply a reduction of the standard resource windows but differ in two aspects. On the one hand, the arc fixing is either performed before every round of resource-window reduction or not performed at all. On the other hand, we consider the attributes for $l\widetilde{oa}d$ and *time* either separately or in combination. The following seven computational settings are analyzed:

AF: arc fixing, no resource-window reduction; this is the baseline setting;
AF/LD: arc fixing and resource-window reduction for the $l\widetilde{oa}d$-attribute;
AF/TW: arc fixing and time-window reduction, i.e., for the *time*-attribute;
AF/LD.TW: arc fixing and resource-window reduction for the attributes $l\widetilde{oa}d$ and *time*;
LD: w/o arc fixing, but with resource-window reduction for the $l\widetilde{oa}d$-attribute;
TW: w/o arc fixing, but with time-window reduction, i.e., for the *time*-attribute;
LD.TW: w/o arc fixing, but with resource-window reduction for the attributes $l\widetilde{oa}d$ and *time*;

For the seven computational setting $X \in \{\text{AF}, \text{AF/LD}, \text{AF/TW}, \text{AF/LD.TW}, \text{LD}, \text{TW}, \text{LD.TW}\}$, the following indicators are computed:

Geo. mean time ratios: Geometric mean of the ratios t_X/t_{AF} of overall computation times t_X relative to the baseline setting AF consuming t_{AF} time;
#Opt: Number of instances solved to proven optimality with setting X;
#Unsolved: Number of unsolved instances with setting X.

Moreover, all computational settings, except for AF, reduce the standard resource windows for either one of the resources $l\widetilde{oa}d$ and *time* or both. We can quantify the reduction (in percent) by computing the value $100 \cdot 1/|N| \sum_{i \in N} (u_i^{end} - l_i^{end})/(u_i^{begin} - l_i^{begin})$, where u_i^{begin} and l_i^{begin} are the bounds of the resource window $[u_i^{begin}, l_i^{begin}]$ of vertex i at the beginning (after preprocessing), and u_i^{end} and l_i^{end} the respective values at the end. Here, we compute the minimum (min.), the average (avg.), and the maximum value (max.) over all 80 instances.

Table 2 summarizes the computational tests. The BPC algorithms that apply arc fixing (group of settings AF, AF/LD, AF/TW, and AF/LD.TW) are very much superior to those that solely rely on resource-window reduction (group of settings LD, TW, and LD.TW). The former solve 77 or 78 (of 80) instances to proven optimality, while the latter solve 72 to 75 instances only. Even more significant is the relative speed compared to the BPC with setting AF reflected by ratios of computation times: here, the sole use of resource-window reduction (without arc fixing) slows down the solution process by more than factor 2. Settings inside each group give rise to BPC algorithms that are rather homogeneous, but the BPC algorithms arising for the two groups are very different.

The reduction of standard resource windows is able to reduce the resource windows by between 12.8 and 14.7 percent on average (over the 80 instances). Differences between the resource reductions for $l\widetilde{oa}d$ and *time* are not large as the differences between performance indicators. The most important insight is that even if reductions are noticeable, they do not make the corresponding BPC algorithm faster. The additional computational effort required to actually compute the reduced resource windows using Proposition 3 is larger than what is finally saved by means of the smaller resource windows. In particular the indicators for the

		Computational Settings						
		AF	AF/LD	AF/TW	AF/LD.TW	LD	TW	LD.TW
Geo. mean time ratios		1.00	1.02	1.04	1.05	2.33	2.33	2.11
#Opt		78	78	77	78	72	73	75
#Unsolved		2	2	3	2	8	7	5
$\widetilde{load}$-window reduction (%)	min.		2.00		2.00	2.00		2.00
	avg.		12.80		12.91	13.10		13.06
	max.		33.92		35.74	38.70		36.40
time-window reduction (%)	min.			0.00	0.08		0.08	1.12
	avg.			14.50	14.67		14.26	14.20
	max.			30.25	32.04		31.80	32.57

Table 2: Comparison of computational settings using reduction technique for the standard resource windows.

computational settings that reduce resource window and use arc fixing (AF/LD, AF/TW, and AF/LD.TW) show that, on average, the additional effort translates into at least a 2-percent increase in the computation time.

4.1.2. Arc-Specific Resource-Window Reduction

We now analyze the arc-specific resource-window reduction techniques. Again, seven different computational settings of the BPC algorithm as named in the previous section are considered. Compared to Section 4.1.1 the meaning of AF (for arc fixing) is slightly different. On the one hand (settings with AF), the arc fixing is *explicitly* performed, always before arc-specific resource-window reductions are computed. On the other hand (settings without AF), no explicit arc fixing procedure is invoked. However, if an arc-specific resource window becomes the empty interval, the associated arc is eliminated. This is an *implicit* arc fixing.

The comparison between explicit and implicit arc fixing (AF/LD versus LD, AF/TW versus TW, and AF/LD.TW and LD.TW) can be quickly summarized: Differences in the number of arcs fixed as well as computation times are very marginal. There is a slight trend (less than 1 percent) in favour of the versions that do only implicit arc fixing (results are however not fully consistent). As a result, only four computational settings remain, i.e., AF that we use as the baseline setting (no resource-window reduction) and settings LD, TW, and LD.TW, which implicitly perform arc fixing on the basis of those arc-specific resource windows that become empty.

Overall results with these four computational setting are diametrically opposite to those obtained with the reduction of standard resource windows. For the 80 relevant instances described above, the geometric mean of the computation time ratios t_X/t_{AF} with $X \in \{\text{LD, TW, LD.TW}\}$ are 0.997, 0.911, and 0.896, respectively. The reduction of the load-related resource windows alone (setting LD) is comparable with the baseline setting, while reduction of time windows and the resource-window reduction with both $\widetilde{load}$ and *time* lead to an average decrease in time of approximately 10 percent (slightly less for TW and slightly more for LD.TW).

The savings in computation time resulting from arc-specific resource-window reduction are not even spread over the 80 instances of the testbed. Savings tend to become larger for instances that are more difficult to solve. In Figure 2, we detail this finding: We filter the ratios by minimum computation time $rt > 0$ and consider the subset of instances for which the BPC algorithm with the baseline setting requires at least $t_{AF} \geq rt$ seconds for the computation. For example, there are $n = 53$ instances running $rt = 1$ second or longer with the baseline BPC algorithm. The second group of bar charts in Figure 2 refers to this subset of instances. We see that the computational settings TW and LD.TW lower the computation time by almost/more than 15 percent, respectively.

Figure 2 visualizes the ratios for increasing minimum computation times rt between 0 and 2000 seconds (in twelve groups). Clearly, $\widetilde{load}$ resource-window reduction alone (setting LD) is not effective for Solomon's VRPTW instances, since the ratio never falls below 0.9, corresponding to an average 10-percent smaller computation time. On the positive side, it shows that time-window reduction (setting TW), and even more

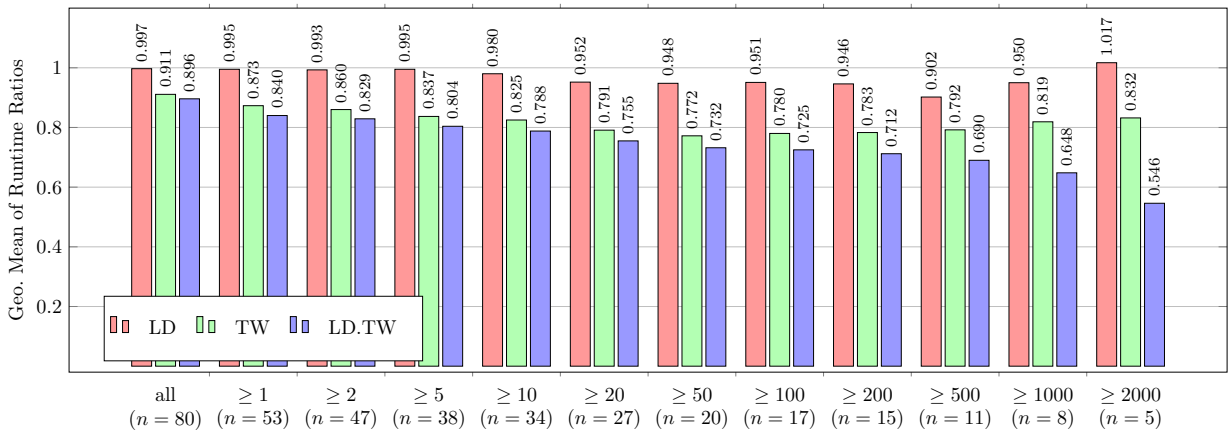


Figure 2: Geometric mean of the computation time ratios w.r.t. to the baseline setting AF (arc fixing) for the resource-window reduction settings LD, TW, and LD.TW; results are grouped according to computation times where $\geq rt$ indicates that only instances with a computation time t_{AF} of at least rt are considered.

the combined *time* and $l\widetilde{oa}d$ resource-window reduction (setting LD.TW), tend to become more and more effective with larger values of rt (however there is a small increase of the ratios for setting TW starting from $rt \geq 100$). In the combined setting LD.TW, an on average 25-percent smaller computation time is gained for all groups of instances such that $rt > 20$ seconds. For the last group of instances requiring more than 2000 seconds, the computation time with setting $t_{LD.TW}$ is almost halved compared to that related to the baseline setting. This is a substantial saving.

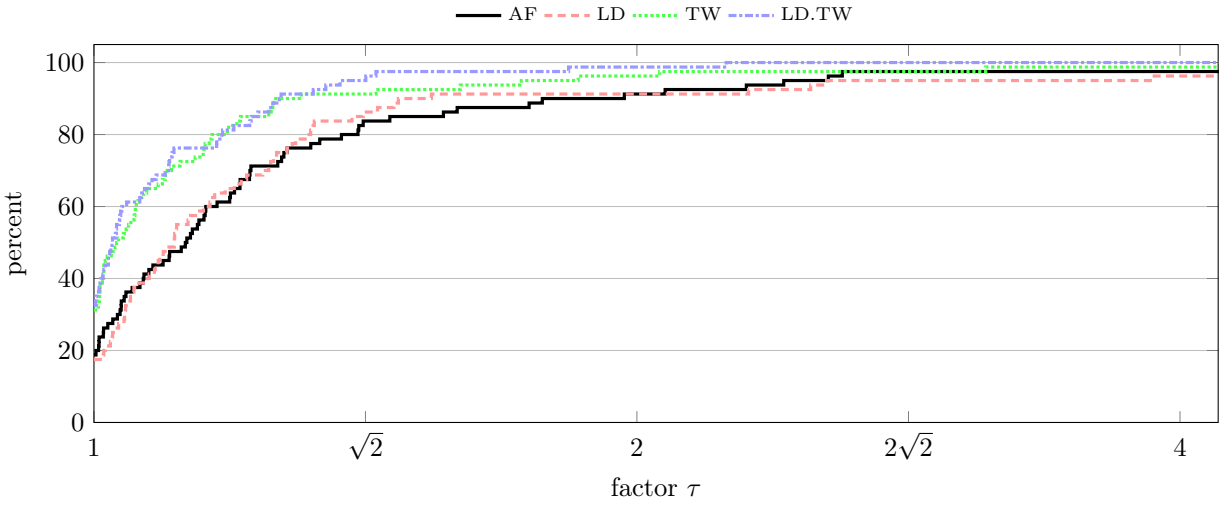


Figure 3: Performance profiles $\rho_X(\tau)$ for the four computational settings $X \in \{\text{AF, LD, TW, LD.TW}\}$. Note that the logarithmic scale on the axis of abscissae (τ values) is considered.

Finally, we compare the four computational settings with the help of performance profiles (Dolan and Moré, 2002). For a set of algorithms $\mathcal{X}$ (those related to the four computational settings in our case), the performance profile $\rho_X(\tau)$ of an algorithm $X \in \mathcal{X}$ describes the ratio of instances that can be solved by X within a factor τ compared to the fastest algorithm, i.e.

$$\rho_X(\tau) = \left|\left\{I \in \mathcal{I} : t_I^X / t_I^* \leq \tau\right\}\right| / |\mathcal{I}|$$

in which $\mathcal{I}$ is the set of instances (the 80 Solomon instances in our case), t_I^X is the computation time of algorithm X when applied to instance $I \in \mathcal{I}$, and t_I^* is the smallest computation time among all algorithms of set $\mathcal{X}$ for instance I. Unsolved instances are taken into account with $t_I^X = \infty$ (assuming that ∞/∞ gives ∞). In particular, $\rho_X(1)$ is the percentage of instances for which X is the fastest algorithm, and $\rho_X(\tau)$ for large values of τ is the percentage of instances that are solved by X within the time limit.

The performance profiles in Figure 3 confirm the findings stated above: The computational setting LD gives comparable results w.r.t. the baseline setting AF. The two related BPC algorithms are clearly outperformed by those considering settings TW and LD.TW. The BPC algorithm based on setting LD.TW is the only one that solves all 80 instances within the 2-hour time limit, and it is visibly also the best-performing algorithm.

5. Conclusions

In this work, we have introduced a new resource-window reduction mechanism for the acceleration of labeling algorithms solving the shortest-path problem with resource constraints (SPPRC) in branch-price-and-cut (BPC) algorithms for vehicle-routing and crew-scheduling problems. For example, resource-window reduction leads to tightened time windows and load-related resource windows. Our new resource-window reduction algorithm relies on established variable-fixing techniques (Irnich *et al.*, 2010) that exploit the minimum reduced cost of ad-hoc subsets of paths. In our case, each subset comprises all the paths with the property that their resource consumption, at some given vertex, is restricted to certain values. We have shown how to compute these minimum reduced costs efficiently once all forward and backward labels are available.

Resource-window reduction impacts several components of a BPC algorithm. First and foremost, the solution of pricing problems is potentially accelerated because more restricted values of the resources lead to more labels that can dominate each other in the SPPRC labeling process. More dominations means fewer labels which, in turn, reduces the effort for subsequent dominance and extension steps. Since pricing and, in particular, the iterative execution of dominance algorithms within pricing consume a large share of the total computation time, the entire BPC algorithm benefits. Second, tightened resource windows also allow the elimination of path variables from the restricted master program leading to slightly faster reoptimization times. Third and finally, slightly improved bounds can result from tightened resource windows, as already discussed and analyzed for other variable-fixing techniques (Irnich *et al.*, 2010; Desaulniers *et al.*, 2020).

The vehicle-routing problem with time windows (VRPTW) is an intensively studied and prototypical VRP giving rise to resources for time and load. Undoubtedly, BPC algorithms for the VRPTW constitute the best-performing exact solution approaches (Costa *et al.*, 2019). Our computational experiments with a state-of-art BPC algorithm on benchmark instances of the VRPTW have provided the following findings: A straightforward reduction of the standard resource windows can indeed tighten resource windows, but this reduction does not translate into savings in computation times. Reversely, the additional computational effort for reduction even increases the total computation time compared to those of a BPC algorithm that uses arc fixing only.

However, resource-window reduction by reduced costs can result in significant overall speedups. The key finding here is that resource-window reduction should be performed as an arc-specific resource-window reduction. We have introduced arc-specific resource windows and modified resource extension functions (REFs) for this purpose. For each resource and each arc (i, j), there are two arc-specific resource windows, one referring to the tail vertex i and one to the head vertex j. We have shown that these arc-specific resource windows can be reduced with the same techniques as the standard resource windows. Arc-specific reduction does not have a negative impact on the labeling algorithm because the modified REFs are computationally as costly as the standard REFs and the dominance test is not affected at all.

Our computational experiments on the VRPTW show that arc-specific resource-window reduction is much more effective than reduction of standard resource windows. Compared to the baseline setting, arc-specific reduction on time windows works better than on the load-related resource windows. More importantly, the combination of arc-specific reductions for the time and the load-related resources outperforms

reductions considering a single resource, and the speedup generally increases with the difficulty of the problem instance. One can observe average speedups by a factor of almost two for instances requiring more than 2000 seconds of computation time using the BPC algorithm with the baseline setting.

The application of resource-window reduction techniques is not limited to the VRPTW. We have developed the theoretical foundations of resource-window reduction so that it can easily be applied to many other types of resource constraints. What is needed is just a resource for which forward and backward labeling rely on identical resource windows. For example, this is fulfilled for the earliest and latest possible service start times used in the case of time windows. We presented a straightforward reformulation of the backward load-related resource that leads to identical resource windows for both directions also in this case. Similar reformulations are known for other constraints such as for simultaneous delivery and pickup, tour duration (Irnich, 2008), and for truck-driver scheduling (Tilk and Goel, 2020).

In all cases, the implementation effort of adding resource-window reduction techniques to an existing BPC algorithm is minor: Implement the actual reduction procedures, replace standard REFs with new REFs using arc-specific resource windows, and add components that enforce the tightened resource windows at the master and pricing problem levels. The computational experiments also revealed that a dedicated arc-fixing algorithm is longer needed because arc-specific resource-window reduction implicitly performs arc elimination once the resource window for at least one resource becomes empty.

Acknowledgement

This research was supported by the Deutsche Forschungsgemeinschaft (DFG) under grants GS 83/1-1 and IR 122/10-1. This support is gratefully acknowledged.

References

Altman, C., Desaulniers, G., and Errico, F. (2021). The fragility-constrained vehicle routing problem with time windows. arXiv arXiv:2109.01883.

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.

Costa, L., Contardo, C., and Desaulniers, G. (2019). Exact branch-price-and-cut algorithms for vehicle routing. *Transportation Science*, **53**(4), 946–985.

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

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. **42**(3), 387–404.

Desaulniers, G., Errico, F., Irnich, S., and Schneider, M. (2016). Exact algorithms for electric vehicle-routing problems with time windows. *Operations Research*, **64**(6), 1388–1405.

Desaulniers, G., Pecin, D., and Contardo, C. (2019). Selective pricing in branch-price-and-cut algorithms for vehicle routing. *EURO Journal on Transportation and Logistics*, **8**(2), 147–168.

Desaulniers, G., Gschwind, T., and Irnich, S. (2020). Variable fixing for two-arc sequences in branch-price-and-cut algorithms on path-based models. *Transportation Science*, **54**, 1170–1188.

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.

Dolan, E. D. and Moré, J. J. (2002). Benchmarking optimization software with performance profiles. *Mathematical Programming*, **91**(2), 201–213.

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.

He, Q., Irnich, S., and Song, Y. (2019). Branch-cut-and-price for the vehicle routing problem with time windows and convex node costs. *Transportation Science*, **53**(3), 1409–1426.

Huisman, D., Freling, R., and Wagelmans, A. P. M. (2005). Multiple-depot integrated vehicle and crew scheduling. *Transportation Science*, **39**(4), 491–502.

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.

Irnich, S., Desaulniers, G., Desrosiers, J., and Hadjar, A. (2010). Path-reduced costs for eliminating arcs in routing and scheduling. *INFORMS Journal on Computing*, **22**(2), 297–313.

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

Min, H. (1989). The multiple vehicle routing problem with simultaneous delivery and pick-up points. *Transportation Research*, **23**, 377–386.

Nemhauser, G. and Wolsey, L. (1988). *Integer and Combinatorial Optimization*. John Wiley & Sons, Inc.

Pecin, D., Pessoa, A., Poggi, M., and Uchoa, E. (2017a). Improved branch-cut-and-price for capacitated vehicle routing. *Mathematical Programming Computation*, **9**(1), 61–100.

Pecin, D., Contardo, C., Desaulniers, G., and Uchoa, E. (2017b). New enhancements for the exact solution of the vehicle routing problem with time windows. *INFORMS Journal on Computing*, **29**(3), 489–502.

Pessoa, A., Sadykov, R., Uchoa, E., and Vanderbeck, F. (2018). Automation and combination of linear-programming based stabilization techniques in column generation. *INFORMS Journal on Computing*, **30**(2), 339–360.

Pessoa, A., Sadykov, R., Uchoa, E., and Vanderbeck, F. (2020). A generic exact solver for vehicle routing and related problems. *Mathematical Programming*, **183**(1-2), 483–523.

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

Sadykov, R., Uchoa, E., and Pessoa, A. (2021). A bucket graph–based labeling algorithm with application to vehicle routing. *Transportation Science*, **55**(1), 4–28.

Tilk, C. and Goel, A. (2020). Bidirectional labeling for solving vehicle routing and truck driver scheduling problems. *European Journal of Operational Research*, **283**(1), 108–124.

Tilk, C. and Irnich, S. (2016). Dynamic programming for the minimum tour duration problem. *Transportation Science*, **51**(2), 549–565.

Tilk, C., Rothenbächer, A.-K., Gschwind, T., and Irnich, S. (2017). Asymmetry matters: Dynamic half-way points in bidirectional labeling for solving shortest path problems with resource constraints faster. *European Journal of Operational Research*, **261**(2), 530–539.

Toth, P. and Vigo, D., editors (2014). *Vehicle Routing: Problems, Methods, and Applications*. Society for Industrial and Applied Mathematics, Philadelphia, PA.

Villeneuve, D. and Desaulniers, G. (2005). The shortest path problem with forbidden paths. *European Journal of Operational Research*, **165**(1), 97–107.