A Bucket Graph Based Labelling Algorithm for Vehicle Routing

Ruslan Sadykov, Artur Pessoa, Eduardo Uchoa

A Bucket Graph Based Labeling Algorithm with Application to Vehicle Routing

Ruslan Sadykov[1], Eduardo Uchoa[2], and Artur Pessoa[2]

[1]INRIA Bordeaux – Sud-Ouest 200 Avenue de la Veille Tour, 33405 Talence, France

[2]Universidade Federal Fluminense - Engenharia de Produção , Rua Passo da Pátria 156, Niterói - RJ - Brasil - 24210-240,

Abstract

We consider the Shortest Path Problem with Resource Constraints (SPPRC) arising as a subproblem in state-of-the-art Branch-Cut-and-Price algorithms for vehicle routing problems. We propose a variant of the bi-directional label correcting algorithm in which the labels are stored and extended according to the so-called bucket graph. Such organization of labels helps to decrease significantly the number of dominance checks and the running time of the algorithm. We also show how the forward/backward route symmetry can be exploited and how to eliminate arcs from the bucket graph using reduced costs. The proposed algorithm can be especially beneficial for vehicle routing instances with large vehicle capacity and/or with time window constraints. Computational experiments were performed on instances from the distance constrained vehicle routing problem, including multi-depot and site-dependent variants, on the vehicle routing problem with time windows, and on the "nightmare" instances of the heterogeneous fleet vehicle routing problem. Significant improvements over the best algorithms in the literature were achieved and many instances could be solved for the first time.

1 Introduction

The best known exact approach for many classical variants of the Vehicle Routing Problem (VRP) is a Branch-Cut-and-Price (BCP) algorithm in which the master problem contains route variables, customer visiting constraints and additional cuts. Because of their exponential number, the route variables are generated dynamically by solving pricing subproblems modeled as Shortest Path Problems with Resource Constraints (SPPRC). In those problems, one looks for least cost paths joining a source vertex with a sink vertex such that the accumulated consumption of resources along the path respects given lower and upper limits. Dynamic Programming labeling algorithms are the most usual way of solving those problems.

Labeling algorithms differ from the traditional "table filling" dynamic programming algorithms because they only store (as *labels*) the reachable states, those representing feasible partial paths. More importantly, while traditional dynamic programming only performs dominance over identical states, labeling algorithms perform dominance between labels corresponding to non-identical states. Typically, a label L' representing a path $P^{L'}$ is dominated, and therefore eliminated, if there is another label L representing a path P^L that ends in the same vertex and is not more costly and does not use more resources than $P^{L'}$. Mono-directional labeling algorithms start from a single label representing a null path at the source vertex. At each step, a non-extended label L is extended to additional labels corresponding to the possible ways of adding a single arc to P^L. Dominance may be used to eliminate labels, avoiding future extensions. If there are no non-extended labels, the algorithm stops and optimal solutions are found among the labels representing paths ending in the sink vertex. The general labeling algorithm

has several degrees of freedom. In particular, there are many possible ways of choosing the next label to be extended and how often and how extensively dominance checks are performed. Particular labeling algorithms are classified as being either label-setting or label-correcting. The defining property of a label-setting algorithm is that it only extends labels that can never be dominated by another label created after that extension.

There is a large literature on labeling algorithms for SPPRC, we refer to Irnich and Desaulniers (2005) and Pugliese and Guerriero (2013) for surveys. However, somehow surprisingly, not so many papers discuss in depth issues related to label organization and dominance strategies. In fact, even though the pricing time is a bottleneck in many BCP algorithms for VRP, authors often skip the details of the labeling algorithm used. An exception is the recent algorithm for the Capacitated VRP (CVRP) in Pecin et al. (2017b). In that case, assuming that the consumptions of the single resource (capacity) are given by positive integer numbers, labels are organized in buckets corresponding to each possible consumption. Dominance is only performed for labels in the same bucket, that by definition have the same resource consumption. However, that approach cannot be applied in cases where resource consumptions are given by arbitrary real numbers. In fact, if the consumptions are given by larger integer numbers the algorithm becomes slow, since few dominance checks are performed and too many undetected dominated labels are expanded. The opposite approach is to perform full label dominance after each extension. In this case, the running time may suffer a lot from too many dominance checks with negative results.

Lozano and Medaglia (2013) proposed the so-called Pulse Algorithm for the SPPRC. Pulse is better viewed as a depth-first branch-and-bound search algorithm than as a dynamic programming labeling algorithm, because it uses dominance in a severely limited way — each vertex only keeps a handful of partial paths for performing dominance checks. The main mechanism in Pulse for trying to avoid an exponential explosion in its search is bounding, a partial path is pruned if it can be shown that it cannot be extended into a full path ending at the sink vertex that costs less than the currently best known source to sink path. Pulse produced excellent results, much better than label-setting algorithms, on some stand-alone SPPRCs where all arc costs are positive. The last feature helps Pulse because it allows good bounds to be computed by Dijkstra's-like algorithms. Up to now, as far as we know, Pulse was only tested as a subproblem solver in a column generation algorithm for VRP with Time Windows (VRPTW) in Lozano, Duque, and Medaglia (2015). The SPPRCs that appear in that context are more complex because the master dual variables make arc costs to be possibly negative. The test was calculating the elementary route bound on instances with 100 customers. The variant of Pulse used for the VRPTW only performs an even more basic form of dominance called rollback pruning. However, it proposes a more sophisticated bounding mechanism that only considers the time resource. The obtained results compare favourably with the labeling algorithms of Desaulniers, Lessard, and Hadjar (2008) and Baldacci, Mingozzi, and Roberti (2011). It is not known how Pulse would perform on larger instances and how it would handle the modifications in the SPPRC induced by cuts, essential for modern BCP algorithms for VRP.

The main original contributions of this paper are the following:

- In Section 3, a new variant of the labeling algorithm is proposed. The approach relies on a so-called bucket graph, consisting of buckets and bucket arcs. Labels for paths ending in the same vertex of the original graph and having similar resource consumption are grouped together in buckets. A bucket arc links two buckets if a label in the first one can be possibly extended to a label in the second. Labels within the same bucket are always pairwise undominated, i.e. there is no label which is dominated by another label in the same bucket. However, dominance checks between labels in different buckets are only performed before an extension, when it is checked whether a candidate label to be extended is dominated or not. Moreover, this inter-bucket dominance uses bounds on the costs of all labels in a bucket, avoiding many unnecessary checks. The key parameter of the algorithm is the *step size*, used for determining the resource consumption intervals that define each bucket. If the step size is sufficiently small, the bucket graph is acyclic and the algorithm becomes a label-setting algorithm. In general, the bucket graph has cycles and

the algorithm is a label-correcting algorithm. A bi-directional version of the bucket graph based labeling algorithm is also proposed (since Righini and Salani (2006) it is known that bi-directional labeling algorithms are often more efficient than their mono-directional counterparts). The concatenation of labels step is also accelerated by bounds on the costs. Bi-directional search also allows us to exploit the forward/backward symmetry of routes that exists on some VRP variants, for example, on the classical Capacitated VRP (CVRP). On those cases, the backward search is replaced by a "reversed copy" of the forward search, reducing the running time.

- In Section 4, a procedure for accelerating the labeling algorithm by removing arcs from the bucket graph using reduced cost arguments is proposed. We call it *bucket arc elimination procedure*. The concept of *jump bucket arcs* is introduced. The new fixing procedure generalizes both the procedures in Irnich et al. (2010) and in Pessoa et al. (2010). After a fixing, some bucket arcs are eliminated, and thus the number of extensions and the running times are reduced in future calls to the labeling algorithm.

The new labeling algorithm was embedded as the pricing method in a modern BCP algorithm for VRPs. Our BCP also includes many ingredients found in other state-of-the-art BCP algorithms, like dynamic $ng-$path relaxation, rounded capacity cuts, limited arc memory rank-1 cuts, automatic dual price smoothing stabilization, a procedure for enumerating elementary paths, and multi-phase strong branching with pseudo-costs. The algorithm was computationally tested on several VRP variants with time window constraints and/or large vehicle capacity. Pretests have shown that the labeling algorithm could perform very well, but that performance is rather sensitive to the choice of the step size parameter. Moreover, the good step size values varied strongly from instance to instance. Therefore, a simple but effective scheme for an automatic dynamic adjustment of the step size was devised. Other specific experiments indicate that, on the hardest instances, the new bucket arc fixing procedure can indeed be better than existing schemes for fixing arcs by reduced cost. Lastly, extensive experiments show that the final BCP algorithm outperforms significantly other recent state-of-art algorithms for the VRPTW and the Multi-Depot VRP with Distance Constraints (MDVRPDC). Moreover, our algorithm is the first exact approach that can handle medium-sized instances of the classical Distance Constrained Vehicle Routing Problem (DCVRP), including the site-dependent variant. Our algorithm is also the first exact method successfully applied for a set of "nightmare" instances of the Heterogeneous Fleet VRP (HFVRP).

The remainder of this article has the following structure. Section 2 defines the exact SPPRC solved in this paper. It is explained how this SPPRC arises as a pricing subproblem for solving a Heterogeneous Fleet Vehicle Routing Problem with Time Windows (HFVRPTW) by a column-and-cut generation algorithm. Section 3 describes the bucket graph labeling algorithm proposed in this work. Section 4 presents the bucket arc elimination procedure and introduces the key concept of jump bucket arcs. Section 5 presents the complete BCP algorithm for HFVRPTW where the bucket graph labeling algorithm was embedded. Section 6 presents results of the computational experiments. There are experiments devised to assess the impact of the bucket step size and experiments for measuring the impact of the introduced bucket arc elimination procedure. Moreover, extensive experiments evaluate the overall BCP performance on several classical VRP variants that are particular cases of the HFVRPTW. Section 7 contains final remarks. Finally, very detailed computational results are presented in the online appendix.

2 Application

In order to give a precise definition of the SPPRC addressed in this paper, we first need to define its application; defining the family of VRPs that we ultimately want to solve and outlining the column-and-cut generation algorithm where that SPPRC arises as the pricing subproblem.

2.1 HFVRPTW Definition

The Heterogeneous Fleet Vehicle Routing Problem with Time Windows (HFVRPTW) (Jiang et al. 2014) is defined as follows. Let $G = (V, A)$ be a directed graph with vertex set $V = \{0, \ldots, n+1\}$. For convenience, the depot vertex is split into the source vertex 0 and the sink vertex $n+1$; vertices in $V' = V \setminus \{0, n+1\}$ represent the n customers. The arc set is defined as $A = \{(v, v') : v, v' \in V, v \neq v', v \neq n+1, v' \neq 0, (v, v') \neq (0, n+1)\}$. Each vertex $v \in V$ has a demand w_v, a service time (duration) s_v, and a time window $[l_v, u_v]$ associated with it. Demands and service times are non-negative for customers $v \in V'$ and equal to zero for the depot vertices, l_0 is assumed to be equal to 0. The fleet is composed of a set M of different types of vehicles. For each $m \in M$, there are U_m available vehicles, each with a capacity W_m. Let $W = \max\{W_m : m \in M\}$ be the largest capacity. Every vehicle type is associated with a fixed cost denoted by f_m. For each arc $a \in A$ and $m \in M$ there is a cost c_a^m and a non-negative travel time t_a^m associated to the traversal of this arc by a vehicle of type m. We assume that G does not have any cycle where, for some vehicle type, all vertices have zero demands and zero service times and all arcs have zero time. A route $P = (v_0 = 0, v_1, \ldots, v_k, v_{k+1} = n+1)$ for a vehicle of type $m \in M$ is a walk where $v_1, \ldots, v_k$ are not necessarily distinct customers in V' and is said to be feasible if 1) it satisfies vehicle capacity: $\sum_{j=1}^{k} w_{v_j} \leq W_m$; and 2) the earliest start of service time S_j at every visited vertex v_j, $0 \leq j \leq k+1$, falls within the corresponding time window: $l_{v_j} \leq S_j \leq u_{v_j}$, where $S_0 = l_0$ and $S_j = \max\{l_{v_j}, S_{j-1} + s_{v_{j-1}} + t^m_{(v_{j-1}, v_j)}\}$. A vehicle may arrive at a vertex before the beginning of its time window and wait, but it cannot arrive after the end of the time window. The cost of route P is calculated as $f_m + \sum_{j=1}^{k+1} c^m_{(v_{j-1}, v_j)}$. A route is said to be elementary if the same customer is not visited more than once. The objective is to determine a set of feasible elementary routes with minimal total cost such that: (i) each customer is visited by exactly one route; (ii) the number of routes associated to a vehicle type does not exceed its availability. Clearly, some of most classical VRP variants, like CVRP, VRPTW and HFVRP, are particular cases of HFVRPTW. Other variants that are also particular cases of HFVRPTW are listed below.

In the Multi-Depot VRP (MDVRP), the vehicles themselves are identical, they only differ by being attached to different depots. It can be easily modeled as a HFVRP by associating each depot to a vehicle type and setting costs $c^m_{(0,v)}$ and times $t^m_{(0,v)}$ for leaving the depot, together with costs $c^m_{(v,n+1)}$ and times $t^m_{(v,n+1)}$ for entering the depot, that depend on $m \in M$. The Site Dependent VRP (SDVRP), a variant where vehicles differ only by the subset of the customers that they can visit, is also easily modeled as a HFVRP by setting infinite costs for $c^m_{(v,v')}$ if vehicle type m cannot visit either v or v'.

The classical Distance Constrained Vehicle Routing Problem (DCVRP) can also be solved as a HFVRPTW. In this variant, besides the capacity constraint, the route length is also forbidden to be above a certain threshold D. Since service times are included in that "length", this is actually a time limit for returning to the depot. It can be modeled by setting time window $[0, D]$ for every $v \in V$.

2.2 Set Partitioning Formulation and Cuts

Let Ω_m be the set of all feasible elementary routes for vehicle type $m \in M$. A set partitioning formulation over those sets would lead to a hard pricing problem, probably intractable on large instances. A more tractable (but weaker) formulation can be defined using a relaxed set of routes that includes some non-elementary routes, i.e. routes in which some customers are visited more than once. In this work we assume that the *ng*-route relaxation (Baldacci, Mingozzi, and Roberti 2011), a route relaxation with a good tradeoff between formulation strength and pricing difficulty, is being used. Let $N_v \subseteq V'$ be the neighborhood of $v \in V'$, typically containing the closest customers to v. An *ng*-route can only revisit a customer v, forming a cycle, if the cycle contains another customer v' with $v \notin N_{v'}$. In many cases, reasonably small neighborhoods (for example, with $|N_v| = 8$) already provide bounds that are close to those that would be obtained by pricing elementary routes (Poggi and Uchoa 2014, Contardo, Desaulniers, and Lessard 2015).

Let $\Omega_m^N \supseteq \Omega_m$ be the set of ng-routes for vehicle type $m \in M$ with respect to a given set of neighborhoods $N = (N_1, \ldots, N_n)$. We denote by c_P the cost of route $P \in \Omega_m^N$, by $x_{(v,v')}^P$ the number of times arc $(v, v') \in A$ appears in route P, and by $y_v^P = \sum_{a \in \delta^+(\{v\}) \cup \delta^-(\{v\})} x_a^P/2$ the number of times vertex $v \in V'$ is visited in route P (this "symmetric" definition of y_v^P is exploited in Section 3.6). Let λ_P be a binary variable indicating whether route P is selected or not in the solution. The Set Partitioning Formulation (SPF) for the HFVRPTW considered in this paper is the following.

$$\text{(SPF) min} \sum_{m \in M} \sum_{P \in \Omega_m^N} c_P \lambda_P \tag{1}$$

$$\text{subject to} \sum_{m \in M} \sum_{P \in \Omega_m^N} y_v^P \lambda_P = 1 \qquad \forall v \in V', \tag{2}$$

$$\sum_{P \in \Omega_m^N} \lambda_P \leq U_m \qquad \forall m \in M, \tag{3}$$

$$\lambda_P \in \{0, 1\} \qquad \forall m \in M, P \in \Omega_m^N. \tag{4}$$

A column generation algorithm should be used to solve the linear relaxation of (1)-(4). However, even using large neighborhoods (or even enforcing that all routes are elementary) the resulting bounds are often not good enough for building an effective branch-and-price algorithm. Therefore, the SPF should be reinforced by adding cuts.

Let $C \subseteq V'$ be a subset of the customers, define $w(C) = \sum_{v \in C} w_v$ as its total demand. The value $\lceil w(C)/W \rceil$ is a valid lower bound on the number of vehicles that must visit C. Therefore, the following Rounded Capacity Cut (RCC) (Laporte and Nobert 1983) is valid:

$$\sum_{m \in M} \sum_{P \in \Omega_m^N} \left(\sum_{a \in \delta^+(C) \cup \delta^-(C)} x_a^P \right) \lambda_P \geq 2\lceil w(C)/W \rceil. \tag{5}$$

RCCs are known to provide a significant reinforcement of the SPF on the CVRP (Fukasawa et al. 2006). However, they do not work so well on many HFVRP and VRPTW instances. This happens because the expression $\lceil w(C)/W \rceil$, which depends only on the largest vehicle capacity W and disregards time windows, can be a poor bound on the actual number of vehicles visiting C. The coefficient of a variable λ_P in a RCC is given by a linear expression over the values of x_a^P. This means that RCCs are *robust cuts* (Pessoa, de Aragão, and Uchoa 2008), such cuts do not have any impact in the difficulty of the pricing subproblem.

Using a Chvátal-Gomory rounding of a subset $C \subseteq V'$ of Constraints (2), relaxed to $\leq$, with multipliers p_v $(0 < p_v < 1)$, $v \in C$, the following valid Rank-1 Cut (R1C) is obtained:

$$\sum_{m \in M} \sum_{P \in \Omega_m^N} \left\lfloor \sum_{v \in C} p_v y_v^P \right\rfloor \lambda_P \leq \left\lfloor \sum_{v \in C} p_v \right\rfloor. \tag{6}$$

The Subset Row Cuts (SRCs) proposed in Jepsen et al. (2008) are the particular R1Cs with multipliers $p_v = 1/K$, where K is an integer, for all $v \in C$. Recently, the optimal multiplier vectors for R1Cs with up to five rows have been determined by Pecin et al. (2017c)

- For $|C| = 3$, $(\frac{1}{2}, \frac{1}{2}, \frac{1}{2})$.
- For $|C| = 4$, $(\frac{2}{3}, \frac{1}{3}, \frac{1}{3}, \frac{1}{3})$ and its permutations.
- For $|C| = 5$, $(\frac{1}{3}, \frac{1}{3}, \frac{1}{3}, \frac{1}{3}, \frac{1}{3})$, $(\frac{2}{4}, \frac{2}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4})$, $(\frac{3}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4}, \frac{1}{4})$, $(\frac{3}{5}, \frac{2}{5}, \frac{2}{5}, \frac{1}{5}, \frac{1}{5})$, $(\frac{1}{2}, \frac{1}{2}, \frac{1}{2}, \frac{1}{2}, \frac{1}{2})$, $(\frac{2}{3}, \frac{2}{3}, \frac{2}{3}, \frac{1}{3}, \frac{1}{3})$, $(\frac{3}{4}, \frac{3}{4}, \frac{2}{4}, \frac{2}{4}, \frac{1}{4})$ and their permutations.

They are optimal in the sense that they generate R1Cs that are equivalent or dominate the R1Cs generated with any other multipliers. When non-elementary routes can be generated, one can also apply cuts (6) with $C = \{v\}$ and $p_v = 1/2$ for some $v \in V'$.

R1Cs are known to be very effective. However, because of the rounding down operator, the coefficient of a variable λ_P in a R1C is not given by a linear expression over the values of x_a^P, so those cuts are non-robust. The concept of *limited-memory* for reducing the negative impact of R1Cs in the pricing, first proposed in Pecin et al. (2014, 2017b), represented a breakthrough in the performance of BCP algorithms for VRP. In this paper, we assume that the generalized arc memory variant of the concept (Pecin et al. 2017a) is being used. Each R1C, indexed by ℓ, is associated to a customer subset $C^\ell \subseteq V'$, a multiplier vector $(p_v^\ell)_{v \in C^\ell}$, and an arc memory set AM^ℓ with $AM^\ell \subseteq A$. The idea is that, when a route $P \in \Omega_m^N$ leaves the memory set, i.e. follows an arc not in AM^ℓ, it "forgets" previous visits to nodes in C^ℓ, yielding a coefficient for λ_P in the cut ℓ that may be smaller than the original coefficient $\lfloor \sum_{v \in C^\ell} p_v^\ell y_v^P \rfloor$. The memory set AM^ℓ is defined during the cut separation procedure as a minimal set preserving the coefficients of the route variables λ_P that have a positive value in the current linear relaxation solution. Given $C \subseteq V'$, a multiplier vector p of dimension $|C|$, and a memory arc set $AM \subseteq A$, the limited-arc-memory R1C cut is defined as:

$$\sum_{m \in M} \sum_{P \in \Omega_m^N} \alpha(C, AM, p, P) \lambda_P \leq \left\lfloor \sum_{v \in C} p_v \right\rfloor \tag{7}$$

where the coefficient $\alpha(C, AM, p, P)$ of the route variable λ_P, $P \in \Omega_m^N$, is computed by the following pseudocode:

Function $\alpha(C,\ AM,\ p,\ P = (v_0 = 0, v_1, \ldots, v_k, v_{k+1} = n+1))$

```
α ← 0, S ← 0;
for j = 1 to k do
    if (v_{j-1}, v_j) ∉ AM then
        S ← 0;
    if v_j ∈ C then
        S ← S + p_{v_j};
        if S ≥ 1 then
            S ← S − 1, α ← α + 1;
return α;
```

This function can be explained as follows. Whenever a route P visits a vertex $v \in C$, the multiplier p_v is added to the state variable $\mathcal{S}$. When $\mathcal{S} \geq 1$, $\mathcal{S}$ is decremented and α is incremented. If $AM = A$, the function returns $\lfloor \sum_{v \in C} p_v y_v^P \rfloor$ and the limited-memory cut would be equivalent to the original cut. On the other hand, if $AM \subset A$, it may happen that P uses an arc not in AM when $\mathcal{S} > 0$, causing the state $\mathcal{S}$ to be reset to zero and "forgetting" some previous visits to nodes in C. In this case, the returned coefficient may be less than the original coefficient. However, memory sets can be "corrected" in the next separation round. The final linear relaxation bound obtainable with limited memory R1Cs is exactly that potentially achieved with ordinary R1Cs.

2.3 Pricing Problem

We now define the pricing problem for the SPF over ng-routes and with additional RCCs and limited-memory R1Cs, which decomposes into subproblems, one for each vehicle type. In the subproblem for type $m \in M$, we search for a route $P \in \Omega_m^N$ with the minimum reduced cost. Let $\mathcal{J}$ be the current set of active RCCs (5), and $\mathcal{L}$ be the current set of active limited memory R1Cs (7). Each cut $\jmath \in \mathcal{J}$ is determined by a set $C^\jmath$ of customers. Each cut $\ell \in \mathcal{L}$ is determined by the triple $(C^\ell, AM^\ell, p^\ell)$. Let (π, μ, ν, σ) be the current dual solution of the linear relaxation of (1)-(4) with sets $\mathcal{J}$ and $\mathcal{L}$ of cuts added, where π corresponds to the partitioning constraints (2), μ corresponds to Constraints (3), ν corresponds to the set $\mathcal{J}$ of active RCCs, and σ corresponds

to the set $\mathcal{L}$ of active limited memory R1Cs. Let $\bar{c}^m(\pi,\nu)$ be the vector of current arc reduced costs, where

$$\bar{c}^m_{(v,v')}(\pi,\nu) = c^m_{(v,v')} - \frac{\pi_v}{2} - \frac{\pi_{v'}}{2} - \sum_{\substack{\jmath\in\mathcal{J}:\\ (v,v')\in\delta^+(C^\jmath)\cup\delta^-(C^\jmath)}} \nu_\jmath, \tag{8}$$

where π_0 and π_{n+1} are defined as zero. The pricing subproblem (PSP^m) for vehicle type $m \in M$ is then formulated as

$$\min_{P\in\Omega^N_m}\left(\sum_{a\in P}\bar{c}^m_a(\pi,\nu) - \sum_{\ell\in\mathcal{L}}\sigma_\ell\cdot\alpha(C^\ell, AM^\ell, p^\ell, P) - \mu_m\right).$$

3 Bucket Graph based Labeling Algorithm

3.1 SPPRC over Forward and Backward graphs

We now formulate the pricing problem defined in Section 2.3 as a SPPRC in a forward graph $\vec{G} = (V, \vec{A})$ that is identical to G, so $\vec{A} = A$. In this section we consider the dual solution (π, μ, ν, σ) fixed and also drop index m for more clarity. Therefore, the reduced cost of an arc $a \in \vec{A}$ is denoted simply as $\bar{c}_a$. Define a set of two resources $R = \{1, 2\}$ corresponding to vehicle capacity and time. The capacity resource consumption of each arc $(v, v') \in \vec{A}$ equals to $w_{v'}$ and the time resource consumption of the same arc equals to $s_v + t_{(v,v')}$. For convenience, we use the same notation $d_{\vec{a},r}$ for the resource consumption of an arc $\vec{a} \in \vec{A}$ for both resources $r \in R$. Each vertex $v \in V$ has lower and upper bounds $l_{v,r}$ and $u_{v,r}$ on the accumulated resource consumption to v of each resource $r \in R$. For the second resource (time) these bounds correspond to original time windows. For the first (capacity) resource, we have $l_{v,1} = 0$ and $u_{v,1} = W$. To simplify the notation, we often drop the resource index r to represent a vector with the corresponding values for all resources in R, e.g. $l_v = (l_{v,1}, \ldots, l_{v,|R|})^\top$. Moreover, we use $x \odot y$ to denote the component-wise product of two vectors x and y. We always represent a forward path $\vec{P}$ in graph $\vec{G}$ as defined by its arcs: $\vec{P} = (\vec{a}^{\vec{P}}_1, \vec{a}^{\vec{P}}_2, \ldots, \vec{a}^{\vec{P}}_{|\vec{P}|})$, with $\vec{a}^{\vec{P}}_j = (v^{\vec{P}}_{j-1}, v^{\vec{P}}_j)$, $v^{\vec{P}}_0 = 0$, $v^{\vec{P}}_{|\vec{P}|} = n + 1$. For lightening the notation, we may drop the superscript $\vec{P}$ when it is clear from the context. Path $\vec{P}$ yields a vector of accumulated resource consumptions $q^{\vec{P}}_j \in \mathbb{R}^{|R|}$ at vertices that can be computed as:

$$q^{\vec{P}}_j = \begin{cases} l_0 & \text{if } j = 0; \\ \max\left\{q^{\vec{P}}_{j-1} + d_{\vec{a}_j},\ l_{v_j}\right\}, & \text{otherwise} \end{cases} \tag{9}$$

where the max denotes the component-wise maximum function of two vectors. A forward path $\vec{P}$ is feasible if it respects

- the resource consumption bounds:

$$q^{\vec{P}}_j \le u_{v_j}, \quad \forall j \in \{0, \ldots, |\vec{P}|\}; \tag{10}$$

- and ng-path neighborhoods:

$$\forall (i,j) \in \{(i,j) :\ 0 \le i < j \le |\vec{P}|,\ v_i = v_j = v\} \quad \exists h :\ i < h < j,\ v \notin N_{v_h}. \tag{11}$$

Condition (11) can be rewritten in the following way. Let $\mathcal{F}^{\vec{P}}_j$ be the set of vertices defined as $\mathcal{F}^{\vec{P}}_0 = \emptyset$ and $\mathcal{F}^{\vec{P}}_j = (\mathcal{F}^{\vec{P}}_{j-1} \cap N_{v_j}) \cup \{v_j\}$ for all $1 \le j \le |\vec{P}|$. Then (11) is equivalent to $v_j \notin \mathcal{F}^{\vec{P}}_{j-1}$ for all $1 \le j \le |\vec{P}|$. Let $\bar{c}^{\vec{P}}(\pi, \mu, \nu, \sigma)$ be the total reduced cost of path $\vec{P}$:

$$\bar{c}^{\vec{P}}(\pi,\mu,\nu,\sigma) = \sum_{\vec{a}\in\vec{P}}\bar{c}_{\vec{a}}(\pi,\nu) - \sum_{\ell\in\mathcal{L}}\sigma_\ell\cdot\alpha(C^\ell, AM^\ell, p^\ell, \vec{P}) - \mu.$$

Let also $\vec{\mathcal{P}}$ be the set of all feasible forward paths. The pricing subproblem then can be reformulated as finding a path in $\vec{\mathcal{P}}$ minimizing the total reduced cost: $\min_{\vec{P}\in\vec{\mathcal{P}}} \bar{c}^{\vec{P}}(\pi,\mu,\nu,\sigma)$. In the remainder of this section, we use simplified notations $\bar{c}^P$ and $\bar{c}_a$ as the dual solution (π,μ,ν,σ) is fixed.

We now define the backward graph $\overleftarrow{G} = (V, \overleftarrow{A})$ with source $n+1$ and sink 0. Set $\overleftarrow{A}$ contains one backward arc $\overleftarrow{a} = (v', v)$ for each forward arc $\vec{a} = (v, v') \in \vec{A}$. Each backward arc $\overleftarrow{a}$ has the same reduced cost $\bar{c}_{\overleftarrow{a}} = \bar{c}_{\vec{a}}$ and same resource consumption $d_{\overleftarrow{a}} = d_{\vec{a}}$ as the corresponding forward arc.

A backward path $\overleftarrow{P}$ is denoted by $\overleftarrow{P} = (\overleftarrow{a}_1, \overleftarrow{a}_2, \ldots, \overleftarrow{a}_{|\overleftarrow{P}|})$ in graph $\overleftarrow{G}$, $\overleftarrow{a}_j = (v_{j-1}, v_j)$, $v_0 = n+1$, $v_{|\overleftarrow{P}|} = 0$, possibly containing cycles, where we may add the superscript $\overleftarrow{P}$ if necessary as in the forward graph. The accumulated consumption vector $q_j^{\overleftarrow{P}} \in \mathbb{R}^{|R|}$ in $\overleftarrow{P}$ is calculated recursively as

$$q_j^{\overleftarrow{P}} = \begin{cases} u_{n+1}, & \text{if } j = 0; \\ \min\left\{q_{j-1}^{\overleftarrow{P}} - d_{\overleftarrow{a}_j},\ u_{v_j}\right\}, & \text{otherwise.} \end{cases}$$

where the min denotes the component-wise minimum function of two vectors. Feasibility conditions for a backward path $\overleftarrow{P}$ are the same as for a forward path except that (10) is replaced by $q_j^{\overleftarrow{P}} \geq l_{v_j}$, for all $j \in \{0, \ldots, |\overleftarrow{P}|\}$. We denote as $\overleftarrow{\mathcal{P}}$ the set of all feasible backward paths. Sets $\mathcal{F}_j^{\overleftarrow{P}}$, $0 \leq j \leq |\overleftarrow{P}|$, are defined in the same way as for a forward path.

It is easy to see that a forward path $\vec{P}$ belongs to $\vec{\mathcal{P}}$ if and only if the corresponding backward path $\overleftarrow{P}$ belongs to $\overleftarrow{\mathcal{P}}$, where $|\vec{P}| = |\overleftarrow{P}|$, $\vec{a}_j = (v, v')$ and $\overleftarrow{a}_{|P|+1-j} = (v', v)$, $1 \leq j \leq |\vec{P}|$. Also, we have $q_j^{\vec{P}} \leq q_{|\overleftarrow{P}|-j}^{\overleftarrow{P}}$ for $0 \leq j \leq |\vec{P}|$. An example of a pair of forward and backward paths together with their consumption of one resource is given in Figure 1. Moreover, the total reduced costs of $\overleftarrow{P}$ and $\vec{P}$ are equal, as coefficients $\alpha(C, AM, p, P)$ do not change if we traverse path P in the backward order in function α. Therefore the pricing problem defined in Section 2.3 can be also reformulated as finding a path in $\overleftarrow{\mathcal{P}}$ minimizing the total reduced cost.

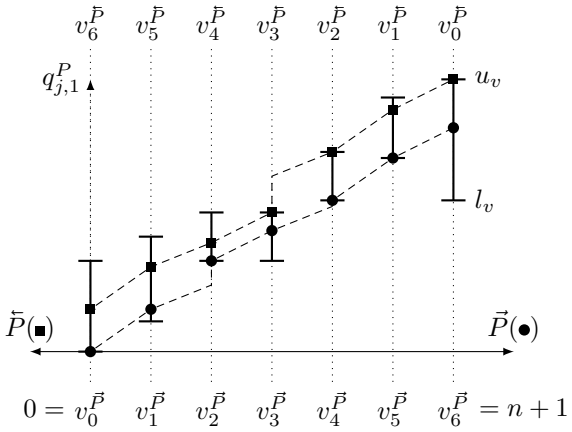

Figure 1: An example of a forward and the corresponding backward paths in the case of one resource

From now on, we use accent $\vec{\circ}$ for a forward entity, accent $\overleftarrow{\circ}$ for a backward entity, and no accent for an entity which can be both forward or backward (when it is applied). We also use accent $\overleftrightarrow{\circ}$ for an entity of the opposite sense.

3.2 Labels and Dominance Rule

We devise a labeling algorithm for the previously defined SPPRC where each label $L = (\bar{c}^L, v^L, q^L, \mathcal{F}^L, \mathcal{S}^L)$ corresponds to a partial forward or backward path P (partial means that we may have $v^P_{|P|} \notin \{0, n+1\}$), where $\bar{c}^L = \bar{c}^P$, $v^L = v^P_{|P|}$, $q^L = q^P_{|P|}$, $\mathcal{F}^L = \mathcal{F}^P_{|P|}$, and $\mathcal{S}^L \in \mathbb{R}^{|\mathcal{L}|}$ gives the current state (computed as in Function α) of each cut $\ell \in \mathcal{L}$ for label L.

A label L dominates label L' if $v^L = v^{L'}$, $q^L \leq q^{L'}$ ($q^L \geq q^{L'}$ for backward labels), $\mathcal{F}^L \subseteq \mathcal{F}^{L'}$, and

$$\bar{c}^L - \sum_{\ell \in \mathcal{L}:\ \mathcal{S}^L_\ell > \mathcal{S}^{L'}_\ell} \sigma_\ell \leq \bar{c}^{L'}, \tag{12}$$

as $\sigma_\ell \leq 0$, $\ell \in \mathcal{L}$ (Jepsen et al. 2008). Condition $\bar{c}^L > \bar{c}^{L'}$ is sufficient to verify that L does not dominate L'.

3.3 Bucket Graph

A critical aspect of labeling algorithms that solve the previously described SPPRC is when to perform dominance checks. Given a set of labels that potentially dominate each other, dominance checks may be performed for all pairs of labels. However, it can be prohibitively time consuming. Alternatively, skipping too many dominance checks may cause a premature explosion on the number of maintained labels. One approach to address this issue is to partition the labels into buckets and to ensure that no pair of labels inside a bucket dominate each other. Pecin et al. (2017b) followed this approach by defining buckets for every possible resource consumption of labels and skipping inter-bucket dominance checks. In this paper we define buckets differently in order to avoid resource discretisation. Also, we perform inter-bucket dominance checks, but less frequently.

In the proposed labeling algorithm, labels are grouped into buckets based on their final vertices and on ranges defined for both accumulated resource consumption values. Moreover, we define bucket arcs connecting pairs of buckets through which the labels can be extended. Different bucket graphs are then defined for forward and backward labeling. Bucket graphs are useful because they help to determine an efficient order of treatment for the buckets. If the bucket graph is acyclic, it is desirable to process the buckets in its topological order because no further extension from a bucket is necessary after it has been processed. If the bucket graph contains cycles, then buckets are handled in the topological order of the graph's strongly connected components, trying to minimize such reprocessing. Additionally, the bucket graph is used to avoid label extensions that are proved not to contribute to a solution that improves the current best one. This is achieved by removing arcs from the bucket graph based on a reduced cost argument. In what follows, we present all the notation required to formalize these ideas.

The set of buckets associated to a given vertex $v \in V$ is determined by a step size $\tilde{d}_r$ for each resource $r \in R$. Let $\mathcal{K}_v$ be the set of all $|R|$-dimensional integer vectors such that the r-th component belongs to $\{0, 1, \ldots, \lfloor (u_{v,\,r} - l_{v,\,r})/\tilde{d}_r \rfloor\}$, $r \in R$. A forward bucket $\vec{b}$ is defined by a pair of vertex and lower bound vector $(\tilde{v}_{\vec{b}}, \tilde{l}_{\vec{b}}) = (v, l_v + \kappa \odot \tilde{d})$, $v \in V$, $\kappa \in \mathcal{K}_v$. A forward label $\vec{L}$ is contained in forward bucket $\vec{b}$ if $v^{\vec{L}} = \tilde{v}_{\vec{b}}$ and $\tilde{l}_{\vec{b}} \leq q^{\vec{L}} < \tilde{l}_{\vec{b}} + \tilde{d}$. Similarly, a backward bucket $\overleftarrow{b}$ is defined by a pair of vertex and upper bound vector $(\tilde{v}_{\overleftarrow{b}}, \tilde{u}_{\overleftarrow{b}}) = (v, u_v - \kappa \odot \tilde{d})$, $v \in V$, $\kappa \in \mathcal{K}_v$. A backward label $\overleftarrow{L}$ is contained in backward bucket $\overleftarrow{b}$ if $v^{\overleftarrow{L}} = \tilde{v}_{\overleftarrow{b}}$ and $\tilde{u}_{\overleftarrow{b}} \geq q^{\overleftarrow{L}} > \tilde{u}_{\overleftarrow{b}} - \tilde{d}$. Let b^L be the bucket containing label L.

For a forward (backward) bucket b, let κ^b be the vector $\kappa \in \mathcal{K}_{\tilde{v}_b}$ such that $\tilde{l}_b = l_{\tilde{v}_b} + \kappa \odot \tilde{d}$ ($\tilde{u}_b = u_{\tilde{v}_b} - \kappa \odot \tilde{d}$). We say that bucket b' is *component-wise smaller* than bucket b (denoted as $b' \prec b$) if buckets are of the same sense, $\tilde{v}_{b'} = \tilde{v}_b$, and $\kappa^{b'} \leq \kappa^b$.

Let $\vec{B}$ ($\overleftarrow{B}$) be the set of all forward (backward) buckets. We define as $\vec{\Gamma}$ ($\overleftarrow{\Gamma}$) the following set of directed forward (backward) *bucket arcs*.

Each forward/backward bucket arc $\gamma \in \Gamma$ is defined by a pair (b_γ, a_γ) of bucket and arc of the corresponding sense, $b_\gamma \in B$, $a_\gamma \in A$, and $\tilde{v}_{b_\gamma}$ is the tail of a_γ. The tail of a forward/backward bucket arc γ is bucket b_γ. Set $\vec{\Gamma}$ contains $\vec{\gamma}$ if $\tilde{l}_{\vec{b}_{\vec{\gamma}}} + d_{\vec{a}_{\vec{\gamma}}} \leq u_{v'}$, where v' is the head of $\vec{a}_{\vec{\gamma}}$. The head of a forward bucket arc $\vec{\gamma} \in \vec{\Gamma}$, denoted by $b^{\text{head}}_{\vec{\gamma}}$, is the bucket $\vec{b}' \in \vec{B}$ such that $\tilde{v}_{\vec{b}'} = v'$ and $\tilde{l}_{\vec{b}'} \leq \max\left\{\tilde{l}_{\vec{b}_{\vec{\gamma}}} + d_{\vec{a}_{\vec{\gamma}}}, l_{v'}\right\} < \tilde{l}_{\vec{b}'} + \tilde{d}$. Set $\overleftarrow{\Gamma}$ contains $\overleftarrow{\gamma}$ if $\tilde{u}_{\overleftarrow{b}_{\overleftarrow{\gamma}}} - d_{\overleftarrow{a}_{\overleftarrow{\gamma}}} \geq l_{v'}$, where v' is the head of $\overleftarrow{a}_{\overleftarrow{\gamma}}$. The head of a backward bucket arc $\overleftarrow{\gamma}$, denoted by $b^{\text{head}}_{\overleftarrow{\gamma}}$, is the bucket $\overleftarrow{b}' \in \overleftarrow{B}$ such that $\tilde{v}_{\overleftarrow{b}'} = v'$ and $\tilde{u}_{\overleftarrow{b}'} \geq \min\left\{\tilde{u}_{\overleftarrow{b}_{\overleftarrow{\gamma}}} - d_{\overleftarrow{a}_{\overleftarrow{\gamma}}}, u_{v'}\right\} > \tilde{u}_{\overleftarrow{b}'} - \tilde{d}$. Let $\vec{\mathcal{B}} = (\vec{B}, \vec{\Gamma})$ and $\overleftarrow{\mathcal{B}} = (\overleftarrow{B}, \overleftarrow{\Gamma})$ be the directed forward and backward bucket graphs, respectively.

For each forward/backward bucket $b \in B$, we define the set Φ_b of "adjacent" component-wise smaller buckets:

$$\Phi_b = \left\{b' :\ \tilde{v}_{b'} = \tilde{v}_b,\ \exists r' \in R, \kappa^{b'}_{r'} = \kappa^b_{r'} - 1,\ \kappa^{b'}_r = \kappa^b_r,\ \forall r \in R \setminus \{r'\}\right\}.$$

As $|R| = 2$, we have $|\Phi_b| \leq 2$. By definition, we have $b' \prec b$ for all $b' \in \Phi_b$. Let $\vec{\Gamma}^\Phi$ ($\overleftarrow{\Gamma}^\Phi$) be the set of directed forward (backward) bucket arcs (b', b) such that $b \in B$, $b' \in \Phi_b$. Let $\vec{\mathcal{B}}^\Phi = (\vec{B}, \phi(\vec{\Gamma}) \cup \vec{\Gamma}^\Phi)$ be the extended forward bucket graph, where $\phi(\vec{\Gamma}) = \{(b_{\vec{\gamma}}, b^{\text{head}}_{\vec{\gamma}}) :\ \vec{\gamma} \in \vec{\Gamma}\}$. Analogously, $\overleftarrow{\mathcal{B}}^\Phi = (\overleftarrow{B}, \phi(\overleftarrow{\Gamma}) \cup \overleftarrow{\Gamma}^\Phi)$ is the extended backward bucket graph. Those extended graphs are not used in the core of the labeling algorithm, they are needed only in the initialization to compute their sets of strongly connected components and appropriate topological orders for them. Let $\mathcal{C}_b$ be the strongly connected component of a forward/backward bucket b.

Forward/backward label L, obtained by extension from label L' along a bucket arc γ with $b_\gamma = b^{L'}$, is not necessary contained in bucket b^{head}_γ. For forward labels, this happens when $q^{L'}_r + d_{a_\gamma, r} \geq \tilde{l}_{b^{\text{head}}_\gamma, r} + \tilde{d}_r$, for some $r \in R$. The backward case is analogous. So, we may have $b^{\text{head}}_\gamma \prec b^L$. Even in this case, we still consider that such an extension has been made along γ. The bucket arcs in Γ^Φ ensure that there is always a path from $b^{L'}$ to b^L in the extended forward bucket graph. Therefore, $\mathcal{C}_{b^L}$ cannot be before $\mathcal{C}_{b^{L'}}$ in the topological order of strongly connected components in Γ^Φ.

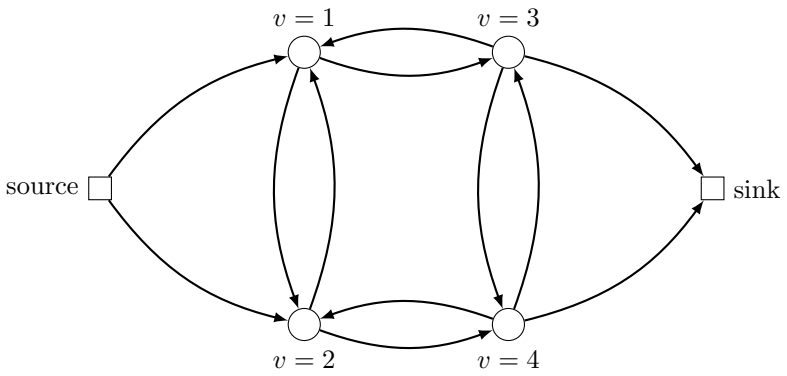

Figure 2: An example of Forward Graph

Figure 3 shows a small extended bucket graph that corresponds to the forward graph depicted in Figure 2. In this figure, each rectangle represents the space of possible resource consumption vectors for partial paths finishing in a corresponding vertex. The consumptions of the first and second resource determine the shifts in the horizontal and vertical directions, respectively. Thus, each square is associated with two ranges for the consumptions of both resources whose dimensions are determined by the bucket steps. Each node of the extended bucket graph is depicted inside the corresponding square. The bucket arcs that belong to $\vec{\Gamma}$ and $\vec{\Gamma}^\Phi$ correspond to arrows between rectangles and between squares of the same rectangle, respectively. Finally, an example of a strongly connected component of this extended bucket graph is bold printed in the figure.

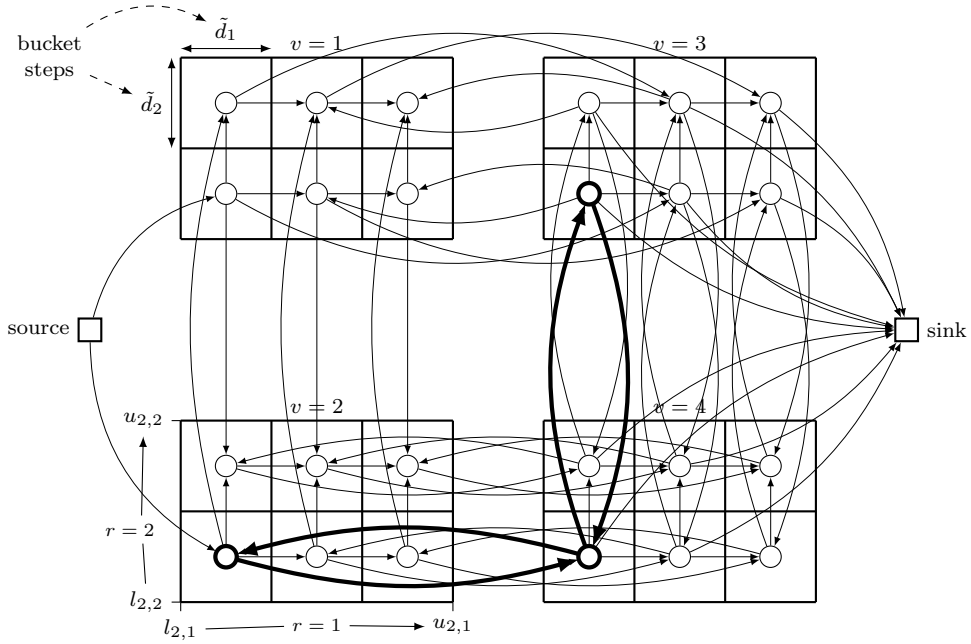

Figure 3: An example of Extended Bucket Graph for the forward graph of Figure 2

3.4 Mono-directional labeling algorithm

In this subsection, we describe the mono-directional labeling algorithm, which can be run in either forward or backward sense to solve the pricing problem. An auxiliary function Extend(L', γ, L) presented below extends a label L' to label L along a bucket arc $\gamma \in \Gamma$. It returns false if this extension is not possible. One can easily see that the steps performed by this function initialize the reduced cost and the final vertex of the new label, calculate its resource consumptions and check their bounds, obtain its *ng*-route information and check its feasibility, and finally update its reduced cost considering the active R1Cs.

The labeling algorithm is presented in Algorithm 1. In this algorithm, we maintain values $\bar{c}_b^{best}$ equal to the smallest reduced cost of labels L such that $b^L \preceq b$. It also uses the auxiliary function DominatedInCompWiseSmallerBuckets$(L, b, B^{visited})$ presented below, which checks whether a label L is dominated by a label contained in a bucket b' such that $b' \preceq b$, $b' \notin B^{visited}$. To avoid checking all component-wise smaller buckets, it assumes that the values of $\bar{c}_b^{best}$ are updated (for all buckets preceding b^L in the topological order) and uses it to prune the search. If the search reaches a bucket b' such that $\bar{c}_{b'}^{best} > \bar{c}^L$, one can conclude that no label in b' or in the component-wise smaller buckets can dominate L, and this search branch can be pruned. This function also receives the set $B^{visited}$ of visited buckets needed to avoid processing the same bucket twice. For our case with one or two resources it is easy to do a specific implementation to verify whether a bucket belongs to set $B^{visited}$ in a constant time. The same implementation can be used in Functions ConcatenateLabel and UpdateBucketsSet described later.

Two buckets b and b' such that $b' \in \Phi_b$ may belong to the same strongly connected component. This may happen, for example, when b and b' are forward buckets and there is another forward bucket b'' with $\tilde{v}_{b''} \neq \tilde{v}_{b'} = \tilde{v}_b$ such that $\tilde{l}_{b',r} < \tilde{l}_{b'',r} < \tilde{l}_{b,r}$, for some $r \in R$. In that case, the extended bucket graph may contain the directed cycle $((b', b), (b, b''), (b'', b'))$. Therefore values $\bar{c}_b^{best}$ for buckets in the same strongly connected component are calculated in a lexicographic order of κ^b to ensure that $\bar{c}_{b'}^{best}$ is set before $\bar{c}_b^{best}$.

Function Extend(L', γ, L)

$\bar{c}^L \leftarrow \bar{c}^{L'} + \bar{c}_{a_\gamma}$, $v^L \leftarrow$ the head of a_γ
$q^L \leftarrow q^{L'}$
if *L is a forward label* **then**
  $q^L \leftarrow \max\{q^L + d_{a_\gamma}, l_{v^L}\}$
  if $q^L > u_{v^L}$ **then return** false
if *L is a backward label* **then**
  $q^L \leftarrow \min\{q^L - d_{a_\gamma}, u_{v^L}\}$
  if $q^L < l_{v^L}$ **then return** false
if $v^L \in \mathcal{F}^{L'}$ **then return** false
$\mathcal{F}^L \leftarrow (\mathcal{F}^{L'} \cap N_{v^L}) \cup \{v^L\}$
for $\ell \in \mathcal{L}$ **do**
  if $a_\gamma \in AM^\ell$ **then** $\mathcal{S}^L_\ell \leftarrow \mathcal{S}^{L'}_\ell$
  else $\mathcal{S}^L_\ell \leftarrow 0$
  if $v^L \in C^\ell$ **then**
    $\mathcal{S}^L_\ell \leftarrow \mathcal{S}^L_\ell + p^\ell_{v^L}$
    if $\mathcal{S}^L_\ell \geq 1$ **then**
      $\mathcal{S}^L_\ell \leftarrow \mathcal{S}^L_\ell - 1$, $\bar{c}^L \leftarrow \bar{c}^L - \sigma_\ell$
return true

3.5 Bi-directional labeling algorithm

We call a pair of forward and backward labels $\vec{L}$ and $\overleftarrow{L}$ *ω-compatible* if

$$v^{\vec{L}} \neq v^{\overleftarrow{L}}, \quad q^{\vec{L}} + d_{(v^{\vec{L}}, v^{\overleftarrow{L}})} \leq q^{\overleftarrow{L}}, \quad \mathcal{F}^{\vec{L}} \cap \mathcal{F}^{\overleftarrow{L}} = \emptyset, \text{ and } \bar{c}(P^{\vec{L}} || P^{\overleftarrow{L}}) < \omega,$$

where $P^{\vec{L}} || P^{\overleftarrow{L}}$ is the path obtained by concatenation of partial paths $P^{\vec{L}}$ and $P^{\overleftarrow{L}}$ along arc $(v^{\vec{L}}, v^{\overleftarrow{L}}) \in \vec{A}$, and its total reduced cost $\bar{c}(P^{\vec{L}} || P^{\overleftarrow{L}})$ is calculated as

$$\bar{c}(P^{\vec{L}} || P^{\overleftarrow{L}}) = \bar{c}^{\vec{L}} + \bar{c}_{(v^{\vec{L}}, v^{\overleftarrow{L}})} + \bar{c}^{\overleftarrow{L}} - \sum_{\substack{\ell \in \mathcal{L}: \\ \mathcal{S}^{\vec{L}}_\ell + \mathcal{S}^{\overleftarrow{L}}_\ell \geq 1}} \sigma_\ell.$$

For a pair of forward and backward labels $\vec{L}$ and $\overleftarrow{L}$ such that $v^{\vec{L}} \neq v^{\overleftarrow{L}}$, $\bar{c}^{\vec{L}} + \bar{c}_{(v^{\vec{L}}, v^{\overleftarrow{L}})} + \bar{c}^{\overleftarrow{L}}$ is a lower bound on value $\bar{c}(P^{\vec{L}} || P^{\overleftarrow{L}})$.

For the bi-directional labeling algorithm, we need to choose a resource $r^* \in R$ and a threshold value $q^* \in [l_{0,r^*}, u_{n+1,r^*}]$ for it. The bi-directional labeling algorithm is presented in Algorithm 2. It uses the auxiliary procedure ConcatenateLabel($\vec{L}, \overleftarrow{b}, P^{best}, \overleftarrow{B}^{visited}$), presented below, which tries to find a backward label $\overleftarrow{L}$ such that $\overleftarrow{b}^{\overleftarrow{L}} \preceq \overleftarrow{b}$, $\overleftarrow{b}^{\overleftarrow{L}} \notin \overleftarrow{B}^{visited}$, and such that pair $(\vec{L}, \overleftarrow{L})$ is $\bar{c}^{P^{best}}$-compatible, i.e. concatenation of partial paths $P^{\vec{L}}$ and $P^{\overleftarrow{L}}$ along arc $(v^{\vec{L}}, v^{\overleftarrow{L}})$ improves on P^{best}. We use values $\bar{c}^{best}$ as bounds to prune the search. From the reasoning of the previous paragraph, if $\bar{c}^{\vec{L}} + \bar{c}_{(v^{\vec{L}}, \tilde{v}_{\overleftarrow{b}})} + \bar{c}^{best}_{\overleftarrow{b}} \geq \bar{c}^{P^{best}}$ then such a label does not exist. All buckets that may contain extensions for $\vec{L}$ that improve on P^{best} are tried. In the literature, values $\bar{c}^{best}_{\overleftarrow{b}}$ are referred as *completion bounds* (Christofides, Mingozzi, and Toth 1981).

3.6 Symmetric case

In this section, we suppose that 1) all time windows are the same; and 2) R1Cs arc memories are symmetric: $(v, v') \in AM^\ell$ if and only if $(v', v) \in AM^\ell$ for all $\ell \in \mathcal{L}$. We now show that in this case the forward/backward route symmetry can be exploited.

Algorithm 1: Mono-directional labeling algorithm

if *forward algorithm* **then** $L^{init} \leftarrow (0, 0, l_0, \emptyset, \mathbf{0})$
if *backward algorithm* **then** $L^{init} \leftarrow (0, n+1, u_{n+1}, \emptyset, \mathbf{0})$
insert initial label L^{init} to its bucket $b^{L^{init}}$ and mark L^{init} as non-extended
foreach *strongly connected component* $\mathcal{C}$ *in* $\mathcal{B}^{\Phi}$ *in a topological order* **do**
  repeat
    foreach *bucket* $b \in \mathcal{C}$ **do**
      foreach *non-extended label* $L' : b^{L'} = b$ **do**
        if not DominatedInCompWiseSmallerBuckets$(L', b^{L'}, \emptyset)$ **then**
          foreach *bucket arc* $\gamma \in \Gamma$ *such that* $b_\gamma = b^{L'}$ **do**
            if Extend(L', γ, L) **then**
              if L *is not dominated by a label in* b^L **then**
                mark L as non-extended and insert in b^L
                remove labels dominated by L from b^L
        mark L' as extended
  until *all labels in all buckets* $b \in \mathcal{C}$ *are extended*
  foreach *bucket* $b \in \mathcal{C}$ *in a lexicographic order of* κ^b **do**
    $\bar{c}_b^{best} \leftarrow \min\left\{\min_{L:\, b^L = b}\{\bar{c}^L\}, \min_{b' \in \Phi_b}\{\bar{c}_{b'}^{best}\}\right\}$
return $P^{best} = \operatorname{argmin}_{P^L : v^L = v'} \bar{c}^L$, where v' is the sink in G

Function DominatedInCompWiseSmallerBuckets$(L, b, B^{visited})$

$B^{visited} \leftarrow B^{visited} \cup \{b\}$
if $\mathcal{C}_b$ *precedes* $\mathcal{C}_{b^L}$ *in the topological order used* **and** $\bar{c}^L < \bar{c}_b^{best}$ **then return** false
if $b \neq b^L$ **and** L *is dominated by a label in bucket* b **then return** true
for $b' \in \Phi_b \setminus B^{visited}$ **do**
  if DominatedInCompWiseSmallerBuckets$(L, b', B^{visited})$ **then return** true
return false

First, we redefine the resource consumption of arcs. We make the capacity resource consumption of each arc $(v, v') \in \vec{A}$ equal to $\frac{1}{2}w_v + \frac{1}{2}w_{v'}$, and we make the time resource consumption of this arc equal to $\frac{1}{2}s_v + t_{(v,v')} + \frac{1}{2}s_{v'}$. Thus, the cost and the resource consumption of two arcs (v, v') and (v', v), $v, v' \in V$, become the same. Moreover, as for any route P coefficients $x^P_{(v,v')}$ and $x^P_{(v',v)}$ are the same in constraints (2) and cuts (5), reduced costs of these arcs are the same:

$$\bar{c}_{(v,v')} = \bar{c}_{(v',v)}, \quad \forall v, v' \in V. \tag{13}$$

Consider now a backward label $\overleftarrow{L}$ and the corresponding partial path $\overleftarrow{P} = \overleftarrow{P}^{\overleftarrow{L}}$. Let $\vec{P}$ be the partial forward path such that $v_j^{\vec{P}} = v_j^{\overleftarrow{P}}$, $1 \leq j \leq |\overleftarrow{P}|$, and $\vec{L}$ be the forward label such that $\vec{P}^{\vec{L}} = \vec{P}$. From (13) and from the fact that the R1Cs arc memories are symmetric, we have $\bar{c}^{\vec{L}} = \bar{c}^{\overleftarrow{L}}$, $\mathcal{F}^{\vec{L}} = \mathcal{F}^{\overleftarrow{L}}$, and $\mathcal{S}^{\vec{L}} = \mathcal{S}^{\overleftarrow{L}}$. As time windows are the same, resource bounds are also the same: $[l_v, u_v] = [l, u]$ for all $v \in V$. Then, $q^{\vec{L}} - l = u - q^{\overleftarrow{L}}$. Moreover, for every backward bucket $\overleftarrow{b}$, there exists a symmetric forward bucket $\vec{b}$ such that $\tilde{l}_{\vec{b}} - l = u - \tilde{u}_{\overleftarrow{b}}$ and $\kappa^{\vec{b}} = \kappa^{\overleftarrow{b}}$.

In this case (which we call symmetric), in the bi-directional labeling algorithm, we set $q^* = \frac{1}{2}l_{0,r^*} + \frac{1}{2}u_{n+1,r^*}$, skip the backward labeling step, and use symmetric forward buckets and labels instead of backward buckets and labels in the concatenation step.

Algorithm 2: Bi-directional labeling algorithm

run forward labeling algorithm where only labels $\vec{L}$, $q^{\vec{L}}_{r^*} \leq q^*$ are kept
run backward labeling algorithm where only labels $\overleftarrow{L}$, $q^{\overleftarrow{L}}_{r^*} > q^*$, are kept
let P^{best} be the best complete path obtained in the two algorithms above
foreach *forward label* $\vec{L}$ **do**
  foreach *bucket arc* $\vec{\gamma} \in \vec{\Gamma}$ *such that* $\vec{b}_{\vec{\gamma}} = \vec{b}^{\vec{L}}$ **do**
    if Extend$(\vec{L}, \vec{\gamma}, \vec{L}')$ **and** $q^{\vec{L}'}_{r^*} > q^*$ **then**
      let $\overleftarrow{b} \in \overleftarrow{B}$ be the bucket such that $\tilde{u}_{\overleftarrow{b}} \geq q^{\vec{L}'} > \tilde{u}_{\overleftarrow{b}} - \tilde{d}$
      ConcatenateLabel$(\vec{L}, \overleftarrow{b}, P^{best}, \emptyset)$
return P^{best}

Procedure ConcatenateLabel$(\vec{L}, \overleftarrow{b}, P^{best}, \overleftarrow{B}^{visited})$

$\overleftarrow{B}^{visited} \leftarrow \overleftarrow{B}^{visited} \cup \{\overleftarrow{b}\}$
if $\bar{c}_{\vec{L}} + \bar{c}_{(v^{\vec{L}}, \tilde{v}_{\overleftarrow{b}})} + \bar{c}^{best}_{\overleftarrow{b}} \geq \bar{c}^{P^{best}}$ **then return**
foreach *label* $\overleftarrow{L} : b_{\overleftarrow{L}} = \overleftarrow{b}$ **do**
  if *pair* $(\vec{L}, \overleftarrow{L})$ *is* $\bar{c}^{P^{best}}$*-compatible* **then**
    $P^{best} \leftarrow \left(P_{\vec{L}},\ \vec{a} = (v^{\vec{L}}, v^{\overleftarrow{L}}),\ P_{\overleftarrow{L}}\right)$
foreach $\overleftarrow{b}' \in \Phi_{\overleftarrow{b}} \setminus \overleftarrow{B}^{visited}$ **do**
  ConcatenateLabel$(\vec{L}, \overleftarrow{b}', P^{best}, \overleftarrow{B}^{visited})$
return

4 Bucket arc elimination

Arc elimination procedure by Irnich et al. (2010) can be employed to remove arcs from the original graph using the argument that any path using one of these arcs is not contributing to an optimal solution. In this section, we generalize this procedure to eliminate bucket arcs from the bucket graph using the same argument. However, this generalization is not straightforward because of the dominance between labels. Even labels from the same bucket may use different bucket arcs when extended through the same path in the original graph. Hence, the same path extension may be feasible for the dominated label and not feasible for the dominating label due to removal of some bucket arcs. We introduce below the new concept of jump bucket arcs to bring back compatibility between bucket arc elimination and dominance.

We call a path $P \in \Omega_m$, $m \in M$, *improving* if λ_P participates in an integer solution of the original problem (SPF) with a cost smaller than the cost of the best solution found so far. We denote the latter cost as UB, as it is an upper bound on the optimal solution value of (SPF). Let Ω^I_m be the set of improving paths in Ω_m. We can exclude paths proved to be non-improving from the solution space of the pricing problem. Note that if there are no improving paths (i.e., the best known solution is optimal) any path can be correctly excluded, so we only need to be careful in the case where some Ω^I_m sets are not empty. So, we assume that case in the remainder of the section.

Suppose that some variables λ_P, $P \notin \Omega^I_m$, are fixed to zero in (SPF). Let $(\bar{\pi}, \bar{\mu}, \bar{\nu}, \bar{\sigma})$ be the dual solution of the current restricted linear relaxation of (SPF), corresponding to constraints (2), (3) and to constraints of type (5) and (7). Let $z^{(\mathrm{RSPF})}(\bar{\pi}, \bar{\mu}, \bar{\nu}, \bar{\sigma})$ be the value of this solution. The value

$$LB^{\mathrm{Lagr}} = z^{(\mathrm{RSPF})}(\bar{\pi}, \bar{\mu}, \bar{\nu}, \bar{\sigma}) + \sum_{m \in M} \min\left\{0, \min_{P \in \Omega^I_m} \bar{c}^P(\bar{\pi}, \bar{\mu}, \bar{\nu}, \bar{\sigma})\right\} \cdot U_m$$

is a Lagrangian lower bound on the optimal solution value of (SPF). It is obtained by dualizing constraints (2), (5) and (7) with multipliers $\bar{\pi}$, $\bar{\nu}$, and $\bar{\sigma}$. For any vector $(z^1, z^2, \ldots, z^{|M|})$ such

that

$$z^m \leq \min_{P \in \Omega^I_m} \bar{c}^P(\bar{\pi}, \bar{\mu}, \bar{\nu}, \bar{\sigma}), \quad m \in M, \tag{14}$$

we have

$$LB = z^{(\mathrm{RSPF})}(\bar{\pi}, \bar{\mu}, \bar{\nu}, \bar{\sigma}) + \sum_{m \in M} \min\{0, z^m\} \cdot U_m \leq LB^{\mathrm{Lagr}}. \tag{15}$$

Thus, LB is a valid lower bound on the optimum solution of (SPF). If an algorithm for solving pricing sub-problem (PSP^m) returns value z^m satisfying condition (14), we say that this algorithm satisfies (14). Such an algorithm can be used to calculate a valid lower bound for (SPF) even if it does not necessarily solve (PSP^m) to optimality. In the remainder of this section, we again drop index m and consider dual solution $(\bar{\pi}, \bar{\mu}, \bar{\nu}, \bar{\sigma})$ fixed for more clarity.

We say that a feasible forward/backward path $P \in \mathcal{P}$ *passes through* bucket arc $\gamma \in \Gamma$ if there is an index j, $1 \leq j \leq |P|$, such that $a^P_j = a_\gamma$, and $\tilde{l}_{b_\gamma} \leq q^P_{j-1} < \tilde{l}_{b_\gamma} + \tilde{d}$ for a forward path ($\tilde{u}_{b_\gamma} \geq q^P_{j-1} > \tilde{u}_{b_\gamma} - \tilde{d}$ for a backward path). We denote such a bucket arc as γ^P_j.

We call a forward/backward $\gamma \in \Gamma$ an *improving bucket arc* if there exists an improving path of the same sense passing through it. The bucket arc elimination procedure described in Section 4.2 reduces the current set Γ by removing some non-improving bucket arcs from it. However, the labeling algorithm proposed in Section 3, in which labels are only extended over a reduced set Γ of bucket arcs does not necessarily satisfy condition (14).

Figure 4 illustrates a case where the value z of the best solution found by the labeling algorithm does not satisfy (14). In this figure, the value of a single resource consumption $r = 1$ for each partial path is represented by the level of its end node in the vertical axis. P and P' are improving and non-improving forward paths, respectively. Thus, for each traversed arc, the corresponding bucket arc that is passed through depends solely on the total resource consumed up to this point. Paths P and P' traverse different arcs until vertex $v^P_{i-1} = v^{P'}_{i'-1}$ and finish with the same subpaths from this point until the sink vertex (but with such subpaths passing through different bucket arcs).

Assume that the reduced bucket graph does not contain the non-improving bucket arc γ corresponding to $(v^{P'}_{i'-1}, v^{P'}_{i'})$. Suppose also that P' has a reduced cost smaller than that of P, allowing the label L associated to P' until vertex $v^{P'}_{i'-1}$ to dominate the label $\bar{L}$ associated to P until the same vertex. The reasoning for such domination is that one can always replace the partial path of $\bar{L}$ by that of L, causing that P is replaced by P' which has a smaller reduced cost. However, absence of bucket arc γ in the reduced bucket graph prevents P' from being found by the labeling algorithm. Thus, neither P nor P' are obtained by the labeling algorithm, and the value z found by the algorithm may violate condition (14).

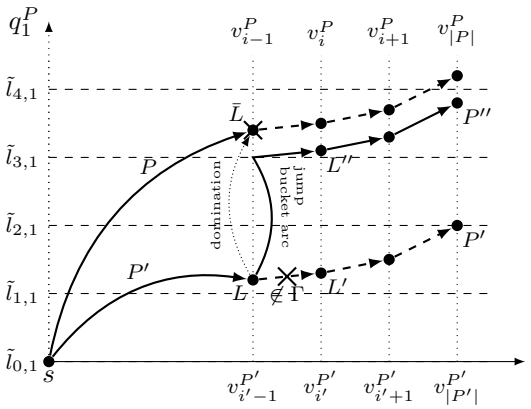

Figure 4: Illustration of the proof of Proposition 1

The remainder of this section is divided in two parts. In Section 4.1, we show how to modify our labeling algorithm so that it satisfies condition (14) when executed over a superset of all

improving bucket arcs. In Section 4.2, we present a procedure for finding non-improving bucket arcs, and show the correcteness of its combination with the modified labeling algorithm.

4.1 Modified labeling algorithm

To guarantee the fulfillment of condition (14), labels in our forward (backward) labeling algorithm should not only be extended along the current reduced set $\vec{\Gamma}$ ($\overleftarrow{\Gamma}$) of bucket arcs, but also along the set $\vec{\Psi}$ ($\overleftarrow{\Psi}$) of *jump bucket arcs* obtained as shown below. Jump bucket arcs impose an additional resource consumption when traversed. Each forward/backward jump bucket arc $\psi \in \Psi$ is characterized by a triple $(b_\psi^{\text{base}}, b_\psi^{\text{jump}}, a_\psi)$, where b_ψ^{base} is its base bucket, b_ψ^{jump} is its jump bucket such that $\tilde{v}_{b_\psi^{\text{base}}} = \tilde{v}_{b_\psi^{\text{jump}}} = \tilde{v}$, $b_\psi^{\text{base}} \prec b_\psi^{\text{jump}}$, and $a_\psi = (\tilde{v}, v') \in A$ is its arc of the corresponding sense. In the labeling algorithm, each label L should be extended along a jump bucket arc $\psi \in \Psi$ if $b^L = b_\psi^{\text{base}}$. When extending label L' along $\psi \in \Psi$, we first do the "jump", i.e. in the second line of function Extend(L',ψ,L), instead of $q^L \leftarrow q^{L'}$, we do $q^L \leftarrow \max\left\{q^{L'}, \tilde{l}_{b_\psi^{\text{jump}}}\right\}$ ($q^L \leftarrow \min\left\{q^{L'}, \tilde{u}_{b_\psi^{\text{jump}}}\right\}$ for a backward label). Function ObtainJumpBucketArcs(Γ) presented below obtains the current set Ψ of jump bucket arcs given the current reduced set Γ of bucket arcs. This function can be used for both forward and backward sense.

Function ObtainJumpBucketArcs(Γ)

```
Ψ ← ∅
foreach bucket b ∈ B do
    foreach a = (v, v') ∈ A such that v = ṽ_b do
        if ∄γ ∈ Γ such that b_γ = b, a_γ = a then
            B̄ ← {b' ≻ b : ∃γ ∈ Γ : b_γ = b', a_γ = a}
            remove from B̄ all buckets which are not component-wise minimal
            foreach b' ∈ B̄ do
                Ψ ← Ψ ∪ {ψ}, where b_ψ^base = b, b_ψ^jump = b', a_ψ = a
return Ψ
```

We denote as L_j^P the label corresponding to the partial path consisting of the first j arcs of path P. If P passes through γ, then there is an index j, $1 \leq j \leq |P|$, such that label L_j^P is contained in bucket b_γ. Let $\vec{\mathbf{L}}(\Gamma, \Psi)$ and $\overleftarrow{\mathbf{L}}(\Gamma, \Psi)$ be the sets of non-dominated labels generated by the modified forward and backward mono-directional labeling algorithms, in which each label L is extended along all bucket arcs $\gamma \in \Gamma$ such that $b_\gamma = b^L$ and along all jump bucket arcs $\psi \in \Psi$ such that $b_\psi^{\text{base}} = b^L$.

The next proposition is auxiliary but very important. It shows that for every improving path P, the modified mono-directional labeling algorithm either generates P or a path dominating P. Moreover, this is true for any prefix of P. This proposition applies for both forward and backward sense. Its proof is illustrated in Figure 4.

Proposition 1. *Let set Γ be a superset of all improving bucket arcs, and Ψ be the set of jump bucket arcs computed using procedure* ObtainJumpBucketArcs(Γ). *Then, for each improving path $P \in \mathcal{P}$ and for each index j, $0 \leq j \leq |P|$, either $L_j^P \in \mathbf{L}$ or there exists a label $L \in \mathbf{L}$ dominating L_j^P, where $\mathbf{L} = \mathbf{L}(\Gamma, \Psi)$.*

Proof. Consider an improving path $P \in \mathcal{P}$. We prove this proposition by induction on the value of the index j, $0 \leq j \leq |P|$. For $j = 0$ the proposition is true since there is a single label L^{init} in the bucket $b^{L^{init}}$ that corresponds to vertex v_0^P. Now, assume that the proposition is true for $j = i-1$, $1 \leq i \leq |P|$. We must prove that it is also valid for $j = i$. By the inductive hypothesis, either $L_{i-1}^P \in \mathbf{L}$ or there exists a label $L \in \mathbf{L}$ dominating L_{i-1}^P. Let us use the notation L defined for the latter case to also denote L_{i-1}^P in the former case. In all cases, $b^L \preceq b^{L_{i-1}^P}$. Let

γ' and γ be the bucket arcs such that $b_{\gamma'} = b^L$, $b_\gamma = b^{L_{i-1}^P}$, and $a_{\gamma'} = a_\gamma = a_i^P$. If $\gamma' \in \Gamma$, the label L' obtained by extending L through $a_{\gamma'}$ dominates L_i^P. In this case, either $L' \in \mathbf{L}$ or L' is dominated by some label from $\mathbf{L}$, which should also dominate L_i^P. In both situations, the proof is finished. Otherwise, since $\gamma' \notin \Gamma$, γ is an improving arc, and $b^L \prec b_\gamma$, there exists a jump bucket arc $\psi \in \Psi$ such that $a_\psi = a_i^P$, $b_\psi^{\text{base}} = b^L$, and $b_\psi^{\text{jump}} \preceq b^{L_i^P}$. Existence of such a jump bucket arcs is guaranteed by the definition of procedure ObtainJumpBucketArcs(Γ). Thus, extending L through a_ψ leads to a label L'' that dominates L_i^P. As before, either $L'' \in \mathbf{L}$ or L'' is dominated by some label from $\mathbf{L}$, which dominates L_i^P, finishing the proof. □

We have just shown essentially that the modified mono-directional labeling algorithm satisfies condition (14). In Proposition 3, we show that the same condition is satisfied by the modified bi-directional labeling algorithm. First observe that any complete path can be divided into forward and backward parts by removing any arc it passes through. Moreover, the total reduced cost of the concatenation of these parts is equal to the reduced cost of the complete path.

Observation 1. *For a pair of corresponding forward/backward path P and the opposite sense path $\overleftarrow{P}$ and each value j, $0 \le j < |P|$, pair of labels L_j^P and $\overleftarrow{L}^{\overleftarrow{P}}_{|P|-j-1}$ is ω-compatible for all $\omega > \bar{c}^P$.*

Secondly, we will need the following auxiliary proposition.

Proposition 2. *Suppose that i) forward and backward labels $\vec{L}'$ and $\overleftarrow{L}'$ are ω-compatible, ii) either $\vec{L} = \vec{L}'$ or label $\vec{L}$ dominates $\vec{L}'$, iii) either $\overleftarrow{L} = \overleftarrow{L}'$ or label $\overleftarrow{L}$ dominates $\overleftarrow{L}'$. Then labels $\vec{L}$ and $\overleftarrow{L}$ are also ω-compatible.*

Proof. Let $a = (v^{\vec{L}}, v^{\overleftarrow{L}}) = (v^{\vec{L}'}, v^{\overleftarrow{L}'})$. From the definition of dominance in Section 3.2 and the definition of ω-compatible labels in Section 3.5, we have $v^{\vec{L}} = v^{\vec{L}'} \neq v^{\overleftarrow{L}'} = v^{\overleftarrow{L}}$, $q^{\vec{L}} + d_a \le q^{\vec{L}'} + d_a \le q^{\overleftarrow{L}'} \le q^{\overleftarrow{L}}$. We also have $\mathcal{F}^{\vec{L}} \subseteq \mathcal{F}^{\vec{L}'}$, $\mathcal{F}^{\overleftarrow{L}} \subseteq \mathcal{F}^{\overleftarrow{L}'}$, and $\mathcal{F}^{\vec{L}'} \cap \mathcal{F}^{\overleftarrow{L}'} = \emptyset$. It follows then that $\mathcal{F}^{\vec{L}} \cap \mathcal{F}^{\overleftarrow{L}} = \emptyset$. From (12) and the definition of $\bar{c}(P^{\vec{L}}||P^{\overleftarrow{L}})$ it follows

$$\bar{c}(P^{\vec{L}}||P^{\overleftarrow{L}}) = \bar{c}^{\vec{L}} + \bar{c}_a + \bar{c}^{\overleftarrow{L}} - \sum_{\substack{\ell\in\mathcal{L}:\\ \mathcal{S}_\ell^{\vec{L}}+\mathcal{S}_\ell^{\overleftarrow{L}}\ge 1}} \sigma_\ell \le \underbrace{\bar{c}^{\vec{L}'} + \sum_{\substack{\ell\in\mathcal{L}:\\ \mathcal{S}_\ell^{\vec{L}}>\mathcal{S}_\ell^{\vec{L}'}}} \sigma_\ell}_{\ge \bar{c}^{\vec{L}} \text{ as } \vec{L} \text{ dominates } \vec{L}'} + \bar{c}_a + \underbrace{\bar{c}^{\overleftarrow{L}'} + \sum_{\substack{\ell\in\mathcal{L}:\\ \mathcal{S}_\ell^{\overleftarrow{L}}>\mathcal{S}_\ell^{\overleftarrow{L}'}}} \sigma_\ell}_{\ge \bar{c}^{\overleftarrow{L}} \text{ as } \overleftarrow{L} \text{ dominates } \overleftarrow{L}'} - \sum_{\substack{\ell\in\mathcal{L}:\\ \mathcal{S}_\ell^{\vec{L}}+\mathcal{S}_\ell^{\overleftarrow{L}}\ge 1}} \sigma_\ell$$

$$= \underbrace{\bar{c}^{\vec{L}'} + \bar{c}_a + \bar{c}^{\overleftarrow{L}'} - \sum_{\substack{\ell\in\mathcal{L}:\\ \mathcal{S}_\ell^{\vec{L}'}+\mathcal{S}_\ell^{\overleftarrow{L}'}\ge 1}} \sigma_\ell}_{=\bar{c}(P^{\vec{L}'}||P^{\overleftarrow{L}'})<\omega} + \underbrace{\sum_{\substack{\ell\in\mathcal{L}:\\ \mathcal{S}_\ell^{\vec{L}'}+\mathcal{S}_\ell^{\overleftarrow{L}'}\ge 1,\\ \mathcal{S}_\ell^{\vec{L}}+\mathcal{S}_\ell^{\overleftarrow{L}}<1}} \sigma_\ell}_{\le 0 \text{ as } \sigma\le 0} + \underbrace{\sum_{\substack{\ell\in\mathcal{L}:\\ \mathcal{S}_\ell^{\vec{L}}>\mathcal{S}_\ell^{\vec{L}'}}} \sigma_\ell + \sum_{\substack{\ell\in\mathcal{L}:\\ \mathcal{S}_\ell^{\overleftarrow{L}}>\mathcal{S}_\ell^{\overleftarrow{L}'}}} \sigma_\ell - \sum_{\substack{\ell\in\mathcal{L}:\\ \mathcal{S}_\ell^{\vec{L}'}+\mathcal{S}_\ell^{\overleftarrow{L}'}<1,\\ \mathcal{S}_\ell^{\vec{L}}+\mathcal{S}_\ell^{\overleftarrow{L}}\ge 1}} \sigma_\ell}_{\le 0 \text{ as } \sigma\le 0}$$

It follows then that $\bar{c}(P^{\vec{L}}||P^{\overleftarrow{L}}) < \omega$. Hence, all conditions for $\vec{L}$ and $\overleftarrow{L}$ to be ω-compatible are met. □

We define as $z(\vec{\Gamma}, \overleftarrow{\Gamma}, \vec{\Psi}, \overleftarrow{\Psi})$ the reduced cost of the best path obtained by the modified bi-directional labeling algorithm in which forward labels are extended along bucket arcs in $\vec{\Gamma}$ and $\vec{\Psi}$, and backward labels are extended along bucket arcs in $\overleftarrow{\Gamma}$ and $\overleftarrow{\Psi}$.

Proposition 3. *Let sets $\vec{\Gamma}$ and $\overleftarrow{\Gamma}$ be supersets of all improving forward and backward bucket arcs, and $\vec{\Psi}$ and $\overleftarrow{\Psi}$ be the sets of forward and backward jump bucket arcs computed using procedure* ObtainJumpBucketArcs(Γ). *Then $z \le c^{\vec{P}}$ for any improving path $\vec{P} \in \vec{\mathcal{P}}$, where $z = z(\vec{\Gamma}, \overleftarrow{\Gamma}, \vec{\Psi}, \overleftarrow{\Psi})$.*

Proof. Consider an arbitrary improving path $\vec{P} \in \vec{\mathcal{P}}$. Let j, $0 \le j \le |\vec{P}|$ be the largest index such that $q_{r^*}^{\vec{L}_j^{\vec{P}}} \le q^*$. We divide the proof into two cases: $j = |\vec{P}|$ and $j < |\vec{P}|$.

For $j = |\vec{P}|$, let $\vec{L} = \vec{L}_j^{\vec{P}}$. We have $v^{\vec{L}} = n+1$. As $\vec{P}$ is improving, by Proposition 1, either $\vec{L} \in \vec{\mathbf{L}}(\vec{\Gamma}, \vec{\Psi})$ or there is a label $\vec{L}' \in \vec{\mathbf{L}}(\vec{\Gamma}, \vec{\Psi})$ dominating $\vec{L}$. As $v^{\vec{L}'} = n+1$, path $\vec{P}^{\vec{L}'} \in \vec{\mathcal{P}}$ and the total reduced cost of $\vec{P}^{\vec{L}'}$ is $\leq \bar{c}^{\vec{P}}$, thus $z \leq \bar{c}^{\vec{P}}$.

Suppose now that $j < |\vec{P}|$. Let $\overleftarrow{P}$ be the corresponding backward path for $\vec{P}$. By Observation 1, the pair of labels $\vec{L}' = \vec{L}_j^{\vec{P}}$ and $\overleftarrow{L}' = \overleftarrow{L}^{\overleftarrow{P}}_{|\overleftarrow{P}|-j-1}$ is ω-compatible for all $\omega > \bar{c}^{\vec{P}}$. As both $\vec{P}$ and $\overleftarrow{P}$ are improving, by Proposition 1 either $\vec{L}' \in \vec{\mathbf{L}}(\vec{\Gamma}, \vec{\Psi})$ or there exists a label $\vec{L} \in \vec{\mathbf{L}}(\vec{\Gamma}, \vec{\Psi})$ dominating $\vec{L}'$. Also, by Proposition 1 either $\overleftarrow{L}' \in \overleftarrow{\mathbf{L}}(\overleftarrow{\Gamma}, \overleftarrow{\Psi})$ or there exists a label $\overleftarrow{L} \in \vec{\mathbf{L}}(\overleftarrow{\Gamma}, \overleftarrow{\Psi})$ dominating $\overleftarrow{L}'$. Thus, by Proposition 2 there exists a pair of ω-compatible labels generated by the modified labeling algorithm for all $\omega > \bar{c}^{\vec{P}}$. Then $\bar{c}(\vec{P}^{\vec{L}} || \overleftarrow{P}^{\overleftarrow{L}}) \leq \bar{c}^{\vec{P}}$, thus $z \leq \bar{c}^{\vec{P}}$. □

We have just shown that our labelling algorithm can be modified in order to satisfy condition (14) by adding jump bucket arcs. Now it can be executed over any superset Γ of improving bucket arcs. In the next section, we show how we can find non-improving bucket arcs, which then can be safely removed from the current set Γ.

4.2 A procedure for finding non-improving bucket arcs

In this paragraph, we temporarily consider again all subproblems (PSP^m), $m \in M$. Let $(\bar{\pi}, \bar{\mu}, \bar{\nu}, \bar{\sigma})$ be the current dual solution of the restricted linear relaxation of (SPF) together with constraints (5) and (7). Suppose that the current sets Γ^m, $m \in M$, include all improving buckets arcs, and the modified labelling algorithms have returned values z^m for all $m \in M$. By Proposition 3, the modified labeling algorithm satisfies condition (14). Thus the value LB given by (15) is a valid lower bound for (SPF). A necessary condition for a path $P \in \Omega_m$ to be improving is $LB - \min\{0, z^m\} + \bar{c}^P(\bar{\pi}, \bar{\mu}, \bar{\nu}, \bar{\sigma}) < UB$. We denote

$$\theta^m = UB - LB + \min\{0, z^m\}. \tag{16}$$

Then, a sufficient condition for a forward/backward path $P \in \Omega_m$ to be non-improving is $\bar{c}^P(\bar{\pi}, \bar{\mu}, \bar{\nu}, \bar{\sigma}) \geq \theta^m$. The next proposition gives a necessary condition for a bucket arc to be improving. The proposition applies for both forward and backward sense. In the following, we drop again index m.

For a (jump) bucket arc $\gamma \in \Gamma$ ($\psi \in \Psi$), we denote as $\vec{b}_\gamma^{\text{arr}}$ ($\vec{b}_\psi^{\text{arr}}$) its *arrival bucket* of the opposite sense. For a forward bucket arc $\vec{\gamma} \in \vec{\Gamma}$, its arrival bucket $\overleftarrow{b}_{\vec{\gamma}}^{\text{arr}}$ is the backward bucket $\overleftarrow{b}$ such that $\tilde{u}_{\overleftarrow{b}} \geq \max\{\tilde{l}_{\vec{b}_{\vec{\gamma}}} + d_{\vec{a}_{\vec{\gamma}}}, l_{v'}\} > \tilde{u}_{\overleftarrow{b}} - \tilde{d}$, where v' is the head node of $\vec{a}_{\vec{\gamma}}$. For a backward bucket arc $\overleftarrow{\gamma} \in \overleftarrow{\Gamma}$, its arrival bucket $\vec{b}_{\overleftarrow{\gamma}}^{\text{arr}}$ is the forward bucket $\vec{b}$ such that $\tilde{l}_{\vec{b}} \leq \min\{\tilde{u}_{\overleftarrow{b}_{\overleftarrow{\gamma}}} - d_{\overleftarrow{a}_{\overleftarrow{\gamma}}}, u_{v'}\} < \tilde{l}_{\vec{b}} + \tilde{d}$, where v' is the head node of $\overleftarrow{a}_{\overleftarrow{\gamma}}$. For a jump bucket arc $\psi \in \Psi$, the definition of its forward/backward arrival buckets is identical, just replacing b_γ by b_ψ^{jump}.

Proposition 4. *Let set Γ be a superset of all improving bucket arcs, Ψ be the set of jump bucket arcs computed using procedure* ObtainJumpBucketArcs(Γ)*, and value θ be calculated according to (16). If $\gamma \in \Gamma$ is an improving bucket arc then there exists a pair of θ-compatible labels $L \in \mathbf{L}$ and $\overleftarrow{L} \in \overleftarrow{\mathbf{L}}$ such that $b^L \preceq b_\gamma$ and $\overleftarrow{b}^{\overleftarrow{L}} \preceq \overleftarrow{b}_\gamma^{\text{arr}}$, where $\mathbf{L} = \mathbf{L}(\Gamma, \Psi)$, and $\overleftarrow{\mathbf{L}} = \overleftarrow{\mathbf{L}}(\Gamma, \Psi)$.*

Proof. If γ is an improving bucket arc, then there exists an improving path P and an index j, $0 \leq j < |P|$, such that $b_\gamma = b^{L_j^P}$ and $a_\gamma = a_{j+1}^P$. We have $\bar{c}^P < \theta$, as it is a necessary condition for P to be improving. Then by Observation 1, labels L_j^P and $\overleftarrow{L}^{\vec{P}}_{|P|-j-1}$ are θ-compatible. If label $L_j^P \in \mathbf{L}$, let $L = L_j^P$, otherwise by Proposition 1 there exists a label $L \in \mathbf{L}$ dominating L_j^P such that $b^L \preceq b^{L_j^P} = b_\gamma$. If label $\overleftarrow{L}^{\vec{P}}_{|P|-j-1} \in \overleftarrow{\mathbf{L}}$, let $\overleftarrow{L} = \overleftarrow{L}^{\vec{P}}_{|P|-j-1}$, otherwise by Proposition 1 there exists a label $\overleftarrow{L} \in \overleftarrow{\mathbf{L}}$ dominating $\overleftarrow{L}^{\vec{P}}_{|P|-j-1}$ such that $\overleftarrow{b}^{\overleftarrow{L}} \preceq \overleftarrow{b}^{\overleftarrow{L}^{\vec{P}}_{|P|-j-1}} \preceq \overleftarrow{b}_\gamma^{\text{arr}}$. Finally, by Proposition 2, L and $\overleftarrow{L}$ are θ-compatible. □

Proposition 4 is the basis for the procedure to find non-improving bucket arcs. To prove that a bucket arc γ is non-improving, we need to verify that there does not exist a pair of θ-compatible labels $L \in \mathbf{L}$ and $\overleftarrow{L} \in \overleftarrow{\mathbf{L}}$ (one at the tail and one at the head of a_γ) such that $b^L \preceq b_\gamma$ and $\overleftarrow{b}^{\overleftarrow{L}} \preceq \overleftarrow{b}_\gamma^{\text{arr}}$. A direct approach is to exhaustively enumerate all such pairs of labels for every bucket arc and verify whether at least one of them is θ-compatible. This approach is slow as the same pair of labels may be verified for θ-compatibility several times.

To guarantee that no pair of labels is verified for θ-compatibility more than once, we compute sets $\overleftarrow{B}_{a,b}$ of buckets of opposite sense with respect to bucket b at the head of arc a. $\overleftarrow{B}_{a,b}$ contains every bucket $\overleftarrow{b}$ containing a label $\overleftarrow{L} \in \overleftarrow{\mathbf{L}}$ which is θ-compatible with a label $L \in \mathbf{L}$ such that $b^L \preceq b$. Thus, a bucket arc γ is non-improving if $\overleftarrow{B}_{a_\gamma,b_\gamma}$ does not contain any bucket $\overleftarrow{b} \preceq \overleftarrow{b}_\gamma^{\text{arr}}$. An advantage of using sets $\overleftarrow{B}_{a,b}$ is that they can be constructed incrementally using adjacent buckets: every set $\overleftarrow{B}_{a,b}$ is initialized as the union of sets $\overleftarrow{B}_{a,b'}$, $b' \in \Phi_b$, and then augmented using the recursive procedure UpdateBucketsSet which tries to find opposite sense buckets containing a label which is θ-compatible with a label in b.

We now present formally the auxiliary procedure UpdateBucketsSet(θ, L, $\overleftarrow{B}$, $\overleftarrow{b}$, $\overleftarrow{\mathbf{L}}$, $\overleftarrow{B}^{visited}$), which adds to set $\overleftarrow{B}$ all opposite sense buckets $\overleftarrow{b}'$, $\overleftarrow{b}' \preceq \overleftarrow{b}$, $\overleftarrow{b}' \notin \overleftarrow{B}^{visited}$, containing at least one label $\overleftarrow{L} \in \overleftarrow{\mathbf{L}}$ such that pair $(L, \overleftarrow{L})$ of labels is θ-compatible.

Procedure UpdateBucketsSet(θ, L, $\overleftarrow{B}$, $\overleftarrow{b}$, $\overleftarrow{\mathbf{L}}$, $\overleftarrow{B}^{visited}$)

$\overleftarrow{B}^{visited} \leftarrow \overleftarrow{B}^{visited} \cup \{\overleftarrow{b}\}$
if $\bar{c}^L + \bar{c}_{v^L,\overleftarrow{v}_{\overleftarrow{b}}} + \bar{c}_{\overleftarrow{b}}^{best} \geq \theta$ **then return**
if $\overleftarrow{b} \notin \overleftarrow{B}$ **then**
  foreach *label* $\overleftarrow{L} \in \overleftarrow{\mathbf{L}}$, $\overleftarrow{b}^{\overleftarrow{L}} = \overleftarrow{b}$ **do**
    if *pair* $(L, \overleftarrow{L})$ *is* θ*-compatible* **then**
      $\overleftarrow{B} \leftarrow \overleftarrow{B} \cup \{\overleftarrow{b}\}$
foreach $\overleftarrow{b}' \in \Phi_{\overleftarrow{b}} \setminus \overleftarrow{B}^{visited}$ **do**
  UpdateBucketsSet($\theta, L, \overleftarrow{B}, \overleftarrow{b}', \overleftarrow{\mathbf{L}}, \overleftarrow{B}^{visited}$)

Finally, we present formally the main procedure BucketArcElimination(θ, Γ, Ψ, $\mathbf{L}$, $\overleftarrow{\mathbf{L}}$) which can be used for both forward and backward sense. The procedure removes some non-improving bucket arcs from Γ, given the set $\mathbf{L}$ of non-dominated labels of the same sense as Γ and the set $\overleftarrow{\mathbf{L}}$ of non-dominated labels of the opposite sense. In this procedure, in order to do the correct initialization of sets $\overleftarrow{B}_{a,b}$, buckets b are considered in a lexicographic order of κ^b. When considering bucket b, procedure UpdateBucketsSet is called to augment set $\overleftarrow{B}_{a,b}$ only if there exists $\gamma \in \Gamma$ such that $b = b_\gamma$ or there exists $\psi \in \Psi$ such that $b = b_\psi^{\text{base}}$. If such a (jump) bucket arc does not exist, there is no bucket arc $\gamma' \in \Gamma$ such that $a_{\gamma'} = a$ and $b_{\gamma'} \succeq b$, and thus set $\overleftarrow{B}_{a,b}$ is not used in the reminder of the algorithm.

The next proposition proves formally that the combination of the modified bi-directional labeling algorithm and the bucket arc elimination procedure is correct, i.e. it allows us to compute a valid lower bound for the original problem.

Proposition 5. *Let z be the reduced cost of the best path obtained by the modified bi-directional labeling algorithm after performing j calls to procedure BucketArcElimination, for some integer $j \geq 0$. Then $z \leq c^{\vec{P}}$ for any improving path $\vec{P} \in \vec{\mathcal{P}}$.*

Proof. It is enough to prove that the current forward (backward) bucket graph obtained after j calls of Procedure BucketArcElimination is a superset of all improving forward (backward) bucket arcs, since this proposition is a direct consequence of this statement and Proposition 3. We prove it by induction on the value of j. For $j = 0$, this is true because the initial forward and backward bucket graphs contain all possible forward and backward bucket arcs. Now, suppose by inductive hypothesis that this is valid for $j = i$, i.e. all improving forward

Procedure BucketArcElimination(θ, Γ, Ψ, $\mathbf{L}$, $\overleftarrow{\mathbf{L}}$)

foreach *bucket* $b \in B$ *in a lexicographic order of* κ^b **do**
  if $\Phi_b \neq \emptyset$ **then**
    foreach *arc* $a = (\tilde{v}_b, v') \in A$ **do** $\vec{\boldsymbol{B}}_{a,b} \leftarrow \cup_{b' \in \Phi_b} \vec{\boldsymbol{B}}_{a,b'}$
  else
    foreach *arc* $a = (\tilde{v}_b, v') \in A$ **do** $\vec{\boldsymbol{B}}_{a,b} \leftarrow \emptyset$
  foreach *jump bucket arc* $\psi \in \Psi$ *such that* $b^{\text{base}}_{\psi} = b$ **do**
    foreach $L \in \mathbf{L}$ *such that* $b^L = b$ **do**
      UpdateBucketsSet($\theta, L, \vec{\boldsymbol{B}}_{a_\psi,b}, \vec{b}^{\text{arr}}_{\psi}, \overleftarrow{\mathbf{L}}, \emptyset$)
  foreach *bucket arc* $\gamma \in \Gamma$ *such that* $b_\gamma = b$ **do**
    foreach $L \in \mathbf{L}$ *such that* $b^L = b$ **do**
      UpdateBucketsSet($\theta, L, \vec{\boldsymbol{B}}_{a_\gamma,b}, \vec{b}^{\text{arr}}_{\gamma}, \overleftarrow{\mathbf{L}}, \emptyset$)
    if $\nexists \vec{b}' \in \vec{\boldsymbol{B}}_{a_\gamma,b} : \vec{b}' \preceq \vec{b}^{\text{arr}}_{\gamma}$ **then** $\Gamma \leftarrow \Gamma \setminus \{\gamma\}$

(backward) bucket arcs are present in the current forward (backward) bucket graph before the $(i+1)$th call to procedure BucketArcElimination. Then, by Proposition 4, for each improving (forward or backward) bucket arc $\gamma \in \Gamma$, the bi-directional labeling algorithm generates a pair of θ-compatible labels L and $\overleftarrow{L}$ such that $b^L \preceq b_\gamma$ and $\overleftarrow{b}^{\overleftarrow{L}} \preceq \vec{b}^{\text{arr}}_{\gamma}$. Since this pair of labels prevents γ to be removed in the next call to procedure BucketArcElimination, the statement under consideration is also true for $j = i+1$, finishing the proof. □

Let us now give some final remarks on our bucket arc elimination procedure.

- Our procedure can be viewed as a "resource-value dependent" arc elimination and thus a generalization of the arc elimination procedure by Irnich et al. (2010). The latter can readily be extended to remove extensions of forward labels $\vec{L}$ along an arc if $q^{\vec{L}}$ exceeds a certain threshold (and extensions of backward labels $\overleftarrow{L}$ if $q^{\overleftarrow{L}}$ is lower than a threshold). The opposite case ("less than" for the forward sense and "greater than" for the backward sense) is more complicated as shown in our analysis and necessitates introduction of jump bucket arcs. Our procedure is more generic than both, and may result in elimination of disjoint "resource-value" intervals for an arc.
- Necessity of adding jump bucket arcs essentially comes from the fact that the dominating label may be contained in a different bucket than the dominated one. Restricting dominance checks only to ones within buckets does not make jump bucket arcs redundant. This is because two labels in the same bucket with distinct resource consumption may be extended (over the same arc) to labels in different buckets. Only restricting the dominance checks to labels with the same resource consumption allows one to skip the generation of jump bucket arcs.
- "Resource-value dependent" arc elimination has already been used in Pessoa et al. (2010) and in Pecin et al. (2017b). However, it was done at the cost of resource discretization, limited dominance checks, and dropping the exploitation of the route symmetry. Our procedure allows one to overcome these drawbacks.

5 Branch-Cut-and-Price algorithm

We implemented a BCP algorithm for the HFVRPTW using the new bucket graph labeling algorithm in its pricing and also the bucket arc elimination procedure. Other algorithmic elements are similar to the ones presented in Pecin et al. (2014), Pecin et al. (2017b) and Pecin et al. (2017a), except that a different dynamic *ng*-relaxation by Bulhoes, Sadykov, and Uchoa (2018)

and automatic dual price smoothing stabilization by Pessoa et al. (2018) are used. However, to make this paper self-contained, we give a succinct algorithm description in what follows.

For each node of the branch-and-bound tree, the lower bound on the optimal cost is computed by solving the linear relaxation of SPF enhanced with RCCs (5) and limited-memory R1Cs (7). Before each cut generation round, the current SPF relaxation is solved through column generation, using automatic (parameterless) dual pricing smoothing stabilization described in Pessoa et al. (2018). To further speed up the convergence, three-stage column generation is implemented. At the first stage, the "light" pricing heuristic is applied in which only one label per bucket (with the smallest reduced cost) is kept. In the second stage, a more expensive pricing heuristic is used in which states $\mathcal{F}$ and $\mathcal{S}$ are not taken into account in the dominance checks. In the last stage, the exact labeling algorithm is used. We use parameters χ^{heur} and χ^{exact} for the maximum number of columns generated at each iteration of column generation at heuristic and exact stages. When the number of columns in the restricted master exceeds $\chi^{\max}$, we clean them up and leave only $\chi^{\text{perc}}\%$ columns with the least reduced cost. Basic columns are never removed, however, non-basic columns with zero reduced cost may be removed.

The threshold value q^* for the bi-directional labeling algorithm is an important parameter in the non-symmetric case. Pecin et al. (2017b) and Tilk et al. (2017) proposed to modify the labeling algorithm to automatically adjust this value. Here we adopt a simpler but still effective approach. Value q^* is initialized as the average middle value for all resource consumption intervals. After each exact pricing we compare the number of forward and backward non-dominated labels. If one number exceeds another by more than 20%, we adjust value q^* by moving it by 5% towards the maximum or minimum possible value.

The limited-arc-memory R1Cs are separated by complete enumeration for $|C| \leq 3$ and by a local search heuristic for $|C| \in \{4, 5\}$. This heuristic is launched for each possible optimal vector p. Given p, it tries to find a subset C of customers such that the corresponding cut is violated with a full memory. If such C exists, an arc-memory AM of minimal size such that the cut violation remains the same is identified. Then AM is completed with all arcs between vertices in C. When there already exists a cut defined for the same subset C and vector p but for a different arc-memory AM', this cut is simply enhanced by enlarging its memory to $AM' \cup AM$. Otherwise, a new cut is added for C, p and AM. Also, memory sets are kept symmetric. This means that if arc (v, v') is added to AM, then (v', v) must also be added. Parameter vector β is used in which β_k, $k \in \{0, 1, 3, 4, 5\}$ is the maximum number of R1C with $|C| = k$ separated per round (β_0 is the maximum number of RCCs generated per round). Other parameters are $\delta^{\text{perc}}\%$ and δ^{num} which are used to stop the cut generation by tailing-off. If the number of times the dual-primal gap is decreased by less than $\delta^{\text{perc}}\%$ reaches δ^{num}, then branching is performed. At the root node, R1Cs are separated only if RCCs are exhausted or tail off. Before each cut round, non-active cuts are removed from the restricted master.

In our algorithm, we use the dynamic variant of the *ng*-relaxation from Bulhoes, Sadykov, and Uchoa (2018) which is inspired by Roberti and Mingozzi (2014). We start by generating *ng*-neighborhood for each customer by including their η^{init} closest customers (including itself). After the convergence of column generation, we augment neighborhoods in order to forbid the cycles in the columns forming the current fractional solution. For each such column we either forbid all cycles of size at most η^1 or, if there is no one, the minimum size cycle. Cycles in at most η^2 columns are forbidden in each round. After augmentation, the neighborhood of any customer cannot exceed size $\eta^{\max}$. The same tailing-off parameters are applied here as for cutting planes. We start the root node by augmenting the *ng*-neighborhoods. Then, after tailing-off, RCCs are separated. After another tailing-off, R1Cs are separated.

Each additional active inequality (7) or *ng*-neighborhood increase make the pricing problem harder, since they weaken the dominance conditions. Because of that, separation is stopped when the average exact pricing time since the last round of cuts has exceeded a given threshold τ. In some extreme cases, R1Cs may be even removed by the roll-back procedure activated if a single pricing time exceeds $\bar{\tau}$, as described in Pecin et al. (2017b).

The bucket arc elimination procedure is called after the initial convergence, and also each

time the current primal-dual gap decreased by at least $\upsilon\%$ since the last call to the bucket arc elimination procedure. A bi-directional enumeration procedure is called after each reduced cost fixing to try to generate all improving elementary routes, using the algorithm proposed in Pecin et al. (2017b). The enumeration procedure is interrupted if the number of labels exceeds ω^{labels} or if the number of enumerated routes exceeds ω^{routes}. After a successful enumeration for a pricing subproblem corresponding to a vehicle type, this subproblem passes to the *enumerated state* and the (SPF) relaxation is updated by excluding non-elementary columns coming from this subproblem. The pricing subproblem in the enumerated state is solved by inspection. Elimination procedure based on reduced costs is also performed for enumerated pricing subproblems, but considering each route individually. If after some point, the number of enumerated routes for each subproblem falls below ω^{MIP}, the residual problem is handled by a standard MIP solver, and the node is considered as processed.

Recall that to perform bucket arc elimination, the forward and backward passes of the labeling algorithm must be performed completely. Thus the running time may increase substantially in comparison with the "standard" exact labeling in which the labels beyond the bi-directional threshold value q^* are not stored. To accelerate the bucket arc elimination procedure, we use what we call *exact completion bounds* technique: for every label L just created by extension beyond q^*, we verify exhaustively whether there exists an opposite sense label $\overleftarrow{L}$ such that L and $\overleftarrow{L}$ are θ-compatible. Here θ is as defined in Section 4. We keep label L only if such a label $\overleftarrow{L}$ exists. A similar approach is used in the bi-directional enumeration procedure. Using completion bounds for labels before the bi-directional threshold value q^*, as it is done in Pecin et al. (2017b), was not computationally beneficial for our labeling algorithm.

If tailing-off condition is attained, or at least one non-enumerated pricing subproblem becomes too time consuming to solve, branching must be performed. This is done by adding constraints to the master LP that correspond to tightening lower and upper bounds on the value of the following aggregated variables corresponding to different branching strategies:

- $g^m = \sum_{P\in\Omega_m} \sum_{v'\in V'} \frac{1}{2}\big(x^P_{(0,v')} + x^P_{(v',n+1)}\big)\lambda_P$, $m \in M$;
- $g^m_v = \sum_{P\in\Omega_m} y^P_v \lambda_P$, $v \in V$, $m \in M$;
- $g_{\{v,v'\}} = \sum_{m\in M} \sum_{P\in\Omega_m} (x^P_{(v,v')} + x^P_{(v',v)})\lambda_P$, $v, v' \in V$, $v \neq v'$.

g^m corresponds to the number of used vehicles of type $m \in M$; g^m_v characterizes whether customer $v \in V$ is served by a vehicle of type m; $g_{\{v,v'\}}$ characterizes whether edge (v, v') is used in the solution. Given a fractional branching variable g with fractional value f, two child nodes are created by adding the constraints $g \leq \lfloor f \rfloor$ and $g \geq \lceil f \rceil$ to the master LP. A generic branching constraint $\sum_{m\in M} \sum_{P\in\Omega_m} \sum_{a\in A} \alpha^m_a x^P_a \lambda_P \geq \beta$ added to the master LP with associated dual variable ι can be considered by the pricing algorithm by subtracting $\iota\alpha^m_a$ from the expression of $\bar{c}^m_a(\pi, \nu)$ given by (8), for each $m \in M$ and $a \in A$, thus, not making the subproblem harder. Moreover, the symmetry is preserved, as for any route P, coefficients $x^P_{(v',v)}$ and $x^P_{(v,v')}$ are the same in all branching constraints we use.

The selection of the branching variable at each node is done using a sophisticated hierarchical evaluation strategy similar to the one proposed in Pecin et al. (2017b). The idea is to spend more time evaluating branching variables in the lowest levels of the branch-and-bound tree where each selection has a greater impact on the overall time, and spend less time as the level increases, taking advantage of the history of previous evaluations. The following three evaluation phases are used:

Phase 0: Half of the candidates are chosen from history using pseudo-costs (if history is not empty). The remaining candidates are chosen in a balanced way between the three strategies. Within the same strategy, the candidates are chosen based on the distance from its fractional value to the closest integer and for branching on edges, distance to the closest depot (the smaller the better) is also taken into account.

Phase 1: Evaluate the selected candidates from phase 0 by solving the current restricted master LP modified for each created child node, without generating columns. Select the variables with the maximum value of $\Delta LB_1 \times \Delta LB_2$, where ΔLB_i denotes the increase in the current lower bound obtained for the ith child node, for $i = 1, 2$ (Product Rule, Achterberg (2007)).

Phase 2: Evaluate the selected candidates from phase 1 by solving the relaxation associated to each created child node, including heuristic column generation, but not cut generation. The best candidate is also selected by the Product Rule.

The parameters ζ_0 and ζ_1 are used to specify the maximum number of candidates chosen in phases 0 and 1.

6 Computational experiments

The BCP algorithm described in the previous section was coded in C++ and compiled with GCC 5.3.0. The BaPCod package by Vanderbeck, Sadykov, and Tahiri (2017) was used to handle the BCP framework. IBM CPLEX Optimizer version 12.6.0 was used as the LP solver in column generation and as the IP solver for the set partitioning problem with enumerated columns. The experiments were run on a 2 Deca-core Ivy-Bridge Haswell Intel Xeon E5-2680 v3 server running at 2.50 GHz. The 128 GB of available RAM was shared between eight copies of the algorithm running in parallel on the server. Each instance is solved by one copy of the algorithm using a single thread.

6.1 Instances

VRPTW instances For the vehicle routing problem with time windows, 14 most difficult (according to Pecin et al. (2017a)) instances with 100 customers by Solomon (1987), as well as 120 instances by Gehring and Homberger (2002) with 200 and 400 customers were considered. Solomon instances used are C203, C204, RC204, RC207, RC208, R202, R203, R204, R206, R207, R208, R209, R210, and R211. The instances are divided into classes of clustered (C), random-clustered(RC), and random (R) location of customers on the plane. Note that the capacity resource in Solomon instances, as well as Gehring and Homberger instances of classes C2, RC2, and R2, is not tight. Therefore, the subproblem only considers the time resource in those classes. The capacity resource is imposed through RCCs, which are treated as "core" cuts, i.e. they are separated for every fractional but also for every integer solution found during the search. The initial upper bounds were obtained by the heuristic in Vidal et al. (2013). The same values were used in Pecin et al. (2017a).

DCVRP instances For the distance constrained vehicle routing problem, we used seven classic instances from Christofides, Mingozzi, and Toth (1979) with 50–200 customers, named CMT6, CMT7, CMT8, CMT9, CMT10, CMT13, and CMT14. In spite of the fact that those instances are ubiquitous in the heuristic literature for the CVRP, no exact methods could solve them. The best known solutions for these instances are taken from Nagata and Bräysy (2009).

MDVRP instances For the multi-depot vehicle routing problem, we used 22 standard distance constrained instances by Cordeau, Gendreau, and Laporte (1997) with 80–360 customers, named p08–p11, p13, p14, p15, p17, p19, p20, p22, p23, and pr01–pr10. The best known solutions are taken from Vidal et al. (2012).

SDVRP instances For the site-dependent vehicle routing problem, we used 10 standard distance constrained instances by Cordeau and Laporte (2001) with 48–288 customers, named pr01–pr10. The best known solutions are taken from Cordeau and Maischberger (2012).

Problem	Size	Resources	τ	$\bar{\tau}$	r^*
VRPTW	100-200	1	10	30	time
VRPTW	200	2	5	10	capacity
VRPTW	400	1	20	60	time
VRPTW	400	2	10	20	capacity
HFVRP	any	1	5	10	capacity
DCVRP	any	2	5	10	time
MDVRP, SDVRP	any	2	10	20	time

Table 1: Parameterisation of pricing time thresholds

HFVRP instances For the heterogeneous fleet vehicle routing problem, we used 96 instances by Duhamel, Lacomme, and Prodhon (2011) with 20–256 customers. Each instance represents one of 96 French departments. The best known solutions are taken from Penna et al. (2019). Following the convention used in those papers, we impose that each customer should be visited exactly once. As the triangle inequality is not always satisfied in those instances, allowing multiple visits could possibly reduce costs.

6.2 Algorithm parameterization

The column management parameter values are $\chi^{\text{heur}} = 30$, $\chi^{\text{exact}} = 150$, $\chi^{\max} = 10'000$, $\chi^{\text{perc}} = 66\%$. The cut generation parameter values are $\beta_0 = \beta_1 = \beta_4 = \beta_5 = 100$, $\beta_3 = 150$. The tailing off parameter values are $\delta^{\text{perc}} = 1.5\%$, $\delta^{\text{num}} = 3$. Dynamic ng-relaxation parameter values are $\eta^{\text{init}} = 8$, $\eta^1 = 5$, $\eta^2 = 100$, $\eta^{\max} = 16$. Parameter value for the bucket graph elimination procedure is $\upsilon = 10\%$. Enumeration parameter values are $\omega^{\text{labels}} = 1'000'000$, $\omega^{\text{routes}} = 5'000'000$, $\omega^{\text{MIP}} = 10'000$. The strong branching parameter values are $\zeta_0 = 50$, $\zeta_1 = 3$.

The pricing time thresholds (in seconds) depend on instance class, instance size, and on the number of resources (dimension of buckets). These thresholds as well as which resource is chosen as r^* in bidirectional labeling are shown in Table 1.

The bucket step size in the labeling algorithm is adjusted dynamically as explained below.

6.3 Impact of bucket step size and improved arc elimination

As two main contributions of the paper are using the bucket graph in the labeling algorithm and improved procedure for bucket (or resource value dependent) arc elimination, we first estimate their impact on the solution time of the BCP algorithm.

These tests were done on the set of 74 VRPTW instances with 100 and 200 customers. Only the root node is considered. For instances which cannot be solved at the root, we establish the "target" lower bounds. These values are determined in such a way that once they are achieved by a good parameterisation of the algorithm in the root node, it is reasonable to branch because either i) cuts are tailing off, or ii) pricing becomes too expensive. In Table 2, we give the instance name, its optimum or Best Known Solution (BKS) value, and the Target Lower Bound (TLB) value we use. For instances not shown in this table, the target lower bound equals the optimum solution value.

The first parameter we test is the "maximum number of buckets per vertex" which we denote as ξ. For instances with one time resource, bucket step size $\tilde{d}$ is determined as $(u_{n+1} - l_0)/\xi$. For instances with two resources, bucket step size $\tilde{d}_1$ for the first (capacity) resource is determined as $W/\sqrt{\xi}$, and bucket step size $\tilde{d}_2$ for the second (time) resource is determined as $(u_{n+1} - l_0)/\sqrt{\xi}$. All values $\tilde{d}_r$ are rounded off to one decimal place.

The second binary parameter to test is whether the standard arc elimination procedure (as in Irnich et al. (2010)) or improved bucket arc elimination procedure is applied. In the former case, we fix arc $a \in A$ if all bucket arcs $\gamma \in \Gamma$ such that $a_\gamma = a$ can be fixed. When fixing original arcs, no jump bucket arcs are needed. The procedure for fixing original arcs runs faster, as once

Instance	BKS	TLB	Instance	BKS	TLB	Instance	BKS	TLB
R208	701.0	699.0	R2_2_4	1928.5	1925.0	RC1_2_7	3177.8	3150.0
C2_2_3	1763.4	1753.1	R2_2_6	2675.4	2674.6	RC1_2_8	3060.0	3044.8
C2_2_4	1695.0	1663.5	R2_2_8	1842.4	1831.6	RC1_2_9	3074.8	3035.7
R1_2_3	3373.9	3352.2	R2_210	2549.4	2545.3	RC1_210	2990.5	2966.0
R1_2_4	3047.6	3035.3	RC1_2_1	3516.9	3499.1	RC2_2_2	2481.6	2477.5
R1_2_5	4053.2	4044.1	RC1_2_2	3221.6	3204.4	RC2_2_4	1854.8	1848.0
R1_2_6	3559.2	3544.2	RC1_2_3	3001.4	2980.7	RC2_2_7	2287.7	2279.7
R1_2_8	2938.4	2933.3	RC1_2_4	2845.2	2832.8	RC2_2_8	2151.2	2150.3
R1_2_9	3734.7	3726.0	RC1_2_5	3325.6	3310.0	RC2_2_9	2086.6	2072.0
R1_210	3293.1	3277.3	RC1_2_6	3300.7	3289.0	RC2_210	1989.2	1962.0

Table 2: Target lower bounds for VRPTW instances

we determine that a bucket arc $\gamma \in \Gamma$ cannot be fixed, we do not try to fix bucket arcs $\gamma' \in \Gamma$ such that $a_{\gamma'} = a_\gamma$. The resource value dependent arc elimination procedure proposed in Pessoa et al. (2010) is valid only for the single resource case with positive integer consumption. This procedure can be viewed as a fixing in a bucket graph with unitary step size ($\tilde{d} = 1$) under the condition that the dominance checks are performed only between labels with the same resource consumption.

The base variant of the algorithm which we denote as (O) is parameterized with $\xi = 1$, which corresponds to the basic variant of the label correcting algorithm, and uses the standard arc elimination procedure. The base variant is compared with variant (A) which is parameterized with a fixed value $\xi \in \Xi = \{1, 2, 5, 10, 20, 50, 100, 200, 500, 1000, 2000\}$ and uses the standard arc elimination procedure. Then variant (A) is compared with variant (A^+) which uses a procedure described below to automatically adjust value ξ and the standard arc elimination procedure. Finally variant (A^+) is compared with variant (A^+B) which uses dynamically adjusted value ξ and the improved procedure for bucket arc elimination.

Different bucket step sizes used in the labeling algorithm usually change the set of routes returned by the "light" pricing heuristic to the master. This happens because a given non-dominated label may be extended if one bucket step size is used and not extended if another bucket step size is used. Thus, for each value ξ, the sequence of primal-dual master solutions in the column generation may follow a different trajectory and converge to distinct alternative optima. In turn, this changes the set of cuts returned by the separation procedures, affecting the pricing difficulty for the next iterations. This "random noise" may make the experimental results less clear. To avoid this, for each instance $i \in I$ we choose a value $\bar{\xi}_i \in \Xi$. Then, two copies of the labeling algorithm are created, the first one with value $\bar{\xi}_i$, and the second one with the value being tested $\xi' \in \Xi$. Both copies are executed in each iteration of columns generation, and the arc elimination procedure is also performed twice. Only the set of columns produced by the first copy of the algorithm is sent to the master. Therefore, the sequence of primal-dual solutions of the master problem is always the same regardless of the bucket step size parameterisation.

For each run, i.e. for each pair (i, ξ'), we collect the following statistics. The first one is the *ξ-subproblem time* which includes the time to create the bucket graph and a topological order for it, the total pricing time, the total bucket arc elimination time, and the total enumeration time. The second statistics is the *master time* which is the difference between the overall running time and the sum of $\bar{\xi}_i$-subproblem time and ξ'-subproblem time. The *total time* is the sum of the master time and the ξ'-subproblem time. For each instance $i \in I$ and each value $\xi' \in \Xi$, we calculate ratios $\varrho^{tot}_{i,\xi'}$ and $\varrho^{sp}_{i,\xi'}$ between the corresponding total time (or ξ'-subproblem time) and the minimum total time (minimum ξ-subproblem time) among all $\xi \in \Xi$ for this instance. Ratio $\varrho^{tot}_{i,1}$ shows the speed-up of variant (A) of the algorithm over variant (O) for instance $i \in I$.

In Figure 5, we show the graphs for average and maximum values for both $\varrho^{tot}_{i,\xi'}$ and $\varrho^{sp}_{i,\xi'}$, as well as graphs for these ratios for some representative individual instances. The averages are shown for two groups of instances: with one and two resources. The time limit was reached for $\xi' = 1, 2, 5, 10, 20, 50, 100$ for 6, 6, 4, 4, 3, 1, 1 instances accordingly. Therefore average and maximum values given in Figure 5 for these values of ξ are in fact their lower bounds. It can be seen from the figure that parameter ξ has a large impact on the total pricing time, as well as

on the overall time of the root node. The best value for this parameter is instance-dependent. Two extreme values $\xi = 1$ and $\xi = 2000$ are bad parameterisations on average. The variant (A) with parameterisation $\xi = 200$ is at least 2.85 times faster on average than variant (O). The maximum speed-up reaches at least the factor of 20 (it is expected to be much higher for instances stopped by the time limit).

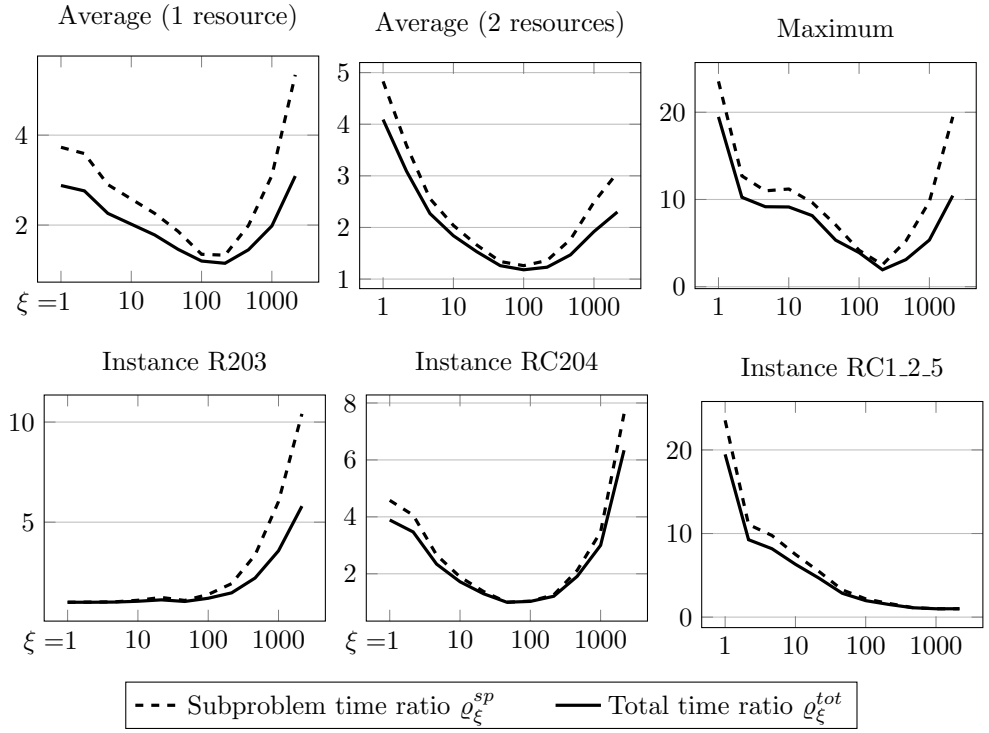

Figure 5: Impact of parameter ξ on the subproblem and total time

In Figure 5, the graph for instance RC_1_2_5 also shows the interest to have smaller buckets than it is required to make the extended bucket graph acyclic. For this instance, setting parameter ξ to 500 makes the extended bucket graph acyclic. However, the subproblem time and the total time of the algorithm are, respectively, 14% and 12% smaller for parameterisation $\xi = 1000$. Most of the time, however, the best parameterisation results in an extended bucket graph with cycles.

In variants (A^+) and (A^+B) we use a scheme for dynamic adjustment of parameter ξ which outperforms the best parameterisation ($\xi = 200$) for variant (A). In this scheme, we start with $\xi = 25$. After each column generation convergence, we calculate the average ratio (among all calls to exact pricing) of the number of dominance checks between labels in a same bucket and the total number of non-dominated labels. If this average ratio is larger than 500 and the average number of non-fixed bucket arcs (including jump bucket arcs) per vertex is less than 10'000, we multiply ξ by 2. After each such multiplication, the bucket graph is regenerated.

In Figure 6(a), we show the performance profile which compares the total time for variant (A) with $\xi = 200$ and for variant (A^+). The geometric mean of the speed-up ratios between these two variants is 7%, and the maximum speed-up reaches 83%.

In Figure 6(b) we show the performance profile which compares variants (A^+) and (A^+B) of the algorithm. Although the geometric mean of the speed-up ratios due to improved bucket arc elimination procedure is only 6%, for two instances the speed-up ratio reaches the factor of two. These are instances RC2_2_8 and R2_2_4. In general, we noticed that one significantly benefits from the improved bucket arc elimination procedure on hard instances with small primal-dual gap. Such instances have usually long routes.

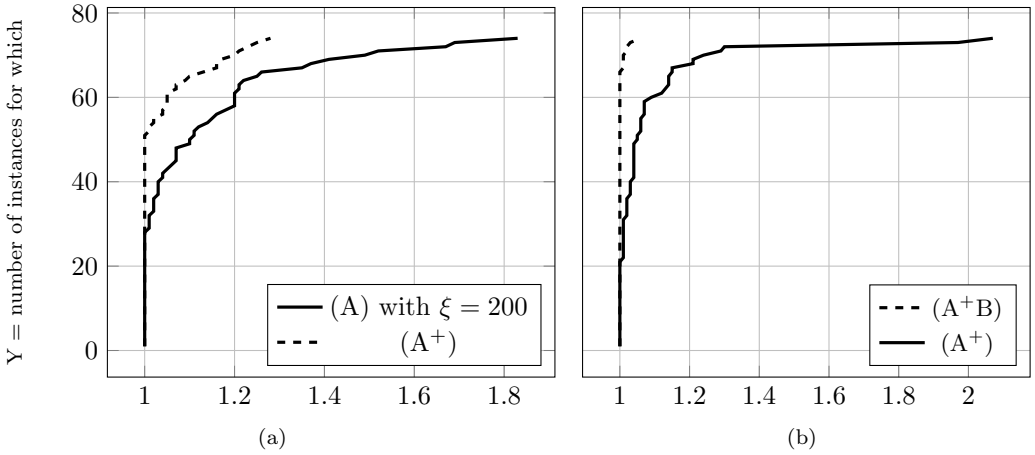

Figure 6: Performance profiles for variants (A), (A^+), and (A^+B) of the algorithm

Concluding this part, we summarize the results in Table 3. In it, we show the geometric means of total solution time ratios for different variants of the algorithm. There, variant (A) uses parameterisation $\xi = 200$. We give these ratios for all 74 instances as well as for subclasses of instances. Looking at Table 3, we can say that the most computational improvement is achieved when passing from variant (O) to variant (A) of the algorithm provided that the bucket step size is well chosen. Additional speed-ups of variants (A^+) and (A^+B) are relatively small on average but can be significant for certain instances.

Instances	(O)→(A)	(A)→(A^+)	(A^+)→(A^+B)
100 customers	1.54	1.18	1.06
200 customers	2.01	1.04	1.07
Series 1	2.10	1.11	1.05
Series 2	1.78	1.03	1.07
Series C	1.20	1.21	1.03
Series R	1.62	0.99	1.08
Series RC	4.15	0.97	1.09
All	1.91	1.07	1.06

Table 3: Comparison of different variants of the algorithm on VRPTW instances

6.4 Impact of two-dimensional buckets

As proposed in Section 3.3, labels are grouped into buckets based on the accumulated resource consumption values. Thus, if there are two resources, we can say that buckets are "two-dimensional". However, buckets can also be formed based only on the accumulated consumption of only one of two resources. Implementation of the labeling algorithm and bucket arc elimination procedure with such "one-dimensional" buckets is easier as sets Φ_b, $b \in B$, contain at most one element. In this section we computationally verify whether an additional effort to implement two-dimensional buckets pays off, i.e. it results in improved algorithm efficiency.

In this experiment, we use variant (A^+B) of the algorithm to solve the root node of instances with two resources. As the number of VRPTW instances with two resources is only 30 (these are instances of classes C1, RC1, and R1), we add in this experiment MDVRP, SDVRP, and DCVRP instances. In total, we have 69 instances. As before, for instances which cannot be solved at the root, we set the "target" lower bounds, given in Table 4. Again, for instances not shown in the table, the target lower bound equals the optimum solution value.

Instance	BKS	TLB	Instance	BKS	TLB	Instance	BKS	TLB
CMT8	865.94	857.8	mdvrp_p10	3631.11	3626.2	sdvrp_pr04	3449.84	3432.3
CMT9	1162.55	1153.3	mdvrp_p11	3546.06	3540.4	sdvrp_pr05	4377.35	4332.6
CMT10	1395.85	1387.3	mdvrp_pr05	2331.20	2318.7	sdvrp_pr06	4422.02	4329.2
CMT13	1541.30	1475.2	mdvrp_pr06	2676.30	2662.9	sdvrp_pr08	2971.01	2962.6
mdvrp_p08	4372.78	4347.6	mdvrp_pr10	2868.26	2860.6	sdvrp_pr09	3536.20	3527.7
mdvrp_p09	3858.66	3849.9	sdvrp_pr03	2575.36	2565.6	sdvrp_pr10	4639.62	4659.4

Table 4: Target lower bounds for MDVRP, SDVRP, and DCVRP instances

As before, when using two-dimensional buckets, given parameter ξ, bucket step size $\tilde{d}_1$ is determined as $W/\sqrt{\xi}$ and $\tilde{d}_2$ is calculated as $(u_{n+1}-l_0)/\sqrt{\xi}$. Naturally, when using one-dimensional buckets, "bucketization" is performed according to the most critical resource. This is the capacity resource for VRPTW instances ($\tilde{d}_1 = W/\xi$) and the time resource for the remaining ones ($\tilde{d}_1 = (u_{n+1}-l_0)/\xi$). The most critical resource can easily be determined by running a reasonable heuristic for the problem and analyzing the solutions obtained. As in the previous section, two copies of the labeling algorithm are created to obtain the same sequence of primal-dual solutions of the master problem for both tested configurations.

In Figure 7 we show the performance profile which compares variants with one-dimensional and two-dimensional buckets. The geometric means of the speed-up ratios between the two variants is 21%. On individual instances the maximum speed-up factor is 6.2. From detailed results (not given here) it can be observed that it is important to use two-dimensional buckets when both resources are similarly constraining.

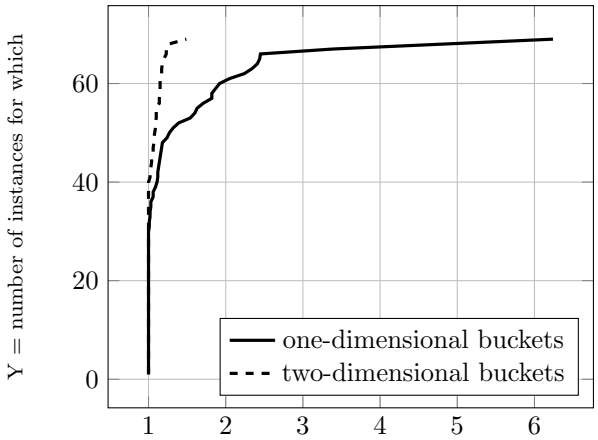

Figure 7: Performance profiles for variants with one-dimensional and two-dimensional buckets

In Table 5 we give the geometric means of total solution time ratios for five variants of the algorithm applied for instances with two resources. First three variants (O), (A), (A^+), (A^+B) use one-dimensional buckets, and the last variant (A^+B) uses two-dimensional buckets. Again, variant (A) uses parameterisation $\xi = 200$. We time ratios for all 69 instances as well as for instances of different problems separately. Results in Table 5 suggest the most computational improvement is again achieved when passing from variant (O) to variant (A). Significant speed-ups achieved when passing from (A) to (A^+) and from one-dimensional buckets to two-dimensional. Impact of the bucket arc elimination procedure is very small here.

6.5 Results for the BCP algorithm

In this section, we present the performance of the proposed BCP algorithm on instances from the literature. We start with 74 VRPTW instances having 100 and 200 customers. We compare our algorithm with the state-of-the-art BCP algorithm for this problem by Pecin et al. (2017a). The

Instances	Number	One-dimensional buckets (O)→(A)	(A)→(A$^+$)	(A$^+$)→(A$^+$B)	One → two (A$^+$B)
DCVRP	7	2.72	1.08	1.05	1.02
MDVRP	22	1.80	1.25	1.01	1.12
SDVRP	10	6.16	0.88	0.93	1.89
VRPTW	30	1.78	1.23	1.05	1.14
All	69	2.23	1.16	1.02	1.21

Table 5: Comparison of different variants of the algorithm on instances with two resources

Class	Our algorithm			Algorithm in Pecin et al. (2017a)		
	Solved	Average time	Geomean time	Solved	Average time	Geomean time
Solomon hard	14/14	456	160	14/14	5324	961
C1	10/10	100	14	10/10	133	61
C2	9/10	4774	510	8/10	4096	3027
R1	10/10	11756	1418	10/10	12854	1861
R2	10/10	7248	2158	8/10	13626	8991
RC1	8/10	15370	6351	8/10	38940	13585
RC2	9/10	29970	3751	7/10	27372	14390
Overall	70/74	9044	690	65/74	13066	1956
Solved by both		4350	500		13066	1956

Table 6: Comparison of our BCP algorithm with the one in Pecin et al. (2017a) on the VRPTW instances

difference between the two algorithms mainly resides in the different labeling routines included. Another difference is a better dual price smoothing stabilization with the automatically adjusted parameter (Pessoa et al. 2018) used by us. Also, we use only standard RCCs and limited-memory R1Cs with up to five rows, whereas algorithm in Pecin et al. (2017a) uses also elementary inequalities (a family of R1Cs) with six and seven rows and clique cuts (after enumeration). Finally, different mechanisms to dynamically adjust *ng*-neighborhoods are employed. Other components used in both algorithms are similar.

The computer used in Pecin et al. (2017a) and our computer have approximately the same speed, according to `www.cpubenchmark.net/singleThread.html`. In Table 6, we present the comparison of the two algorithms. Our algorithm outperforms the one by Pecin at al. for all classes of VRPTW instances, except class R1 for which the results are similar. Our algorithm proved optimality for five out of nine open instances with 200 customers for the first time. On the set of instances solved by the both algorithms, our approach is three times faster if we compare average solution times, and four times faster if we compare the geometric mean of solution times. Speed-up factor on individual instances reaches 47.5 as demonstrated in the detailed results presented in the online appendix. Interestingly, the largest improvement factors are reached on instances for which the improved arc elimination procedure had the most impact.

We have also tested our algorithm on VRPTW instances with 400 customers. In the literature only one of these instances has been solved to optimality, by Kallehauge, Larsen, and Madsen (2006). We managed to solve to optimality 24 instances out of 60. We observed three main reasons (in decreasing order of the number of impacted instances) why algorithm could not solve more 400-customer instances: (i) the labeling algorithm is not fast enough, so only relatively few R1Cs can be added before the pricing time limit is exceeded, leaving a gap that is too large (ii) R1Cs with up to five rows can be added almost until convergence, but they still leave a gap that is too large, and (iii) column generation convergence is very slow.

In the next experiment, we compare our algorithm with the exact algorithm by Contardo and Martinelli (2014) on the set of distance-constrained MDVRP instances. The main features of that previous algorithm that differ from ours are:

Class	Our algorithm			Algorithm in Contardo and Martinelli (2014)		
	Solved	Average time	Geomean time	Solved	Average time	Geomean time
"p"	12/12	4538	201	6/6	938	342
"pr"	10/10	13773	937	4/7	39058	903
Overall	22/22	7881	392	10/13	16186	504
Solved by both		147	30		16186	504

Table 7: Comparison of our BCP algorithm with the one in Contardo and Martinelli (2014) on the distance-constrained MDVRP instances

- Use of standard mono-directional label correcting algorithm for solving the pricing problem (improved by decremental state-space relaxation);
- Absence of column generation stabilization;
- Separation of different families of valid inequalities, particularly, only a subset of R1Cs is used (Subset Row Cuts), and the limited memory technique is not applied;
- Exclusive use of route enumeration for finishing the instance, without any branching.

The speed of the computer used in Contardo and Martinelli (2014) is approximately 1.6 times slower than ours, according to `www.cpubenchmark.net/singleThread.html`. Therefore, in the following comparison, as well in the detailed results in the online appendix, the solutions times of Contardo and Martinelli (2014) are divided by 1.6. In Table 7, we present the comparison of the two algorithms. Our algorithm solves all instances, most of them for the first time. When compared on the set of instances solved by the both algorithms, our approach is 110 times faster in terms of the average time, and 16.5 times faster in terms of the geometric mean time (33 times faster if the smallest instance "pr01" is not taken into account). Speed-up factor on individual instances reaches three orders of magnitude as demonstrated in the detailed results presented in the online appendix. We believe that all the differences pointed above, not only the bucket graph pricing, contribute to this big improvement. For the instance "pr10", we improved the best known solution. For other instances, the values obtained by Vidal et al. (2012) are optimal.

In the following Table 8 we give the summary results for remaining instances. To our knowledge, these instances are treated by an exact approach for the first time. We solved to optimality six out of seven classic distance-constrained CVRP instances by Christofides, Mingozzi, and Toth (1979). All best known solution values for these instances given in Nagata and Bräysy (2009) are optimal. The only unsolved instance CMT13 is clustered and has many long arcs in the solution. The gap for this instance is large even after adding many R1Cs. This gap can be reduced by adding even more cuts, but exact pricing becomes too expensive.

We solved to optimality seven out of ten of distance-constrained site-dependent instances. These instances happen to be harder than similar multi-depot instances in class "pr". Best known solution values from Cordeau and Maischberger (2012) were improved for four instances.

Finally, we have considered HFVRP instances from Duhamel, Lacomme, and Prodhon (2011) who called these instances "nightmare". Particularities of these instances are large capacities of vehicles, very heterogeneous customer demands, and many – up to eight – very heterogeneous vehicle types. We confirm that these instances are particularly difficult. We could not solve several instances which have slightly more than 100 customers. Nevertheless, we have solved to optimality all instances with up to 100 customers, 29 from 38 instances with 101–150 customers, 13 from 30 instances with 151–200 customers. Improved solutions were found for 43 instances.

The detailed results for all tested instances are given in the online appendix.

Class	Solved	Largest solved n	Smallest unsolved n	Average time	Geomean time
DCVRP	6/7	199	120	5h30m	16m44s
SDVRP	7/10	216	240	32m09s	11m26s
HFVRP	56/96	186	107	3h56m	23m07s

Table 8: Results for instances never attacked by exact methods before

7 Conclusions

This article proposed a new bucket-graph based variant of the labeling algorithm for the SPPRC. It can handle the particularities of the SPPRCs that arise as pricing suproblems in the most effective existing exact algorithms for the VRP: the partial route elementarity defined by ng-sets and the numerous additional dimensions induced by limited-memory R1Cs. The algorithm is especially better than existing variants when the consumptions, in at least one of the resources, are given either by real numbers or by large integer numbers. Besides, a new procedure that generalizes the principle of arc elimination by reduced costs to the bucket graph context was given.

The new algorithms were embedded in an advanced BCP algorithm, that includes several other features picked from the literature. The computational results were highly positive. On the classical VRPTW, the new BCP is at least four times faster on average than a very recent BCP algorithm (Pecin et al. 2017a). Moreover, the new BCP solved five additional VRPTW instances to optimality. Since most other elements in those two BCP algorithms, like ng-paths and rank-1 cuts, are similar, the significant improvement is clearly due to the techniques introduced in the paper. The experimental investigation suggests that the largest improvement comes from the organisation of labels in buckets which decreases the number of dominance checks in the pricing. In fact, this proposed labeling implementation can better handle the large numbers that define the time resource in the benchmark VRPTW instances. Other components as dynamic adjustment of the bucket step size and an improved arc elimination procedure bring less speed-up on average but can be important for some instances.

Very good results were also obtained on instances from other important VRP variants, including classical DCVRP instances proposed almost 40 years ago, but only now solved to optimality.

Although the bucket implementation presented in this paper considered up to two resources, it is not difficult to extend it for handling the SPPRCs with several resources that arise in other VRP variants. We believe that those harder problems may also benefit from the approach. However, substantial additional experimental work would be needed for determining how many and which resources should be used for defining the buckets, the remaining resources being treated as in the standard labeling.

Acknowledgements

Experiments presented in this paper were carried out using the PlaFRIM (Federative Platform for Research in Computer Science and Mathematics), created under the Inria PlaFRIM development action with support from Bordeaux INP, LABRI and IMB and other entities: Conseil Régional d'Aquitaine, Université de Bordeaux, CNRS and ANR in accordance to the "Programme d'Investissements d'Avenir" (see `www.plafrim.fr/en/home`).

AP and EU visited Bordeaux with funding from project FAPERJ E26/110.075/2014 (International Cooperation FAPERJ/INRIA).

The authors are thankful to Thibaut Vidal for running the hybrid genetic algorithm and communicating initial upper bounds for VRPTW instances with 200 and 400 customers.

The authors are grateful to two anonymous referees and the associate editor whose comments

helped to improve the quality of the paper.

References

Achterberg T, 2007 *Constraint Integer Programming*. Ph.D. thesis, Technische Universitat Berlin.

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

Bulhoes T, Sadykov R, Uchoa E, 2018 *A branch-and-price algorithm for the minimum latency problem. Computers & Operations Research* 93:66–78.

Christofides N, Mingozzi A, Toth P, 1979 *The vehicle routing problem*. Christofides N, Mingozzi A, Toth P, Sandi C, eds., *Combinatorial Optimization*, 315–338 (Chichester, UK: Wiley).

Christofides N, Mingozzi A, Toth P, 1981 *State-space relaxation procedures for the computation of bounds to routing problems. Networks* 11(2):145–164.

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

Contardo C, Martinelli R, 2014 *A new exact algorithm for the multi-depot vehicle routing problem under capacity and route length constraints. Discrete Optimization* 12:129 – 146.

Cordeau JF, Gendreau M, Laporte G, 1997 *A tabu search heuristic for periodic and multi-depot vehicle routing problems. Networks* 30(2):105–119.

Cordeau JF, Laporte G, 2001 *A tabu search algorithm for the site dependent vehicle routing problem with time windows. INFOR: Information Systems and Operational Research* 39(3):292–298.

Cordeau JF, Maischberger M, 2012 *A parallel iterated tabu search heuristic for vehicle routing problems. Computers and Operations Research* 39(9):2033 – 2050.

Dell'Amico M, Díaz JCD, Hasle G, Iori M, 2016 *An adaptive iterated local search for the mixed capacitated general routing problem. Transportation Science* 50(4):1223–1238.

Desaulniers G, Lessard F, 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.

Duhamel C, Lacomme P, Prodhon C, 2011 *Efficient frameworks for greedy split and new depth first search split procedures for routing problems. Computers and Operations Research* 38(4):723 – 739.

Escobar JW, Linfati R, Toth P, Baldoquin MG, 2014 *A hybrid granular tabu search algorithm for the multi-depot vehicle routing problem. Journal of Heuristics* 20(5):483–509.

Fukasawa R, Longo H, Lysgaard J, Poggi de Aragão M, Reis M, Uchoa E, Werneck R, 2006 *Robust branch-and-cut-and-price for the capacitated vehicle routing problem. Mathematical Programming* 106(3):491–511.

Gehring H, Homberger J, 2002 *Parallelization of a two-phase metaheuristic for routing problems with time windows. Journal of Heuristics* 8(3):251–276.

Irnich S, Desaulniers G, 2005 *Shortest path problems with resource constraints*. Desaulniers G, Desrosiers J, Solomon MM, eds., *Column Generation*, 33–65 (Boston, MA: Springer US).

Irnich S, Desaulniers G, Desrosiers J, 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, Pisinger D, 2008 *Subset-row inequalities applied to the vehicle-routing problem with time windows. Operations Research* 56(2):497–511.

Jiang J, Ng KM, Poh KL, Teo KM, 2014 *Vehicle routing problem with a heterogeneous fleet and time windows. Expert Systems with Applications* 41(8):3748 – 3760.

Kallehauge B, Larsen J, Madsen OB, 2006 *Lagrangian duality applied to the vehicle routing problem with time windows. Computers and Operations Research* 33(5):1464 – 1487.

Laporte G, Nobert Y, 1983 *A branch and bound algorithm for the capacitated vehicle routing problem. OR Spectrum* 5(2):77–85.

Lozano L, Duque D, Medaglia AL, 2015 *An exact algorithm for the elementary shortest path problem with resource constraints. Transportation Science* 50(1):348–357.

Lozano L, Medaglia AL, 2013 *On an exact method for the constrained shortest path problem. Computers and Operations Research* 40(1):378 – 384.

Nagata Y, Bräysy O, 2009 *Edge assembly-based memetic algorithm for the capacitated vehicle routing problem. Networks* 54(4):205–215.

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

Pecin D, Pessoa A, Poggi M, Uchoa E, 2014 *Improved branch-cut-and-price for capacitated vehicle routing.* Lee J, Vygen J, eds., *Integer Programming and Combinatorial Optimization*, 393–403 (Springer).

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

Pecin D, Pessoa A, Poggi M, Uchoa E, Santos H, 2017c *Limited memory rank-1 cuts for vehicle routing problems. Operations Research Letters* 45(3):206 – 209.

Penna PHV, Subramanian A, Ochi LS, Vidal T, Prins C, 2019 *A hybrid heuristic for a broad class of vehicle routing problems with heterogeneous fleet. Annals of Operations Research* 273(1):5–74.

Pessoa A, de Aragão MP Marcus, Uchoa E, 2008 *Robust branch-cut-and-price algorithms for vehicle routing problems.* Golden B, Raghavan S, Wasil E, eds., *The Vehicle Routing Problem: Latest Advances and New Challenges*, volume 43 of *Operations Research/Computer Science Interfaces*, 297–325 (Springer US).

Pessoa A, Sadykov R, Uchoa E, 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, Uchoa E, de Aragão MP, Rodrigues R, 2010 *Exact algorithm over an arc-time-indexed formulation for parallel machine scheduling problems. Mathematical Programming Computation* 2(3):259–290.

Poggi M, Uchoa E, 2014 *New exact algorithms for the capacitated vehicle routing problem. Vehicle Routing: Problems, Methods, and Applications, P. Toth, and D. Vigo (Eds), SIAM* 59–86.

Pugliese LDP, Guerriero F, 2013 *A survey of resource constrained shortest path problems: Exact solution approaches. Networks* 62(3):183–200.

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

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

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

Tilk C, Rothenbächer AK, Gschwind T, 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.

Vanderbeck F, Sadykov R, Tahiri I, 2017 *BaPCod — a generic Branch-And-Price Code.* URL `https://realopt.bordeaux.inria.fr/?page_id=2`.

Vidal T, Crainic TG, Gendreau M, Lahrichi N, Rei W, 2012 *A hybrid genetic algorithm for multidepot and periodic vehicle routing problems. Operations Research* 60(3):611–624.

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

A Detailed results

In this appendix, we present the detailed results for our BCP algorithm on all tested instances.

In the following tables, we show the instance name, n — the number of customers, m — the number of different vehicle types or the number of depots, when applicable, the initial upper bound, the "robust" dual bound (before adding non-robust cuts), the dual bound obtained in the root node, the root node time, the number of R1Cs at the end of the root node, the number of nodes, the total time, and the best solution value obtained by the algorithm. For instances solved before by another algorithm in the literature, we give the corresponding time (which takes into account the difference between the speed of the computers used), and the improvement factor

(ratio of the running times of the literature algorithm and ours). In Tables 9–15, "Time PCDU" refers to the running time of the algorithm by Pecin et al. (2017a). In Tables 22–23, "Time CM" refers to the running time of the algorithm by Contardo and Martinelli (2014). In the tables, * near the best solution value indicates that its optimality was proven for the first time. If the best solution value is underlined, it improves on the best known solution value. All obtained solutions were proven to be optimal except when the total time is given with the ">" sign.

Instance	Initial	Robust	Root			Nodes	Total	Final	Time	Improv.
	UB	bound	bound	time	#R1C	number	time	UB	PCDU	factor
C203	588.8	588.7	588.7	27s	0	1	27s	588.7	6m05s	13.3
C204	588.2	588.1	588.1	40s	0	1	40s	588.1	27m28s	40.9
R202	1029.7	1022.2	1029.6	2m25s	72	1	2m25s	1029.6	5m16s	2.2
R203	870.9	866.9	870.8	1m16s	17	1	1m16s	870.8	9m25s	7.4
R204	731.4	724.9	731.3	3m04s	56	1	3m04s	731.3	24m55s	8.1
R206	876.0	866.8	875.9	2m30s	51	1	2m30s	875.9	9m06s	3.6
R207	794.1	790.1	794.0	1m43s	27	1	1m43s	794.0	12m08s	7.0
R208	701.1	691.6	697.7	14m57s	138	9	1h14m	701.0	17h48m	14.3
R209	854.9	841.4	854.8	3m45s	105	1	3m45s	854.8	9m59s	2.7
R210	900.6	889.4	900.5	4m34s	107	1	4m34s	900.5	12m09s	2.7
R211	746.8	734.6	746.7	3m42s	110	1	3m42s	746.7	22m00s	5.9
RC204	783.6	779.3	783.5	2m11s	15	1	2m11s	783.5	19m27s	8.9
RC207	963.0	947.3	962.9	3m21s	91	1	3m21s	962.9	5m47s	1.7
RC208	776.2	766.6	776.1	2m02s	36	1	2m02s	776.1	9m59s	4.9

Table 9: Detailed results for the hardest 100-customers Solomon VRPTW instances

Instance	Initial	Robust	Root			Nodes	Total	Final	Time	Improv.
	UB	bound	bound	time	#R1C	number	time	UB	PCDU	factor
C1_2_1	2698.7	2698.6	2698.6	8s	0	1	8s	2698.6	20s	2.2
C1_2_2	2694.4	2694.3	2694.3	20s	0	1	20s	2694.3	44s	2.1
C1_2_3	2675.9	2659.7	2675.8	4m39s	66	1	4m39s	2675.8	10m26s	2.2
C1_2_4	2625.7	2619.8	2625.6	6m51s	42	1	6m51s	2625.6	5m40s	0.8
C1_2_5	2695.0	2694.9	2694.9	11s	0	1	11s	2694.9	24s	2.1
C1_2_6	2695.0	2694.9	2694.9	15s	0	1	15s	2694.9	26s	1.7
C1_2_7	2695.0	2694.9	2694.9	18s	0	1	18s	2694.9	22s	1.2
C1_2_8	2684.1	2683.7	2684.0	19s	0	1	19s	2684.0	28s	1.4
C1_2_9	2639.7	2637.2	2639.6	54s	0	1	54s	2639.6	1m24s	1.5
C1_2_10	2624.8	2620.2	2624.7	2m40s	34	1	2m40s	2624.7	1m53s	0.7

Table 10: Detailed results for 200-customer Gehring and Homberger VRPTW instances of class C1

Instance	Initial	Robust	Root			Nodes	Total	Final	Time	Improv.
	UB	bound	bound	time	#R1C	number	time	UB	PCDU	factor
C2_2_1	1922.2	1915.0	1922.1	3m06s	70	1	3m06s	1922.1	14m20s	4.6
C2_2_2	1851.5	1839.4	1851.4	6m57s	80	1	6m57s	1851.4	1h29m	12.9
C2_2_3	1763.5	1737.0	1752.1	32m16s	287	37	11h10m	1763.4*	>96h	>8.6
C2_2_4	1695.1	1652.3	1663.2	28m59s	187	355	>60h	1695.1	>96h	–
C2_2_5	1869.7	1858.7	1869.6	3m57s	52	1	3m57s	1869.6	23m48s	6.0
C2_2_6	1844.9	1836.6	1844.8	3m22s	23	1	3m22s	1844.8	34m03s	10.1
C2_2_7	1842.3	1834.9	1842.2	3m03s	21	1	3m03s	1842.2	44m33s	14.6
C2_2_8	1813.8	1801.3	1813.7	3m11s	12	1	3m11s	1813.7	42m22s	13.3
C2_2_9	1815.1	1795.0	1815.0	9m17s	101	1	9m17s	1815.0	2h01m	13.1
C2_2_10	1791.3	1765.7	1791.2	13m04s	96	1	13m04s	1791.2	2h56m	13.5

Table 11: Detailed results for 200-customer Gehring and Homberger VRPTW instances of class C2

Instance	Initial	Robust	Root			Nodes	Total	Final	Time	Improv.
	UB	bound	bound	time	#R1C	number	time	UB	PCDU	factor
R1_2_1	4667.3	4655.0	4667.2	14s	22	1	14s	4667.2	59s	4.0
R1_2_2	3920.0	3907.2	3919.9	37s	75	1	37s	3919.9	2m42s	4.3
R1_2_3	3374.0	3320.3	3352.5	13m27s	502	693	17h01m	3373.9	25h56m	1.5
R1_2_4	3047.7	3001.5	3035.3	23m03s	635	267	10h45m	3047.6	5h00m	0.5
R1_2_5	4053.3	4007.6	4044.8	3m21s	290	9	10m51s	4053.2	7m25s	0.7
R1_2_6	3559.3	3493.4	3545.3	11m16s	559	71	2h00m	3559.1	2h00m	1.0
R1_2_7	3142.0	3099.2	3141.9	9m27s	335	1	9m27s	3141.9	16m42s	1.8
R1_2_8	2938.5	2902.2	2933.9	19m02s	436	5	27m38s	2938.4	41m39s	1.5
R1_2_9	3734.8	3678.3	3726.4	6m23s	443	7	14m43s	3734.7	12m51s	0.9
R1_2_10	3293.2	3244.9	3278.2	8m35s	489	49	1h47m	3293.1	1h23m	0.8

Table 12: Detailed results for 200-customer Gehring and Homberger VRPTW instances of class R1

Instance	Initial	Robust	Root			Nodes	Total	Final	Time	Improv.
	UB	bound	bound	time	#R1C	number	time	UB	PCDU	factor
R2_2_1	3468.1	3462.5	3468.0	2m20s	19	1	2m20s	3468.0	34m43s	14.9
R2_2_2	3008.3	3000.9	3008.2	7m33s	69	1	7m33s	3008.2	1h07m	9.0
R2_2_3	2537.6	2516.3	2537.5	33m26s	181	1	33m26s	2537.5	5h50m	10.5
R2_2_4	1928.6	1918.3	1924.9	33m39s	94	33	3h59m	1928.5*	>96h	>24.0
R2_2_5	3061.2	3029.4	3061.1	21m41s	123	1	21m41s	3061.1	1h45m	4.9
R2_2_6	2675.5	2644.9	2675.2	37m20s	249	3	50m39s	2675.4	4h04m	4.8
R2_2_7	2304.8	2277.8	2304.7	1h48m	335	1	1h48m	2304.7	12h16m	6.8
R2_2_8	1842.5	1808.9	1832.1	1h41m	393	19	11h47m	1842.4*	>96h	>8.1
R2_2_9	2843.4	2824.3	2843.3	12m37s	110	1	12m37s	2843.3	1h23m	6.6
R2_2_10	2549.5	2528.7	2545.4	15m33s	202	3	24m04s	2549.4	3h14m	8.1

Table 13: Detailed results for 200-customer Gehring and Homberger VRPTW instances of class R2

Instance	Initial	Robust	Root			Nodes	Total	Final	Time	Improv.
	UB	bound	bound	time	#R1C	number	time	UB	PCDU	factor
RC1_2_1	3517.0	3459.6	3498.7	4m26s	338	43	40m29s	3516.9	42m10s	1.0
RC1_2_2	3221.7	3177.8	3206.1	11m52s	522	17	33m38s	3221.6	3h10m	5.7
RC1_2_3	3001.5	2952.1	2982.9	22m40s	630	187	14h41m	3001.4	18h15m	1.2
RC1_2_4	2845.3	2814.4	2831.0	26m52s	361	41	4h58m	2845.2	6h23m	1.3
RC1_2_5	3325.7	3272.9	3312.7	49m59s	678	15	1h22m	3325.6	2h23m	1.7
RC1_2_6	3300.8	3255.3	3295.1	12m30s	468	5	16m35s	3300.7	26m46s	1.6
RC1_2_7	3177.9	3114.7	3151.8	17m14s	554	109	10h46m	3177.8	52h20m	4.9
RC1_2_8	3060.1	3022.6	3047.4	19m03s	614	5	49m56s	3060.0	2h50m	3.4
RC1_2_9	3074.9	3006.6	3035.6	16m42s	464	493	>60h	3074.9	>96h	–
RC1_2_10	2990.6	2938.7	2966.4	24m40s	583	683	>60h	2990.6	>96h	–

Table 14: Detailed results for 200-customer Gehring and Homberger VRPTW instances of class RC1

Instance	Initial	Robust	Root			Nodes	Total	Final	Time	Improv.
	UB	bound	bound	time	#R1C	number	time	UB	PCDU	factor
RC2_2_1	2797.5	2789.4	2797.4	4m11s	38	1	4m11s	2797.4	35m49s	8.6
RC2_2_2	2481.7	2469.5	2477.0	19m21s	183	3	39m25s	2481.6	4h50m	7.4
RC2_2_3	2227.8	2204.7	2227.7	1h54m	308	1	1h54m	2227.7	13h32m	7.1
RC2_2_4	1854.9	1839.7	1848.3	1h33m	196	31	10h52m	1854.8*	>96h	>8.8
RC2_2_5	2493.6	2479.0	2491.4	13m36s	142	1	13m36s	2491.4	1h25m	6.3
RC2_2_6	2496.4	2475.6	2495.1	14m09s	83	1	14m09s	2495.1	1h33m	6.6
RC2_2_7	2287.8	2271.2	2279.0	17m22s	98	11	59m07s	2287.7	8h11m	8.3

RC2_2_8	2151.3	2140.7	2151.2	29m09s	92	1	29m09s	2151.2	23h05m	47.5
RC2_2_9	2086.7	2056.5	2071.1	1h54m	240	109	59h29m	2086.6*	>96h	>1.6
RC2_2_10	1989.3	1953.2	1955.8	51m23s	12	201	>60h	1989.2	>96h	–

Table 15: Detailed results for 200-customer Gehring and Homberger VRPTW instances of class RC2

Instance	Initial	Robust	Root			Nodes	Total	Final
	UB	bound	bound	time	#R1C	number	time	UB
C1_4_1	7138.9	7138.8	7138.8	1m34s	0	1	1m34s	7138.8[c]
C1_4_2	7113.2	7095.0	7102.2	6m13s	79	3	7m17s	7113.2*
C1_4_3	6930.0	6919.8	6929.9	12m51s	81	1	12m51s	6929.9*
C1_4_4	6777.8	6769.6	6777.7	45m22s	118	1	45m22s	6777.7*
C1_4_5	7138.9	7138.8	7138.8	1m58s	0	1	1m58s	7138.8*
C1_4_6	7140.2	7140.1	7140.1	2m11s	0	1	2m11s	7140.1*
C1_4_7	7136.3	7135.9	7136.2	2m27s	0	1	2m27s	7136.2*
C1_4_8	7107.1	7030.0	7071.2	13m02s	277	61	2h07m	<u>7083.0</u>*
C1_4_9	6927.9	6888.3	6927.8	22m43s	355	1	22m43s	6927.8*
C1_4_10	6825.5	6798.8	6819.2	44m59s	347	13	1h02m	6825.4*

[c] this instance has been solved before by Kallehauge, Larsen, and Madsen (2006)

Table 16: Detailed results for 400-customer Gehring and Homberger VRPTW instances of class C1

Instance	Initial	Robust	Root			Nodes	Total	Final
	UB	bound	bound	time	#R1C	number	time	UB
C2_4_1	4100.4	4088.4	4100.3	15m08s	35	1	15m08s	4100.3*
C2_4_2	3914.2	3897.5	3914.1	9h39m	283	1	9h39m	3914.1*
C2_4_3	3755.3	3722.7	3740.8	60h00m	566	1	>60h	3755.3
C2_4_4	3524.1	3483.3	3498.4	3h00m	288	11	>60h	3524.1
C2_4_5	3923.3	3919.7	3923.2	23m01s	58	1	23m01s	3923.2*
C2_4_6	3860.2	3843.8	3860.1	39m33s	99	1	39m33s	3860.1*
C2_4_7	3871.0	3861.6	3870.9	29m47s	82	1	29m47s	3870.9*
C2_4_8	3773.8	3743.4	3770.6	1h11m	397	9	4h29m	3773.7*
C2_4_9	3843.8	3782.0	3815.9	4h09m	542	15	>60h	3843.8
C2_4_10	3665.2	3616.9	3656.2	2h33m	682	73	>60h	3665.1

Table 17: Detailed results for 400-customer Gehring and Homberger VRPTW instances of class C2

Instance	Initial	Robust	Root			Nodes	Total	Final
	UB	bound	bound	time	#R1C	number	time	UB
R1_4_1	10313.9	10243.5	10284.1	4m00s	181	25	16m48s	<u>10305.8</u>*
R1_4_2	8876.0	8776.2	8834.3	17m52s	534	1683	>60h	8876.0
R1_4_3	7794.8	7673.7	7732.2	54m14s	658	327	>60h	7794.8
R1_4_4	7281.8	7182.4	7239.2	2h06m	1095	123	>60h	7281.8
R1_4_5	9188.7	9082.3	9145.8	6m35s	355	2393	>60h	9188.7
R1_4_6	8366.6	8257.8	8314.1	28m04s	596	1139	>60h	8366.6
R1_4_7	7609.1	7489.2	7558.6	1h25m	1020	185	>60h	7609.1
R1_4_8	7253.5	7148.5	7200.0	1h57m	967	147	>60h	7253.5
R1_4_9	8674.8	8545.7	8618.4	13m45s	611	1265	>60h	8674.8
R1_4_10	8088.9	7944.9	8024.2	31m57s	782	611	>60h	8088.9

Table 18: Detailed results for 400-customer Gehring and Homberger VRPTW instances of class R1

Instance	Initial	Robust	Root			Nodes	Total	Final
	UB	bound	bound	time	#R1C	number	time	UB
R2_4_1	7522.1	7485.3	7520.7	42m42s	96	1	42m42s	7520.7*
R2_4_2	6482.9	6466.3	6482.8	1h33m	127	1	1h33m	6482.8*
R2_4_3	5373.0	5349.3	5372.9	6h18m	333	1	6h18m	5372.9*
R2_4_4	4213.8	4144.1	4169.8	12h06m	1131	15	>60h	4213.8
R2_4_5	6572.1	6498.0	6559.6	10h06m	450	3	>60h	6572.1
R2_4_6	5814.6	5763.1	5811.8	24h11m	1423	3	>60h	5814.6
R2_4_7	4893.6	4843.2	4878.4	15h39m	1492	11	>60h	4893.6
R2_4_8	4001.1	3897.7	3913.9	20h11m	444	9	>60h	4001.1
R2_4_9	6070.3	6001.5	6059.4	3h44m	704	25	58h55m	6067.8*
R2_4_10	5665.1	5561.6	5619.5	4h09m	758	21	>60h	5665.1

Table 19: Detailed results for 400-customer Gehring and Homberger VRPTW instances of class R2

Instance	Initial	Robust	Root			Nodes	Total	Final
	UB	bound	bound	time	#R1C	number	time	UB
RC1_4_1	8523.0	8368.9	8470.0	16m38s	762	667	>60h	8523.0
RC1_4_2	7887.2	7770.3	7843.0	49m29s	1123	197	>60h	7887.2
RC1_4_3	7525.2	7423.3	7444.7	44m01s	252	269	>60h	7525.2
RC1_4_4	7295.3	7207.8	7242.9	1h43m	805	37	>60h	7295.3
RC1_4_5	8152.4	8033.8	8101.5	44m15s	1115	37	>60h	8152.4
RC1_4_6	8148.6	8005.9	8089.7	47m39s	1192	127	>60h	8148.6
RC1_4_7	7937.0	7830.6	7878.9	48m09s	841	235	>60h	7937.0
RC1_4_8	7765.8	7639.3	7679.0	42m19s	574	201	>60h	7765.8
RC1_4_9	7724.4	7604.8	7639.1	48m15s	455	209	>60h	7724.4
RC1_4_10	7583.2	7479.2	7521.6	1h42m	931	33	>60h	7583.2

Table 20: Detailed results for 400-customer Gehring and Homberger VRPTW instances of class RC1

Instance	Initial	Robust	Root			Nodes	Total	Final
	UB	bound	bound	time	#R1C	number	time	UB
RC2_4_1	6147.4	6120.1	6146.6	3h18m	238	3	3h37m	6147.3*
RC2_4_2	5407.6	5389.5	5407.5	4h06m	458	1	4h06m	5407.5*
RC2_4_3	4573.1	4549.6	4573.0	35h25m	1294	1	35h25m	4573.0*
RC2_4_4	3598.0	3535.9	3535.9	8h25m	0	5	>60h	3598.0
RC2_4_5	5396.7	5363.0	5388.6	4h47m	466	7	>60h	5396.7
RC2_4_6	5332.1	5270.9	5308.5	9h04m	738	3	>60h	5332.1
RC2_4_7	4987.9	4944.9	4968.6	6h08m	453	19	>60h	4987.9
RC2_4_8	4703.2	4602.0	4618.7	4h53m	216	31	>60h	4703.2
RC2_4_9	4519.1	4448.9	4457.0	5h35m	278	21	>60h	4519.1
RC2_4_10	4252.4	4157.0	4157.0	2h11m	0	83	>60h	4252.4

Table 21: Detailed results for 400-customer Gehring and Homberger VRPTW instances of class RC2

Inst.	n	m	Initial	Robust	Root			Nod.	Total	Final	Time	Impr.
			UB	bound	bound	time	#R1C	num.	time	UB	CM	factor
p08	249	2	4372.90	4294.01	4347.67	49m01s	567	29	5h43m	4372.78*,a	–	–
p09	249	3	3858.80	3798.47	3849.96	1h34m	650	19	2h25m	3858.66*	–	–
p10	249	4	3631.30	3582.48	3626.26	37m26s	522	3	41m46s	3631.11*,a	–	–
p11	249	5	3546.20	3488.64	3540.43	44m15s	597	3	49m46s	3546.06*,a	–	–
p13	80	2	1319.10	1318.95	1318.95	1s	0	1	1s	1318.95	31s	17.5
p14	80	2	1360.30	1360.12	1360.12	1s	0	1	1s	1360.12	38s	20.5
p16	160	4	2572.40	2572.23	2572.23	12s	0	1	12s	2572.23	8m27s	39.7

p17	160	4	2709.20	2709.09	2709.09	55s	0	1	55s	2709.09	8m07s	8.8
p19	240	6	3827.20	3822.92	3827.06	1m25s	0	1	1m25s	3827.06	37m15s	26.3
p20	240	6	4058.20	4058.07	4058.07	2m30s	0	1	2m30s	4058.07	38m47s	15.5
p22	360	9	5702.30	5693.87	5702.16	2m28s	0	1	2m28s	5702.16*	–	–
p23	360	9	6078.90	6078.75	6078.75	5m52s	0	1	5m52s	6078.75*	–	–

[a] a solution with smaller value is claimed in Escobar et al. (2014), however the authors were not able to communicate it to us

Table 22: Detailed results for Cordeau et al. MDVRP instances of class "p"

Inst.	n	m	Initial UB	Robust bound	Root			Nod. num.	Total time	Final UB	Time CM	Impr. factor
					bound	time	#R1C					
pr01	48	4	861.40	861.32	861.32	3s	0	1	3s	861.32	0s	0.03
pr02	96	4	1307.50	1287.52	1307.34	4m18s	110	1	4m18s	1307.34*	>75h	>1042.8
pr03	144	4	1803.90	1773.42	1803.80	10m22s	185	1	10m22s	1803.80	31h11m	180.5
pr04	192	4	2058.50	2029.87	2058.31	22m52s	327	1	22m52s	2058.31*	>75h	>196.7
pr05	240	4	2331.30	2293.32	2318.80	1h14m	628	27	6h19m	2331.20*	–	–
pr06	288	4	2676.40	2635.08	2662.92	1h18m	522	21	6h39m	2676.30*	–	–
pr07	72	6	1089.70	1085.59	1089.56	17s	0	1	17s	1089.56	18m28s	63.8
pr08	144	6	1665.00	1640.18	1664.85	8m42s	105	1	8m42s	1664.85	11h53m	82.0
pr09	216	6	2133.30	2109.20	2133.20	23m06s	242	1	23m06s	2133.20*	>75h	>194.8
pr10	288	6	2868.40	2818.03	2850.68	1h40m	583	77	24h06m	2867.26*	–	–

Table 23: Detailed results for Cordeau et al. MDVRP instances of class "pr"

Instance	n	m	Initial UB	Robust bound	Root			Nodes number	Total time	Final UB
					bound	time	#R1C			
pr01	48	4	1380.90	1377.09	1380.77	19s	0	1	19s	1380.77*
pr02	96	4	2304.00	2283.37	2303.89	2m51s	23	1	2m51s	2303.89*
pr03	144	4	2575.50	2538.93	2565.67	19m01s	248	7	24m12s	2574.56*
pr04	192	4	3450.00	3396.73	3432.33	42m22s	453	9	1h47m	3449.84*
pr05	240	4	4377.50	4288.21	4332.64	46m35s	442	217	>60h	4375.67
pr06	288	4	4422.20	4326.08	4329.25	21m45s	21	353	>60h	4421.95
pr07	72	6	1890.00	1868.89	1889.82	6m30s	0	1	6m30s	1889.82*
pr08	144	6	2971.20	2933.81	2962.61	25m44s	222	7	32m37s	2969.92*
pr09	216	6	3536.30	3511.38	3527.76	1h00m	239	5	1h03m	3536.20*
pr10	288	6	4639.80	4537.82	4559.48	48m34s	246	297	>60h	4639.80

Table 24: Detailed results for Cordeau and Laporte SDVRP instances

Instance	n	Initial UB	Robust bound	Root			Nodes number	Total time	Final UB
				bound	time	#R1C			
CMT6	50	555.60	540.52	555.43	7s	49	1	7s	555.43*
CMT7	75	909.80	898.95	909.68	4s	35	1	4s	909.68*
CMT8	100	866.10	842.15	857.86	6m48s	319	17	54m47s	865.94*
CMT9	150	1162.70	1133.47	1153.37	7m58s	429	447	13h48m	1162.55*,[b]
CMT10	199	1396.00	1368.79	1387.35	16m22s	579	443	17h29m	1395.85*
CMT13	120	1541.30	1458.00	1475.23	10m39s	281	149	>60h	1541.30
CMT14	100	866.50	851.24	866.37	45m22s	318	1	45m22s	866.37*

[b] solution with value 1158.41, claimed in Dell'Amico et al. (2016), is infeasible (different rounding convention was used)

Table 25: Detailed results for Christofides et al. DCVRP instances

Instance	n	m	Initial UB	Robust bound	Root bound	Root time	Root #R1C	Nodes number	Total time	Final UB
DLP_75	20	3	453	448.43	452.85	0s	6	1	0s	452.85*
DLP_92	35	3	565	553.05	559.82	4m58s	110	5	9m39s	564.39*
DLP_93	39	3	1037	1011.57	1023.69	12s	46	11	1m39s	1036.99*
DLP_94	46	5	1379	1369.07	1378.25	17s	42	1	17s	1378.25*
DLP_55	56	3	10245	10134.23	10244.34	38s	29	1	38s	10244.34*
DLP_52	59	3	4028	3912.72	3931.40	2m59s	53	11	12m57s	4027.27*
DLP_10	69	4	2108	2048.67	2060.75	3m00s	77	13	18m38s	2107.55*
DLP_39	77	5	2922	2874.88	2902.55	6m03s	302	25	28m47s	2918.87*
DLP_70	78	4	6685	6561.07	6659.88	5m02s	252	3	13m53s	6684.56*
DLP_82	79	3	4767	4630.03	4648.96	1m10s	107	123	1h12m	4718.27*
DLP_08	84	3	4592	4569.97	4591.75	2s	0	1	2s	4591.75*
DLP_36	85	6	5685	5610.15	5684.62	1m29s	115	1	1m29s	5684.62*
DLP_43	86	7	8738	8686.41	8737.02	43s	31	1	43s	8737.02*
DLP_01	92	4	9211	8974.44	9081.63	10m33s	158	3	24m37s	9210.14*
DLP_11	95	4	3368	3288.08	3296.82	3m19s	31	9	20m54s	3367.41*

Table 26: Detailed results for Duhamel et al. HFVRP instances with up to 100 customers

Instance	n	m	Initial UB	Robust bound	Root bound	Root time	Root #R1C	Nodes number	Total time	Final UB
DLP_90	102	4	2347	2249.36	2257.75	5m05s	136	655	40h29m	2278.05*
DLP_17	105	3	5363	5278.74	5326.71	2m50s	131	7	3m30s	5362.83*
DLP_84	105	4	7228	7117.02	7142.28	2m02s	136	19	21m01s	7227.88*
DLP_81	106	4	10687	10483.95	10521.23	3m06s	182	313	5h36m	10583.50*
DLP_2B	107	6	8463	8300.92	8332.22	4m17s	37	1639	>60h	8463.00
DLP_07	108	4	8090	8009.35	8049.38	3m51s	104	19	12m07s	8071.97*
DLP_87	108	4	3754	3658.33	3692.55	6m32s	236	7	29m51s	3753.87*
DLP_47	111	5	16157	15971.69	16123.03	3m54s	250	5	5m49s	16156.12*
DLP_48	111	5	21310	21041.94	21228.34	19m51s	321	7	28m25s	21257.38*
DLP_61	111	3	7293	6997.61	7027.77	3m15s	33	1435	>60h	7293.00
DLP_12	112	4	3544	3410.14	3410.14	1m53s	0	25	1h39m	3543.99*
DLP_30	112	3	6314	6185.97	6207.60	4m11s	186	1489	31h29m	6278.62*
DLP_2A	113	6	7794	7708.25	7793.16	1m42s	77	1	1m42s	7793.16*
DLP_53	115	3	6435	6281.51	6376.47	12m26s	209	31	2h14m	6434.83*
DLP_05	116	5	10877	10764.90	10869.04	29s	30	1	29s	10869.04*
DLP_13	119	5	6697	6618.19	6649.75	5m20s	160	1261	>60h	6696.43
DLP_06	121	8	11689	11618.62	11682.98	2m37s	53	1	2m37s	11682.98*
DLP_03	124	4	10710	10635.69	10669.77	55s	106	5	1m13s	10709.66*
DLP_83	124	4	10020	9871.11	9899.78	2m33s	127	59	51m57s	10001.80*
DLP_68	125	4	8971	8799.96	8852.89	5m39s	192	89	3h00m	8889.03*
DLP_74	125	5	11587	11543.40	11563.64	41s	78	7	4m18s	11586.58*
DLP_21	126	3	5140	5079.13	5110.52	8m11s	289	11	25m05s	5139.84*
DLP_26	126	5	6434	6352.66	6382.52	6m00s	230	453	10h10m	6406.16*
DLP_88	127	5	12389	12328.90	12385.74	29s	25	1	29s	12385.74*
DLP_16	129	6	4157	4000.74	4006.42	4m33s	16	1573	>60h	4157.00
DLP_51	129	3	7722	7628.81	7666.22	10m11s	123	3	29m17s	7721.47*
DLP_31	131	8	4092	4045.03	4083.80	15m20s	284	5	18m24s	4091.52*
DLP_40	132	5	11119	10974.62	11015.52	2m32s	160	317	2h59m	11056.13*
DLP_89	134	5	7087	6982.53	7023.43	4m11s	239	1027	>60h	7087.00
DLP_41	135	7	7598	7444.36	7444.36	4m48s	0	3305	>60h	7598.00
DLP_34	136	6	5748	5700.07	5727.33	7m35s	306	27	1h10m	5739.02*
DLP_60	137	4	17037	16768.33	16934.97	5m40s	285	37	44m38s	17012.42*
DLP_73	137	5	10196	10138.46	10174.15	57s	88	3	2m07s	10195.33*
DLP_28	141	5	5531	5478.74	5499.59	4m24s	208	4705	>60h	5525.90
DLP_25	143	6	7207	7030.07	7042.25	5m46s	100	959	>60h	7207.00
DLP_85	146	4	8774	8690.09	8726.51	7m27s	295	111	2h40m	8763.90*

DLP_79	147	4	7260	7132.58	7165.36	8m22s	187	21	1h13m	7257.97*
DLP_66	150	4	12784	12484.59	12587.01	8m31s	104	999	>60h	12776.24

Table 27: Detailed results for Duhamel et al. HFVRP instances with 101–150 customers

Instance	n	m	Initial UB	Robust bound	Root bound	Root time	Root #R1C	Nodes number	Total time	Final UB
DLP_69	152	4	9148	9003.97	9086.90	7m52s	335	147	3h36m	9127.16*
DLP_76	152	8	11995	11900.23	11933.72	5m28s	212	1195	>60h	11994.22
DLP_56	153	4	31031	30571.34	30764.30	8m26s	250	1591	58h50m	30905.95*
DLP_86	153	5	9028	8935.26	8957.90	8m23s	274	249	8h45m	9020.63*
DLP_37	161	5	6851	6751.15	6787.92	7m57s	354	2017	>60h	6838.72
DLP_64	161	3	17136	16552.72	16552.72	6m38s	25	995	>60h	17135.16
DLP_24	163	4	9102	8990.07	8990.07	3m26s	0	1907	>60h	9102.00
DLP_57	163	4	44782	44302.32	44409.19	8m53s	194	1565	>60h	44782.00
DLP_29	164	4	9143	9044.54	9078.84	9m59s	182	43	2h18m	9132.03*
DLP_09	167	5	7604	7543.04	7563.84	12m52s	274	135	2h16m	7599.72*
DLP_35	168	6	9556	9492.98	9521.16	2m55s	114	3	3m17s	9522.45*
DLP_45	170	3	10477	10294.91	10343.43	8m34s	124	929	>60h	10477.00
DLP_80	171	3	6817	6712.75	6716.88	8m39s	29	219	13h07m	6816.89*
DLP_44	172	3	12192	11895.60	11895.60	12m31s	19	1187	>60h	12192.00
DLP_54	172	4	10352	10089.37	10089.37	7m45s	21	1323	>60h	10352.00
DLP_67	172	5	10885	10604.14	10662.94	9m53s	233	325	23h44m	10772.81*
DLP_63	174	5	19952	19670.12	19799.92	8m54s	108	97	5h24m	19890.65*
DLP_14	176	4	5645	5511.85	5511.85	4m37s	0	1023	>60h	5639.98
DLP_42	178	7	10818	10615.08	10615.08	7m00s	0	889	>60h	10805.94
DLP_02	181	4	11676	11603.82	11632.01	4m27s	104	25	9m34s	11649.81*
DLP_04	183	4	10749	10582.45	10673.39	9m41s	277	985	13h53m	10714.84*
DLP_95	183	2	6176	6100.98	6119.06	6m53s	290	1709	>60h	6170.20
DLP_71	186	3	9835	9711.77	9771.86	10m30s	292	87	4h08m	9798.06*
DLP_72	186	4	5904	5780.39	5831.66	9m35s	322	365	31h31m	5870.43*
DLP_50	187	6	12371	12035.29	12035.29	9m56s	0	1521	>60h	12371.00
DLP_15	188	7	8221	8083.44	8083.44	3m27s	0	873	>60h	8221.00
DLP_33	189	7	9411	9248.07	9283.05	9m51s	135	1129	>60h	9333.39
DLP_77	190	3	6917	6821.36	6826.04	7m49s	10	1215	>60h	6917.00
DLP_78	190	4	7036	6899.96	6899.96	6m40s	0	1099	>60h	7036.00
DLP_59	193	6	14283	13786.96	13786.96	23m42s	0	737	>60h	14283.00
DLP_91	196	4	6375	6228.90	6244.50	10m26s	157	1167	>60h	6348.36

Table 28: Detailed results for Duhamel et al. HFVRP instances with 151–200 customers

Instance	n	m	Initial UB	Robust bound	Root bound	Root time	Root #R1C	Nodes number	Total time	Final UB
DLP_23	203	4	7742	7590.28	7603.76	8m46s	70	1043	>60h	7742.00
DLP_38	205	5	11195	11048.66	11092.13	14m28s	375	869	>60h	11195.00
DLP_58	220	6	23371	22800.97	22800.97	36m42s	10	969	>60h	23371.00
DLP_27	220	5	8423	8315.95	8343.41	10m15s	174	687	>60h	8423.00
DLP_65	223	3	13044	12648.64	12648.64	41m17s	24	657	>60h	13044.00
DLP_19	224	5	11687	11503.00	11526.84	8m25s	98	1053	>60h	11687.00
DLP_62	225	5	23011	22594.36	22645.25	13m36s	47	1047	>60h	22987.90
DLP_22	239	2	13069	12923.02	12956.02	13m12s	357	1193	>60h	13050.00
DLP_32	244	8	9383	9205.76	9235.38	14m29s	108	695	>60h	9383.00
DLP_49	246	8	16182	15681.80	15681.80	15m27s	0	735	>60h	16182.00
DLP_46	250	5	24567	23856.71	23856.71	14m14s	21	913	>60h	24567.00
DLP_18	256	5	9653	9491.55	9547.80	22m29s	519	1159	>60h	9649.05

Table 29: Detailed results for Duhamel et al. HFVRP instances with more than 200 customers