

CADERNOS DO LOGIS

Volume 2017, Number 7

A Bucket Graph Based Labeling Algorithm with Application to Vehicle Routing

Ruslan Sadykov, Eduardo Uchoa, Artur Pessoa

October, 2017



A Bucket Graph Based Labeling Algorithm with Application to Vehicle Routing

Ruslan Sadykov^{a,*}, Eduardo Uchoa^b, Artur Pessoa^b

^a*INRIA Bordeaux – Sud-Ouest*

200 Avenue de la Veille Tour, 33405 Talence, France

^b*Universidade Federal Fluminense - Departamento de Engenharia de Produção
Rua Passo da Pátria 156, Niterói - RJ - Brasil - 24210-240*

Abstract

We consider the Resource Constrained Shortest Path problem 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 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 filter 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 constraints. Computational experiments were performed on instances from the distance constrained vehicle routing problem, including multi-depot and site-depended variants, on the vehicle routing problem with time windows, and on the “nightmare” instances of the heterogeneous fleet vehicle routing problem. Very significant improvements over the best algorithms in the literature were achieved and many instances could be solved for the first time.

Keywords: Labeling Algorithm; Resource Constrained Shortest Path; Routing

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 involves 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 Resource Constrained Shortest Path (RCSP) problems. In those problems, one looks for least cost paths joining a source vertex to 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.

*Corresponding author.

Email addresses: ruslan.Sadykov@inria.fr (Ruslan Sadykov), uchoa@producao.uff.br (Eduardo Uchoa), artur@producao.uff.br (Artur Pessoa)

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, initially marked as non-extended. 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 the 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 RCSP, 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 algorithms 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 can not 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, corresponding to the standard textbook labeling algorithm, 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) recently proposed the so-called Pulse Algorithm for the RCSP. 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 can not be extended into a full path ending at the sink vertex

that cost less than the currently best known source to sink path. Pulse produced excellent results, much better than label-setting algorithms, on some stand-alone RCSPs 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 subproblem solver in a column generation algorithm for VRP in Lozano et al. (2015). The RCSPs that appear in that context are more complex because the master dual variables make many arc costs to be negative. The test was calculating the elementary route bound for the VRP with Time Windows (VRPTW) on instances with 100 customers. The variant of Pulse used 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 favorably with the labeling algorithms from Desaulniers et al. (2008) and Baldacci et al. (2011). It is not known how Pulse would perform on larger instances and how it would handle the modifications in the RCSP 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 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 “dominance-free”, i.e. there are no labels which are dominated by a label in the same bucket. Dominance between labels in different buckets is 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, 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 will be acyclic and the algorithm becomes label-setting. In general, the bucket graph has cycles and the algorithm is label-correcting. 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 problems, for example, on the classical Capacitated VRP (CVRP). On those cases, the backward search is replaced by a “reversed copy” of the forward search, thus halving 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. The concept of *jump bucket arc* 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 VRP. In fact, the 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 constraints and/or large vehicle capacity. Early experiments have shown that the labeling algorithm could perform very well, but that performance was quite sensitive to the choice of the step size parameter. Moreover, the optimal step size value varied a lot from instance to instance. Therefore, a simple but effective scheme for automatic dynamic adjustment of the step size was devised. Other specific experiments indicate that, on the hardest instances, the new bucket graph fixing procedure can indeed be significantly better than existing schemes for fixing 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).

2. Application

In order to give a precise definition of the RCSP 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 RCSP arises as the pricing subproblem.

2.1. HFVRPTW Definition

The Heterogeneous Fleet Vehicle Routing Problem with Time Windows (HFVRPTW) is defined as follows. Let $G = (V, A)$ be a directed graph where $V = \{0, \dots, n+1\}$. 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 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. 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 time t_a^m associated to the traversal of this arc by a vehicle of type m . In many cases, the arc cost is proportional to the arc time: $c_a^m = \rho_m t_a^m$, where ρ_m is a type-dependent travel cost per time unit. We assume that G does not have any cycle where, for some vehicle type, all vertices have zero demands and service times and all arcs have zero time. A route $P = (v_0^P = 0, v_1^P, \dots, v_k^P, v_{k+1}^P = n + 1)$ for a vehicle of type $m \in M$ visiting $k > 0$ customers $v_1^P, \dots, v_k^P \in V'$ is said to be feasible if 1) it satisfies vehicle capacity: $\sum_{j=1}^k w_{v_j^P} \leq W_m$; and 2) the earliest start of service time S_j at every visited vertex v_j^P , $0 \leq j \leq k + 1$, falls within the corresponding time window: $l_{v_j^P} \leq S_j \leq u_{v_j^P}$, where $S_0 = l_0$ and $S_j = \max\{l_{v_j^P}, S_{j-1} + s_{v_{j-1}^P} + t_{(v_{j-1}^P, v_j^P)}^m\}$. 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 $f_m + \sum_{j=1}^{k+1} c_{(v_{j-1}^P, v_j^P)}^m$. The objective is to determine a set of feasible routes that minimize the sum their costs 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), vehicles have identical capacities and cost, but 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 $c_{(0,v)}^m$ and times $t_{(0,v)}^m$ for leaving the depot, together with costs $c_{(v,n+1)}^m$ and times $t_{(v,n+1)}^m$ 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 infinity costs for $c_{(v,v')}^m$ if vehicle type m can not 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 route duration constraint. It can be modeled by setting time window $[0, D]$ for every $v \in V$. Note that the variant in which capacity, route duration and time windows constraints are present is not a special case of HFVRPTW, and is not considered in this paper. This happens because the route duration is measured with respect to the time the vehicle leaves the depot. If there are restricting time windows in the customers, it may be advantageous not to leave the depot at time 0 in order to reduce route duration.

2.2. Set Partitioning Formulation and Cuts

Let Ω_m be the set of all feasible routes for vehicle type $m \in M$. A set partitioning relaxation with a stronger relaxation can be obtained if one excludes all non-elementary routes, i.e. routes in which some customers are visited more than once, from the formulation. However, the pricing problem would become much harder and probably intractable on large instances. Instead, in this

work we assume that the *ng*-route relaxation (Baldacci et al., 2011), the known route relaxation with better 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 it passes first by another customer v' such that $v \notin N_{v'}$. 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).

Let $\Omega_m^N \subseteq \Omega_m$ be the set of *ng*-routes for vehicle type $m \in M$ with respect to a given set of neighborhoods $N = (N_1, \dots, N_n)$. We denote by c_P the cost of route $P \in \Omega_m^N$, by $x_{(v,v')}^P$ the number of times arc $(v, v') \in A$ participates 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 will be explored 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 for the HFVRPTW considered in this paper is the following.

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

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

$$\sum_{P \in \Omega_m^N} \lambda_P \leq U_m \quad \forall m \in M, \quad (3)$$

$$\lambda_P \in \{0, 1\} \quad \forall m \in M, P \in \Omega_m^N. \quad (4)$$

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 so good. 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:

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

RCCs are known to be a quite effective reinforcement of the SPF on the CVRP (Fukasawa et al., 2006). However, they are not much effective 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 . Remark that 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* (Poggi de Aragão and Uchoa, 2003), such cuts do not have any impact in the complexity 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:

$$\sum_{m \in M} \sum_{P \in \Omega_m^N} \left[\sum_{v \in C} p_v y_v^P \right] \lambda_P \leq \left[\sum_{v \in C} p_v \right]. \quad (6)$$

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 5 rows have been determined by Pecin et al. (2017c)

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

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 $[\sum_{v \in C^\ell} p_v^\ell y_v^P]$ coefficient. 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:

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

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

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

```

 $\alpha \leftarrow 0, \mathcal{S} \leftarrow 0;$ 
for  $j = 1$  to  $k$  do
  if  $(v_{j-1}^P, v_j^P) \notin AM$  then
     $\mathcal{S} \leftarrow 0;$ 
  if  $v_j^P \in C$  then
     $\mathcal{S} \leftarrow \mathcal{S} + p_{v_j^P};$ 
    if  $\mathcal{S} \geq 1$  then
       $\alpha \leftarrow \alpha + 1, \mathcal{S} \leftarrow \mathcal{S} - 1;$ 
return  $\alpha;$ 

```

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 procedure returns a value of α equal to $\lfloor \sum_{v \in C} p_v y_v^P \rfloor$ 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 bounds obtainable with limited memory R1Cs are exactly those 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 RCCs (5), and \mathcal{L} be the current set of active limited memory R1Cs (7). Each cut $j \in \mathcal{J}$ is determined by a set C^j of customers. Each cut $\ell \in \mathcal{L}$ is determined by the triple $(C^\ell, AM^\ell, p^\ell)$. Let (μ, ν, π) be the current dual solution of the linear relaxation of (1)-(4) with sets \mathcal{J} and \mathcal{L} of cuts added, where μ corresponds to partitioning constraints (2), ν corresponds to set \mathcal{J} of active RCCs, and π corresponds to the set \mathcal{L} of active limited memory R1Cs. Let $\bar{c}^m(\mu, \nu)$ be the vector of current arc reduced costs, where

$$\bar{c}_{(v,v')}^m(\mu, \nu) = c_{(v,v')}^m - \frac{\mu_v}{2} - \frac{\mu_{v'}}{2} - \sum_{\substack{j \in \mathcal{J}: \\ (v,v') \in \delta^+(C^j) \cup \delta^-(C^j)}} \nu_j. \quad (8)$$

The pricing subproblem (PSP^m) for vehicle type $m \in M$ is then formulated as

$$z^m(\mu, \nu, \pi) = \min_{P \in \Omega_m^N} \left(\sum_{a \in P} \bar{c}_a^m(\mu, \nu) - \sum_{\ell \in \mathcal{L}} \pi_\ell \cdot \alpha(C^\ell, AM^\ell, p^\ell, P) \right).$$

3. Bucket Graph based Labeling Algorithm

3.1. RCSP over Forward and Backward graphs

We will now formulate the pricing problem defined in Section 2.3 as a RCSP in a forward graph $\vec{G} = (V, \vec{A})$ that is identical to G , so $\vec{A} = A$. In the remaining of Section 3 and in Section 4, we consider the dual solution 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 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 $u_{v, 1} = W$. Consider a forward path \vec{P} in graph \vec{G} as defined by its arcs: $\vec{P} = (\vec{a}_1^{\vec{P}}, \vec{a}_2^{\vec{P}}, \dots, \vec{a}_{|\vec{P}|}^{\vec{P}})$, with $\vec{a}_j^{\vec{P}} = (v_{j-1}^{\vec{P}}, v_j^{\vec{P}})$, $v_0^{\vec{P}} = 0$, $v_{|\vec{P}|}^{\vec{P}} = n + 1$. It yields accumulated consumption of resources $q_{j, r}^{\vec{P}}$ at vertices that can be computed as:

$$q_{j, r}^{\vec{P}} = \begin{cases} l_{0, r} & \text{if } j = 0; \\ \max \left\{ q_{j-1, r}^{\vec{P}} + d_{\vec{a}_j^{\vec{P}}, r}, l_{v_j^{\vec{P}}, r} \right\} & \text{otherwise.} \end{cases} \quad (9)$$

A forward path \vec{P} is feasible if it respects

- the resource consumption bounds:

$$l_{v_j^{\vec{P}}, r} \leq q_{j, r}^{\vec{P}} \leq u_{v_j^{\vec{P}}, r}, \quad \forall j \in \{0, \dots, |\vec{P}|\}, r \in R; \quad (10)$$

- and *ng*-path neighbourhoods:

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

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

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

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: $\min_{\vec{P} \in \vec{\mathcal{P}}} \vec{c}^{\vec{P}}$.

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

For a backward path $\vec{P} = (\vec{a}_1^{\vec{P}}, \vec{a}_2^{\vec{P}}, \dots, \vec{a}_{|\vec{P}|}^{\vec{P}})$ in graph \vec{G} , $\vec{a}_j^{\vec{P}} = (v_{j-1}^{\vec{P}}, v_j^{\vec{P}})$, $v_0^{\vec{P}} = n + 1$, $v_{|\vec{P}|}^{\vec{P}} = 0$, possibly containing cycles, accumulated consumption $q_{j,r}^{\vec{P}}$ of each resource $r \in R$ in \vec{P} is calculated recursively as

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

Feasibility conditions for a backward path \vec{P} are the same as for a forward path. We denote as $\vec{\mathcal{P}}$ the set of all feasible backward paths. Sets $\mathcal{F}_j^{\vec{P}}$, $0 \leq j \leq |\vec{P}|$, 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 corresponding backward path \vec{P} belongs to $\vec{\mathcal{P}}$, where $|\vec{P}| = |\vec{P}|$, $\vec{a}_j^{\vec{P}} = (v, v')$ and $\vec{a}_{|\vec{P}|+1-j}^{\vec{P}} = (v', v)$, $1 \leq j \leq |\vec{P}|$. Also, we have $q_{j,r}^{\vec{P}} \leq q_{|\vec{P}|-j,r}^{\vec{P}}$ for $0 \leq j \leq |\vec{P}|$ and $r \in R$. 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 \vec{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 Alpha. Therefore the pricing problem defined in Section 2.3 can be also reformulated as finding a path in $\vec{\mathcal{P}}$ minimizing the total reduced cost.

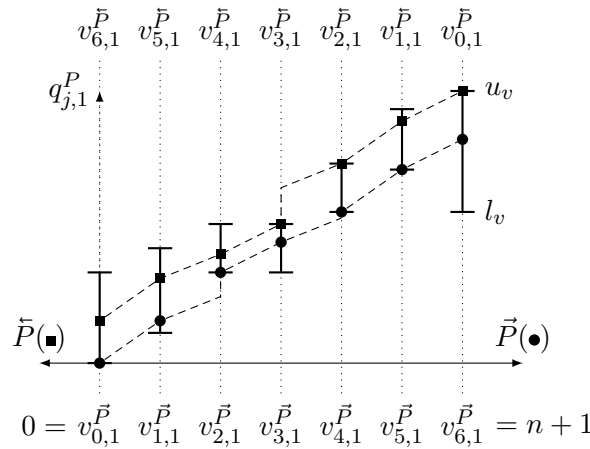


Figure 1: An example of a forward and the corresponding backward paths in the case of one resource

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

use accent $\tilde{\sigma}$ for the opposite sense entity when there is a bijection between forward and backward entities, as for arcs and paths.

3.2. Labels and Dominance Rule

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

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

$$\bar{c}^L - \sum_{\ell \in \mathcal{L}: \mathcal{S}_\ell^L > \mathcal{S}_\ell^{L'}} \pi_\ell \leq \bar{c}^{L'}, \quad (12)$$

as $\pi_\ell \leq 0$, $\ell \in \mathcal{L}$. Note that $\bar{c}^L > \bar{c}^{L'}$ is a sufficient condition to verify that L does not dominate L' ,

3.3. Bucket Graph

A critical aspect of labeling algorithms that solve the previously described RCSP 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. An usual approach to address this issue is partitioning the labels into buckets where it is ensured that no pair of labels dominate each other. Additionally, dominance checks may be performed less frequently between labels stored in different buckets.

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 its 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 performed 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 final vertex is determined by a step size \tilde{d}_r for each resource $r \in R$. A forward bucket \vec{b} corresponds to a pair $(\vec{v}_{\vec{b}}, \{\vec{l}_{\vec{b}, r}\}_{r \in R})$, where $\vec{v}_{\vec{b}} \in V$ and

$\tilde{l}_{\vec{b},r} = l_{\tilde{v}_{\vec{b}},r} + \kappa_{\vec{b},r} \cdot \tilde{d}_r \leq u_{\tilde{v}_{\vec{b}},r}$, $\kappa_{\vec{b}} \in \mathbb{Z}_+^2$. A forward label \vec{L} is contained in forward bucket \vec{b} if $v^{\vec{L}} = \tilde{v}_{\vec{b}}$ and $\tilde{l}_{\vec{b},r} \leq q_r^{\vec{L}} < \tilde{l}_{\vec{b},r} + \tilde{d}_r$ for all $r \in R$. A backward bucket \vec{b} corresponds to a pair $(\tilde{v}_{\vec{b}}, \{\tilde{u}_{\vec{b},r}\}_{r \in R})$, where $\tilde{v}_{\vec{b}} \in V$ and $\tilde{u}_{\vec{b},r} = u_{\tilde{v}_{\vec{b}},r} - \kappa_{\vec{b},r} \cdot \tilde{d}_r \geq l_{\tilde{v}_{\vec{b}},r}$, $\kappa_{\vec{b}} \in \mathbb{Z}_+^2$. A backward label \vec{L} is contained in backward bucket \vec{b} if $v^{\vec{L}} = \tilde{v}_{\vec{b}}$ and $\tilde{u}_{\vec{b},r} \geq q_r^{\vec{L}} > \tilde{u}_{\vec{b},r} - \tilde{d}_r$ for all $r \in R$. Let b^L be the bucket containing label L .

We will say that bucket b' is *component-wise smaller* than bucket b (denoted as $b' \prec b$) if buckets are of the same sense, $v_{b'} = v_b$, and $\kappa_{b'} \leq \kappa_b$.

Let \vec{B} (\vec{B}) be the set of all forward (backward) buckets. We define as $\vec{\Gamma}$ ($\vec{\Gamma}$) the set of directed forward (backward) *bucket arcs*, each one connecting two buckets. Each forward/backward bucket arc $\gamma \in \Gamma$ is associated with a bucket $b_\gamma \in B$ and an arc $a_\gamma = (\tilde{v}_{b_\gamma}, v') \in A$ of the corresponding sense. Forward bucket arc $\vec{\gamma}$ connects bucket $\vec{b}_{\vec{\gamma}}$ with bucket $\vec{b}' \in \vec{B}$ such that $\tilde{v}_{\vec{b}'}$ is the head of $\vec{a}_{\vec{\gamma}}$ and $\tilde{l}_{\vec{b}',r} \leq \tilde{l}_{\vec{b}_{\vec{\gamma}},r} + d_{\vec{a}_{\vec{\gamma}},r} < \tilde{l}_{\vec{b}_{\vec{\gamma}},r} + \tilde{d}_r$ for all $r \in R$, and $\vec{\gamma} \in \vec{\Gamma}$ if $\tilde{l}_{\vec{b}_{\vec{\gamma}},r} + d_{\vec{a}_{\vec{\gamma}},r} \leq u_{\tilde{v}_{\vec{b}'},r}$ for all $r \in R$. Backward bucket arc $\vec{\gamma}$ connects bucket $\vec{b}_{\vec{\gamma}}$ with bucket $\vec{b}' \in \vec{B}$ such that $\tilde{v}_{\vec{b}'}$ is the head of $\vec{a}_{\vec{\gamma}}$ and $\tilde{u}_{\vec{b}',r} \geq \tilde{u}_{\vec{b}_{\vec{\gamma}},r} - d_{\vec{a}_{\vec{\gamma}},r} > \tilde{u}_{\vec{b}_{\vec{\gamma}},r} - \tilde{d}_r$ for all $r \in R$, and $\vec{\gamma} \in \vec{\Gamma}$ if $\tilde{u}_{\vec{b}_{\vec{\gamma}},r} - d_{\vec{a}_{\vec{\gamma}},r} \geq l_{\tilde{v}_{\vec{b}'},r}$ for all $r \in R$.

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

$$\Phi_b = \{b' : \tilde{v}_{b'} = \tilde{v}_b, \exists r' \in R, \kappa_{b',r'} = \kappa_{b,r'} - 1, \kappa_{b',r} = \kappa_{b,r}, \forall r \in R \setminus \{r'\}\}.$$

As $|R| = 2$, we have $|\Phi_b| \leq 2$. By definition, we have $b' \prec b$ for all $b' \in \Phi_b$. Let $\vec{\Gamma}^\Phi$ ($\vec{\Gamma}^\Phi$) be the set of directed forward (backward) bucket arcs (b', b) such that $b \in B$, $b' \in \Phi_b$.

We are ready to define the directed forward and backward bucket graphs $\vec{B} = (\vec{B}, \vec{\Gamma} \cup \vec{\Gamma}^\Phi)$ and $\vec{B} = (\vec{B}, \vec{\Gamma} \cup \vec{\Gamma}^\Phi)$. In each of these graphs, we find the set of strongly connected components and a topological order for it. Let \mathcal{C}_b be the strongly connected component of a forward/backward bucket b . We remark that bucket arcs in Γ^Φ are not used in the labeling algorithm below, they are needed only to compute the set of connected components of the bucket graph and an appropriate topological order for it.

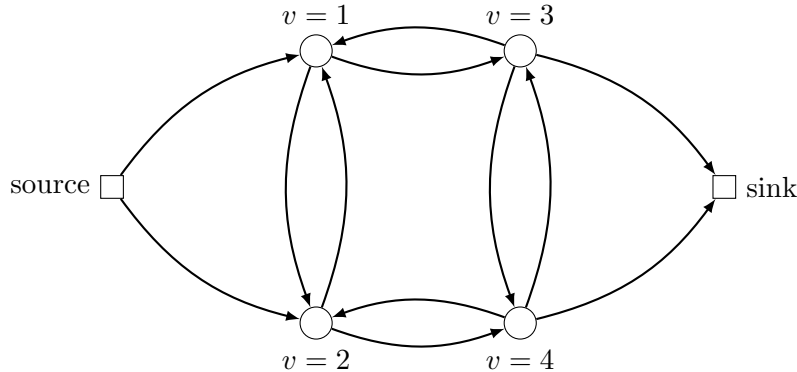


Figure 2: An example of Forward Graph

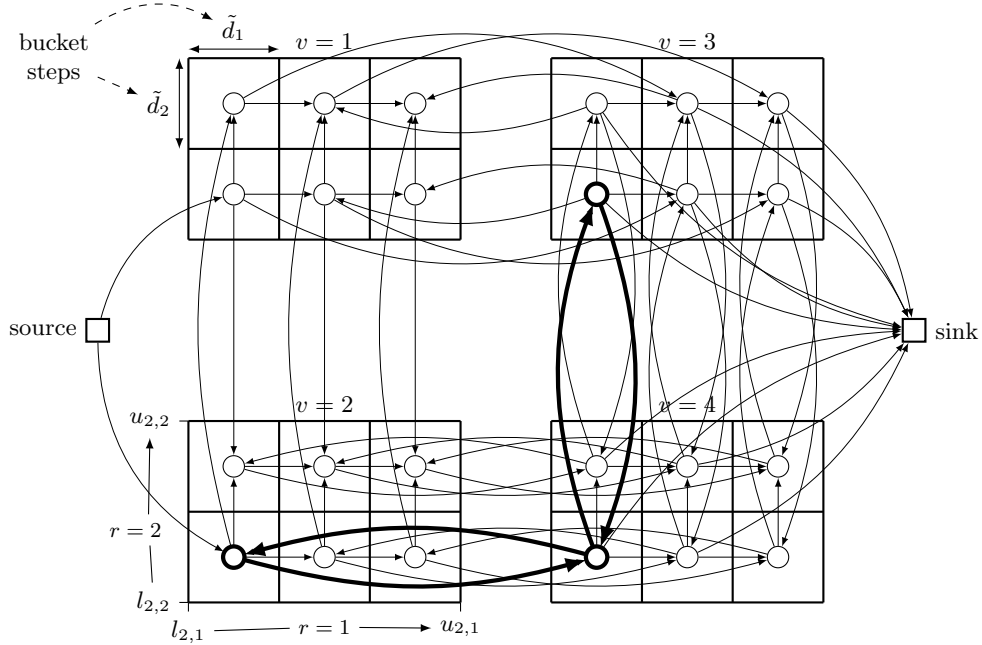


Figure 3: An example of Bucket Graph for the forward graph of Figure 2

Figure 3 shows a small 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 resources 1 and 2 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 bucket graph is depicted inside the corresponding square. Note that 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 bucket graph is bold printed in the figure.

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. In this algorithm, we maintain values \bar{c}_b^{best} equal to the smallest reduced cost of labels L such that $b^L \preceq b$. For the mono-directional labeling algorithm, we need two auxiliary functions presented below. Function `DominatedInCompWiseSmallerBuckets(L, b)` checks whether a label L is dominated by a label contained in a bucket b' such that $b' \preceq b$. 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. Before calling this function, we unmark all buckets in B . Function `Extend(L', γ, L)` 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 follow that the steps performed by this

function initialize the reduced cost and the final vertex of the new label, calculates its resource consumptions and checks their bounds, obtains its *ng*-route information and checks its feasibility, and, finally, updates its reduced cost considering the active R1Cs.

Function DominatedInCompWiseSmallerBuckets(L, b)

```

mark bucket  $b$ 
if  $C_b$  precedes  $C_{b^L}$  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 non-marked  $b' \in \Phi_b$  do
   $\lfloor$  if DominatedInLexSmallerBuckets( $L, b'$ ) then return true
return false
  
```

Function Extend(L', γ, L)

```

 $\bar{c}^L \leftarrow \bar{c}^{L'} + \bar{c}_{a_\gamma}, v^L \leftarrow$  the head of  $a_\gamma$ 
for  $r \in R$  do
   $q_r^L \leftarrow q_r^{L'}$ 
  if  $L$  is a forward label then
     $q_r^L \leftarrow \max\{q_r^L + d_{a_\gamma, r}, l_{v^L, r}\}$ 
    if  $q_r^L > u_{v^L, r}$  then return false
  if  $L$  is a backward label then
     $q_r^L \leftarrow \min\{q_r^L - d_{a_\gamma, r}, u_{v^L, r}\}$ 
    if  $q_r^L < l_{v^L, r}$  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  $S_\ell^L \leftarrow S_\ell^{L'}$ 
  else  $S_\ell^L \leftarrow 0$ 
  if  $v^L \in C^\ell$  then
     $S_\ell^L \leftarrow S_\ell^L + p_\ell$ 
    if  $S_\ell^L \geq 1$  then
       $S_\ell^L \leftarrow S_\ell^L - 1, \bar{c}^L \leftarrow \bar{c}^L - \pi_\ell$ 
return true
  
```

The labeling algorithm is presented in Algorithm 1. It can be used in both forward and backward sense.

3.5. Bi-directional labeling algorithm

We will call a pair of forward and backward labels \vec{L} and \bar{L} ω -compatible if

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

Algorithm 1: Mono-directional labeling algorithm

```

if forward algorithm then  $L^{init} \leftarrow (0, 0, \{l_{0,r}\}_{r \in R}, \emptyset, \mathbf{0})$ 
if backward algorithm then  $L^{init} \leftarrow (0, n + 1, \{u_{n+1,r}\}_{r \in R}, \emptyset, \mathbf{0})$ 
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  $\mathcal{B}$  in topological order do
  repeat
    foreach bucket  $b \in \mathcal{C}$  do
      foreach non-extended label  $L' : b^{L'} = b$  do
        unmark all buckets in  $\mathcal{B}$ 
        if not DominatedInCompWiseSmallerBuckets( $L', b^{L'}$ ) then
          foreach bucket arc  $\gamma \in \Gamma$  such that  $b_\gamma = b^{L'}$  do
            if Extend( $L', \gamma, L$ ) 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 non-decreasing order of  $\kappa_b$  do
           $\bar{c}_b^{best} \leftarrow \min \{ \min_{L: b^L=b} \{ \bar{c}^L \}, \min_{b' \in \Phi_b} \{ \bar{c}_{b'}^{best} \} \}$ 
      return  $P^{best} = \operatorname{argmin}_{P^L: v^L=v'} c^L$ , where  $v'$  is the sink in  $G$ 

```

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

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

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

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

```

mark bucket  $\vec{b}$ 
if  $\bar{c}_{\vec{L}} + \bar{c}_{(v^{\vec{L}}, v^{\vec{L}})} + \bar{c}_{\vec{b}}^{best} \geq c_{P^{best}}$  then return
foreach label  $\vec{L} : b_{\vec{L}} = \vec{b}$  do
  if pair  $(\vec{L}, \vec{L})$  is  $c_{P^{best}}$ -compatible then
     $P^{best} \leftarrow (P_{\vec{L}}, \vec{a} = (v^{\vec{L}}, v^{\vec{L}}), P_{\vec{L}})$ 
  foreach non-marked  $\vec{b}' \in \Phi_{\vec{b}}$  do
    ConcatenateLabel( $\vec{L}, \vec{b}', P^{best}$ )
return

```

The auxiliary procedure ConcatenateLabel($\vec{L}, \vec{b}, P^{best}$), presented below, tries to find, in a

“branch-and-bound manner”, a backward label \tilde{L} such that $\tilde{b}^{\tilde{L}} \preceq \tilde{b}$, and such that pair (\vec{L}, \tilde{L}) is $\bar{c}^{P^{best}}$ -compatible, i.e. concatenation of partial paths $P^{\vec{L}}$ and $P^{\tilde{L}}$ along arc $(v^{\vec{L}}, v^{\tilde{L}})$ improves on P^{best} . We use values \bar{c}^{best} as bounds: from the reasoning of the previous paragraph, if $\bar{c}^{\vec{L}} + \bar{c}_{(v^{\vec{L}}, \tilde{v}_{\tilde{b}})} + \bar{c}_{\tilde{b}}^{best} \geq \bar{c}^{P^{best}}$ then such a label does not exist. For this reason, in the literature, values $\bar{c}_{\tilde{b}}^{best}$ are referred as *completion bounds*.

Algorithm 2: Bi-directional labeling algorithm

run forward labeling algorithm where only labels \vec{L} , $q_r^{\vec{L}} \leq q^*$ are kept
run backward labeling algorithm where only labels \tilde{L} , $q_r^{\tilde{L}} > q^*$, 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** $q_r^{\vec{L}'} > q^*$ **then**

 unmark all backward buckets

 let $\tilde{b} \in \tilde{B}$ be the bucket such that $\tilde{u}_{\tilde{b}, r} \geq q_r^{\vec{L}'} > \tilde{u}_{\tilde{b}, r} - \tilde{d}_r, \forall r \in R$

 ConcatenateLabel($\vec{L}, \tilde{b}, P^{best}$)

return P^{best}

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.

3.6. Symmetric case

In this section, we suppose that 1) all time windows are the same; and 2) R1Cs arc memory is 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.

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) , $v, v' \in V$, becomes the same. Moreover, as for any route P coefficients $x_{(v, v')}^P$ and $x_{(v', v)}^P$ are the same in constraints (2) and cuts (5), reduced costs of these arcs is the same:

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

Consider now a backward label \tilde{L} and the corresponding partial path $\vec{P} = \vec{P}^{\tilde{L}}$. Let \vec{P} be the partial forward path such that $v_j^{\vec{P}} = v_j^{\tilde{P}}$, $1 \leq j \leq |\vec{P}|$, 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 $\bar{c}^{\vec{L}} = \bar{c}^{\tilde{L}}$, $\mathcal{F}^{\vec{L}} = \mathcal{F}^{\tilde{L}}$, and $\mathcal{S}^{\vec{L}} = \mathcal{S}^{\tilde{L}}$. As time windows are the same, for each $r \in R$, resource bounds are also the same: $[l_{v, r}, u_{v, r}] = [l_r, u_r]$ for all $v \in V$. Then $q_r^{\vec{L}} = u_r - q_r^{\tilde{L}}$ for all $r \in R$. Moreover, for every backward bucket \tilde{b} , there exists a symmetric forward bucket \vec{b} such that $\tilde{l}_{\vec{b}, r} = u_r - \tilde{u}_{\tilde{b}, r}$ and $\kappa_{\vec{b}} = \kappa_{\tilde{b}}$.

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

4. Reduced Cost Fixing of Bucket Arcs

We will 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 smaller cost 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 optimum solution value of (SPF). Note that improving paths must be elementary. From the solution space of the pricing problem we can exclude non-improving paths.

Let (μ, ν, π) be the current dual solution of the linear relaxation of (SPF), corresponding to constraints (2), (5), (6) and $z^{(\text{SPF})}$ be its value. Recall that $z^m(\mu, \nu, \pi)$ is the solution value returned by pricing subproblem (PSP m). Then, as long as

$$z^m(\mu, \nu, \pi) \leq \bar{c}^P, \quad \forall m \in M, \forall \text{ improving } P \in \Omega_m, \quad (14)$$

the value

$$LB = z^{(\text{SPF})} + \sum_{m' \in M} z^{m'}(\mu, \nu, \pi) \cdot U_{m'}. \quad (15)$$

is a valid (Lagrangian) lower bound on the optimum solution.

A path $P \in \Omega_m$ is improving only if $LB - z^m(\mu, \nu, \pi) + \bar{c}^P \leq UB$. We denote as θ_m the current primal-dual gap disregarding the reduced cost of one vehicle of type m : $\theta_m = UB - LB + z^m(\mu, \nu, \pi)$. Then, a sufficient condition for a forward/backward path $P \in \Omega_m$ to be non-improving is $\bar{c}^P \geq \theta_m$. In the remaining of this section, we again drop index m 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 $a_j^P = a_\gamma$, and $\tilde{l}_{b_\gamma, r} \leq q_{j-1, r}^P < \tilde{l}_{b_\gamma, r} + \tilde{d}_r$ for a forward path ($\tilde{u}_{b_\gamma, r} \geq q_{j-1, r}^P > \tilde{u}_{b_\gamma, r} - \tilde{d}_r$ for a backward path), for all $r \in R$. We will denote such bucket arc as γ_j^P .

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 reduced cost fixing procedure tries to reduce current set Γ by removing some non-improving bucket arcs from it. However, executing the proposed labeling algorithm over this reduced bucket graph may result in a violation of condition (14) and thus the correctness of the algorithm may be lost.

Figure 4 illustrates a case where it occurs. 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 paths, respectively. Recall that such paths are defined over the forward graph. Thus, for each traversed arc, the corresponding bucket arc that is passed through depends solely on the total resource consumed up to this point. Note that P

and P' traverse different arcs until vertex $v_{i-1}^P = v_{i-1}^{P'}$ and finish with the same subpath from this point until the sink vertex (but with such subpath passing through different bucket arcs). Now, assume that, at some iteration of the column generation, P' has a large reduced cost that allows the bucket arc corresponding to $(v_{i-1}^{P'}, v_i^{P'})$ to be removed, and later, at another iteration, P' has a reduced cost smaller than that of P , allowing the label L associated to P' until vertex $v_{i-1}^{P'}$ 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, the previous bucket arc removal prevents P' from being found by the labeling algorithm. Thus, condition (14) may be violated, as neither P nor P' are obtained by the labeling algorithm.

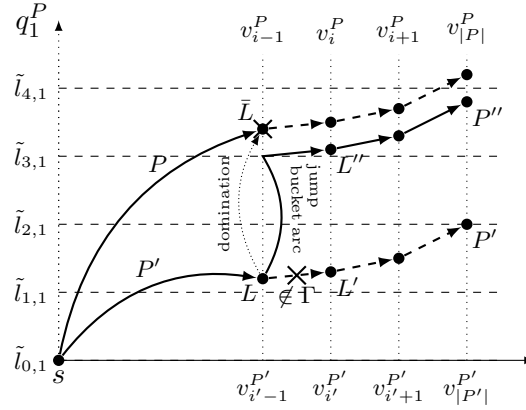


Figure 4: Illustration of the proof of Proposition 1

To fix this problem, we propose the insertion of *jump bucket arcs* into the bucket graph. Jump bucket arcs impose additional resource consumptions when traversed that allow to skip removed bucket arcs. Let $\vec{\Psi}$ ($\vec{\Psi}$) be the current set of forward (backward) jump bucket arcs. Each forward/backward jump bucket arc $\psi \in \Psi$ is associated with a base bucket b_ψ^{base} , a jump bucket b_ψ^{jump} , $\tilde{v}_{b_\psi^{\text{base}}} = \tilde{v}_{b_\psi^{\text{jump}}} = \tilde{v}$, $b_\psi^{\text{base}} \prec b_\psi^{\text{jump}}$, and an arc $a_\psi = (\tilde{v}, v') \in A$ 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 first line of function $\text{Extend}(L, \psi, L')$, instead of $q^L \leftarrow q^{L'}$, we do $q_r^L \leftarrow \max \{q_r^{L'}, \tilde{l}_{b_\psi^{\text{jump}}, r}\}$ ($q_r^L \leftarrow \min \{q_r^{L'}, \tilde{u}_{b_\psi^{\text{jump}}, r}\}$ for a backward label), for all $r \in R$. Function $\text{ObtainJumpBucketArcs}(\Gamma)$ presented below obtains the current set Ψ of jump bucket arcs given the set Γ of current bucket arcs (which could not be shown to be non-improving). This function can be used for both forward and backward sense.

We denote as L_j^P the label corresponding to the partial path consisting of first j arcs of path P . Note that 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 $\vec{\bar{L}}(\Gamma, \Psi)$ and $\vec{\bar{L}}(\Gamma, \Psi)$ be the sets of non-dominated labels generated by the 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

Function ObtainJumpBucketArcs(Γ)

```

 $\Psi \leftarrow \emptyset$ 
foreach bucket  $b \in B$  do
  foreach  $a = (v, v') \in A$  such that  $v = \tilde{v}_b$  do
    if  $\exists \gamma \in \Gamma$  such that  $b_\gamma = b, a_\gamma = a$  then
       $\bar{B} \leftarrow \{b' \succ b : \exists \gamma \in \Gamma : b_\gamma = b', a_\gamma = a\}$ 
      remove from  $\bar{B}$  all non component-wise minimal buckets
      foreach  $b' \in \bar{B}$  do
         $\Psi \leftarrow \Psi \cup \{\psi\}$ , where  $b_\psi^{\text{base}} = b, b_\psi^{\text{jump}} = b', a_\psi = a$ 
  return  $\Psi$ 

```

$$b_\psi^{\text{base}} = b^L.$$

The next proposition is auxiliary (but very important) and will be used to prove a sufficient condition for a bucket arc to be non-improving. This proposition applies for both forward and backward sense. Its proof is illustrated in Figure 4.

Proposition 1. *Let set Γ contain 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 $b^{L^{\text{init}}}$ that corresponds to vertex v_0^P . Now, assume that the proposition is true for $j = i-1, 1 \leq i \leq |P|-1$. 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, note that $b^L \preceq b^{L_{i-1}^P}$. 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, if $\gamma' \notin \Gamma$, since γ is an improving arc and $b^L \prec b^\gamma$, there should exist a jump bucket arc $\psi \in \Psi$ such that $a_\psi = a_i^P, b_\psi^{\text{base}} = b^L$, and $b_\psi^{\text{jump}} \preceq b^{L_i^P}$. Thus, extending L through a_ψ should lead 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 should also dominate L_i^P , finishing the proof. \square

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

For a (jump) bucket arc $\gamma \in \Gamma$ ($\psi \in \Psi$), we will 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 $\vec{b}_{\vec{\gamma}}^{\text{arr}}$ is the backward bucket \vec{b} such that $\tilde{u}_{\vec{b},r} \geq \tilde{l}_{\vec{b},r} + d_{\vec{a}_{\vec{\gamma}},r} > \tilde{u}_{\vec{b},r} - \tilde{d}_r, \forall r \in R$. For a backward bucket arc $\overleftarrow{\gamma} \in \overleftarrow{\Gamma}$, its arrival bucket $\overleftarrow{b}_{\overleftarrow{\gamma}}^{\text{arr}}$ is the forward bucket \vec{b} such that $\tilde{l}_{\vec{b},r} \leq \tilde{u}_{\vec{b},r} - d_{\vec{a}_{\overleftarrow{\gamma}},r} < \tilde{l}_{\vec{b},r} + \tilde{d}_r, \forall r \in R$. For a jump bucket arc $\psi \in \Psi$, in the definition of its forward/backward arrival buckets, b_γ and a_γ are replaced by b_ψ^{jump} and a_ψ .

The next proposition gives a necessary condition for a bucket arc to be improving. It also applies for both forward and backward sense.

Proposition 2. *Let set Γ contain all improving bucket arcs, and Ψ be the set of jump bucket arcs computed using procedure ObtainJumpBucketArcs(Γ). If $\gamma \in \Gamma$ is an improving bucket arc then there exists a pair of θ -compatible labels $L \in \mathbf{L}$ and $\vec{L} \in \vec{\mathbf{L}}$ such that $b^L \preceq b_\gamma$ and $\vec{b}^{\vec{L}} \preceq \vec{b}_\gamma^{\text{arr}}$, where $\mathbf{L} = \mathbf{L}(\Gamma, \Psi)$, and $\vec{\mathbf{L}} = \vec{\mathbf{L}}(\Gamma, \Psi)$.*

Proof: If γ is an improving bucket arc then there exists an improving path P passing through γ and $\bar{c}^P < \theta$. Then there exists an index j , $0 \leq j < |P|$, such that $b_\gamma = b^{L_j^P}$ and $a_\gamma = a_{j+1}^P$. As $\bar{c}^P < \theta$, by Observation 1, pair of labels L_j^P and $\vec{L}_{|P|-j-1}^{\vec{P}}$ is θ -compatible. If label $L_j^P \in \mathbf{L}$, let $L = L_j^P$, otherwise by Proposition 1 there should exist label $L \in \mathbf{L}$ dominating L_j^P such that $b^L \preceq b^{L_j^P} = b_\gamma$. If label $\vec{L}_{|P|-j-1}^{\vec{P}} \in \vec{\mathbf{L}}$, let $\vec{L} = \vec{L}_{|P|-j-1}^{\vec{P}}$, otherwise by Proposition 1 there should exist label $\vec{L} \in \vec{\mathbf{L}}$ dominating $\vec{L}_{|P|-j-1}^{\vec{P}}$ such that $\vec{b}^{\vec{L}} \preceq \vec{b}_{|P|-j-1}^{\vec{P}} \preceq \vec{b}_\gamma^{\text{arr}}$. From definitions of dominance and θ -compatible labels, labels L and \vec{L} are also θ -compatible. \square

Before the reduced cost fixing procedure we first define an auxiliary function UpdateBucketsSet($\theta, L, \vec{B}, \vec{b}, \vec{\mathbf{L}}$) which adds to set \vec{B} all opposite sense buckets \vec{b}' , $\vec{b}' \preceq \vec{b}$, containing at least one label $\vec{L} \in \vec{\mathbf{L}}$ such that pair (L, \vec{L}) of labels is θ -compatible. Note that buckets already contained in \vec{B} are not removed and the function returns true only if at least one bucket has been inserted in \vec{B} .

Below we present the main procedure ReducedCostFixing($\theta, \Gamma, \Psi, \mathbf{L}, \vec{\mathbf{L}}$) which is used to find non-improving bucket arcs in Γ , given set \mathbf{L} of non-dominated labels of the same sense as Γ and set $\vec{\mathbf{L}}$ of non-dominated labels of the opposite sense. There are forward and backward variants of this procedure.

Note that, by Proposition 2, for a bucket arc γ to be considered as non-improving, there should exist no pair of θ -compatible labels $L \in \mathbf{L}$ and $\vec{L} \in \vec{\mathbf{L}}$ such that $b^L \preceq b_\gamma$ and $\vec{b}^{\vec{L}} \preceq \vec{b}_\gamma^{\text{arr}}$. Since buckets b_γ are visited in the topological order of the strongly connected components of \mathcal{B} , the procedure uses the results from the previous checks of buckets $b' \prec b_\gamma$ to find labels not in b_γ that may prevent γ to be fixed. After processing each strongly connected component \mathcal{C} of \mathcal{B} , set $\vec{B}_{a,b}$, $a = (\tilde{v}_b, v') \in A$, $b \in \mathcal{C}$, contains all opposite sense buckets \vec{b} , $\tilde{v}_{\vec{b}} = v'$, having at least one label $\vec{L} \in \vec{\mathbf{L}}$ such that pair of labels (L, \vec{L}) is θ -compatible, where $L \in \mathbf{L}$, $b^L \preceq b$. Thus, initially checking all opposite sense buckets contained in $\cup_{b' \in \Phi_b} \vec{B}_{a_\gamma, b'}$ (where b' is necessarily contained in

Function UpdateBucketsSet($\theta, L, \vec{B}, \vec{b}, \vec{L}$)

```

mark bucket  $\vec{b}$ 
if  $\bar{c}^L + \bar{c}_{v^L, \vec{v}_{\vec{b}}} + \bar{c}_{\vec{b}}^{best} \geq \theta$  then return false
 $updated \leftarrow$  false
if  $\vec{b} \notin \vec{B}$  then
    foreach label  $\vec{L} \in \vec{L}, \vec{b}^{\vec{L}} = \vec{b}$  do
        if pair  $(L, \vec{L})$  is  $\theta$ -compatible then
             $\vec{B} \leftarrow \vec{B} \cup \{\vec{b}\}$ 
             $updated \leftarrow$  true
    foreach non-marked  $\vec{b}' \in \Phi_{\vec{b}}$  do
        if UpdateBucketsSet( $\theta, L, \vec{B}, \vec{b}', \vec{L}$ ) then
             $updated \leftarrow$  true
return  $updated$ 

```

a previously visited strongly connected component) is enough to ensure that any pair of labels (L, \vec{L}) with $b^L \prec b_\gamma$ preventing γ to be fixed will be found. After that, it remains only to check the bucket b_γ itself, which is done by calling function UpdateBucketsSet.

In the next proposition, we finally show that condition (14) is satisfied. For it, we define as $\bar{c}^*(\vec{\Gamma}, \vec{\Gamma}, \vec{\Psi}, \vec{\Psi})$ the minimum reduced cost of the best path obtained by the 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 $\vec{\Gamma}$ and $\vec{\Psi}$.

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

Proof: Consider an arbitrary improving path $\vec{P} \in \vec{\mathcal{P}}$. Let $j, 0 \leq j \leq |\vec{P}|$ be the largest index such that $q_{r^*}^{\vec{L}_j^{\vec{P}}} \leq q^*$.

Suppose that $j = |\vec{P}|$. Let $\vec{L} = \vec{L}_j^{\vec{P}}$. We have $v^{\vec{L}} = n + 1$. As \vec{P} is improving, by Proposition 1, either $\vec{L} \in \vec{L}(\vec{\Gamma}, \vec{\Psi})$ or there is a label $\vec{L}' \in \vec{L}(\vec{\Gamma}, \vec{\Psi})$ 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 $\bar{c}^* \leq \bar{c}^{\vec{P}}$.

Suppose now that $j < |\vec{P}|$. Let \vec{P} be the corresponding backward path for \vec{P} . By Observation 1, the pair of labels $\vec{L}_j^{\vec{P}}$ and $\vec{L}_{|\vec{P}|-j-1}^{\vec{P}}$ is ω -compatible for all $\omega > \bar{c}^{\vec{P}}$. As both \vec{P} and \vec{P} are improving, by Proposition 1 there should exist a pair of labels $\vec{L} \in \vec{L}(\vec{\Gamma}, \vec{\Psi})$ and $\vec{L} \in \vec{L}(\vec{\Gamma}, \vec{\Psi})$ which is also ω -compatible for all $\omega > \bar{c}^{\vec{P}}$. Then $\bar{c}(\vec{P}^{\vec{L}} || \vec{P}^{\vec{L}}) \leq \bar{c}^{\vec{P}}$, thus $\bar{c}^* \leq \bar{c}^{\vec{P}}$. \square

Procedure ReducedCostFixing($\theta, \Gamma, \Psi, \mathbf{L}, \vec{\mathbf{L}}$)

```

foreach strongly connected component  $\mathcal{C}$  in  $\mathcal{B}$  in topological order do
  foreach bucket  $b \in \mathcal{C}$  in non-decreasing order of  $\kappa_b$  do
    foreach arc  $a = (\tilde{v}_b, v') \in A$  do  $\vec{B}_{a,b} \leftarrow \cup_{b' \in \Phi_b} \vec{B}_{a,b'}$ 
    foreach jump bucket arc  $\psi \in \Psi$  such that  $b_\psi^{\text{base}} = b$  do
      foreach  $L \in \mathbf{L}$  such that  $b^L = b$  do
        unmark all opposite sense buckets
        if UpdateBucketsSet( $\theta, L, \vec{B}_{a_\psi, b}, \vec{b}_\psi^{\text{arr}}, \vec{\mathbf{L}}$ ) then
           $toRemove \leftarrow \text{false}$ 
      foreach bucket arc  $\gamma \in \Gamma$  such that  $b_\gamma = b$  do
        if  $\exists \vec{b}' \in \vec{B}_{a_\gamma, b} : \vec{b}' \preceq \vec{b}_\gamma^{\text{arr}}$  then  $toRemove \leftarrow \text{false}$ 
        else  $toRemove \leftarrow \text{true}$ 
        foreach  $L \in \mathbf{L}$  such that  $b^L = b$  do
          unmark all opposite sense buckets
          if UpdateBucketsSet( $\theta, L, \vec{B}_{a_\gamma, b}, \vec{b}_\gamma^{\text{arr}}, \vec{\mathbf{L}}$ ) then
             $toRemove \leftarrow \text{false}$ 
        if  $toRemove$  then  $\Gamma \leftarrow \Gamma \setminus \{\gamma\}$ 

```

5. Branch-Cut-and-Price algorithm

We constructed a Branch-Cut-and-Price algorithm for the HFVRPTW using the new bucket graph labeling algorithm in its pricing and also the corresponding reduced cost fixing 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 et al. (2017) and automatic dual price smoothing stabilization by Pessoa et al. (2017) 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. (2017). 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 χ^{max} , we clean them up and leave only $\chi^{\text{perc}}\%$ columns with the least reduced cost.

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. Note that AM is then 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 the branching is performed. Note that, 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 et al. (2017) which is inspired by Roberti and Mingozzi (2014). We start by generating ng -neighbourhood for each customer by including there η^{init} closest customers (including itself). After the convergence of column generation, we augment neighbourhoods 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 neighbourhood of any customer cannot exceed size η^{max} . The same tailing-off parameters are applied here as for cutting planes. We start the root node by augmenting the ng -neighbourhoods. Then, after tailing-off, RCCs are separated. After another tailing-off, R1Cs are separated.

Each additional active inequality of (7) or ng -neighbourhood 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).

Reduced cost fixing of bucket arcs is performed after the initial convergence, and also each time the current primal-dual gap decreased by at least $\sigma\%$ since the last call to reduced cost

fixing. 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 (SPF) relaxation is updated by excluding non-elementary columns coming from this subproblem. The pricing subproblem in the enumerated state is solved by inspection. Reduced cost fixing is also performed for enumerated pricing subproblems, but considering each route individually. If after some point, the number of enumerated routes for each subproblem fall below ω^{MIP} , the residual problem is handled by a standard MIP solver, and the node is considered as processed.

Remember that to perform reduced cost fixing, the forward and backward passes of the labeling algorithm should 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 reduced cost fixing, 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 \tilde{L} such that L and \tilde{L} are θ -compatible. We keep label L only if such label \tilde{L} exists. A similar approach is used in the bi-directional enumeration procedure. Note that 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:

- $z^m = \sum_{P \in \Omega_m} \sum_{v' \in V'} \frac{1}{2} (x_{(0,v')}^P + x_{(v',n+1)}^P) \lambda_P, m \in M;$
- $z_v^m = \sum_{P \in \Omega_m} y_v^P \lambda_P, v \in V, m \in M;$
- $z_{\{v,v'\}} = \sum_{m \in M} \sum_{P \in \Omega_m} (x_{(v,v')}^P + x_{(v',v)}^P) \lambda_P, v, v' \in V, v \neq v'.$

z^m corresponds to the number of used vehicles of type $m \in M$; z_v^m characterises whether customer $v \in V$ is served by a vehicle of type m ; $z_{\{v,v'\}}$ characterises whether edge (v, v') is used in the solution. Given a fractional branching variable z with fractional value f , two child nodes are created by adding the constraints $z \leq \lfloor f \rfloor$ and $z \geq \lceil f \rceil$ to the master LP. Note that a generic branching constraint $\sum_{m \in M} \sum_{P \in \Omega_m} \sum_{a \in A} \alpha_a^m x_a^P \lambda_P \geq \beta$ added to the master LP with associated dual variable π can be considered by the pricing algorithm by subtracting $\pi \alpha_a^m$ from the expression of $\bar{c}_a^m(\mu, \nu)$ 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 $x_{(v',v)}^P$ and $x_{(v,v')}^P$ are the same in all branching constrains we use.

The selection of the branching variable on 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 in 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 best value of $\Delta LB_1 \times \Delta LB_2$, where ΔLB_i denotes the increase in the current lower bound obtained for the i th child node, for $i = 1, 2$.

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 selected the same way as in the phase 1.

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 branch-cut-and-price algorithm described in the previous section was coded in C++ and compiled with GCC 5.3.0. BaPCod package by Vanderbeck et al. (2017) was used to handle the Branch-Cut-and-Price framework. IBM CPLEX Optimizer version 12.6.0 was used as the LP solver in column generation and as the IP solver to solve 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 8 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 7 classic instances from Christofides et al. (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 et al. (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).

HFVRP instances For the heterogeneous fleet vehicle routing problem, we used 96 instances by Duhamel et al. (2011) with 20–256 customers. Each instance represent one of 96 French departments. The best known solutions are taken from Penna et al. (2017). Following the convention used in those papers, we assume 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 $\chi^{\text{heur}} = 30$, $\chi^{\text{exact}} = 150$, $\chi^{\text{max}} = 10'000$, $\chi^{\text{perc}} = 66\%$. 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 $\eta^{\text{init}} = 8$, $\eta^1 = 5$, $\eta^2 = 100$, $\eta^{\text{max}} = 16$. Reduced cost fixing parameter value is $\sigma = 10\%$. 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. For VRPTW instances with 100 and 200 customers and one resource $\tau = 10$, $\bar{\tau} = 30$, for VRPTW instances with 100 and 200 customers and two resources $\tau = 5$, $\bar{\tau} = 10$, for VRPTW instances with 400 customers

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	R2.2.10	2549.4	2545.3	RC1.2.10	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
R1.2.10	3293.1	3277.3	RC1.2.6	3300.7	3289.0	RC2.2.10	1989.2	1962.0

Table 1: Target lower bounds for VRPTW instances

and one resource $\tau = 20$, $\bar{\tau} = 60$, for VRPTW instances with 400 customers and two resources $\tau = 10$, $\bar{\tau} = 20$, for HFVRP and DCVRP instances $\tau = 5$, $\bar{\tau} = 10$, for MDVRP and SDVRP instances $\tau = 10$, $\bar{\tau} = 20$.

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

6.3. Impact of bucket step size

In the first experiment, we measure the impact of different bucket step size on the solution time of the labeling algorithm used in the pricing step of the column-and-cut generation approach. This test was 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 1, we give the instance name, its optimum or best known solution value, and the target lower bound value we use. For instances not shown in this table, the target lower bound equals the optimum solution value.

The 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 $(u_0 - l_0)/\xi$. 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 $(u_0 - l_0)/\sqrt{\xi}$. All values \tilde{d}_r are rounded off to one decimal place. We test the following set Ξ of values ξ : $\Xi = \{1, 2, 5, 10, 20, 50, 100, 200, 500, 1000, 2000\}$.

Note that different parameterizations of the labeling algorithm are likely to change the set of routes returned by the pricing to the master, even if the value of the best route is the same. Thus, for each parameter value, the sequence of primal-dual master solutions in the column generation may follow a different path and converge to a distinct alternative optimal. In turn, this would change 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 tested value $\xi' \in \Xi$. Both copies are executed in each iteration of columns generation, and reduced cost fixing 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 tested value ξ' .

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 reduced cost fixing 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 $\varrho_{i,\xi'}^{tot}$ and $\varrho_{i,\xi'}^{sp}$ 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.

In Figure 5, we show the graphs for average and maximum values for both $\varrho_{i,\xi'}^{tot}$ and $\varrho_{i,\xi'}^{sp}$ among all instances $i \in I$, as well as graphs for these ratios for some representative individual instances. Note that the time limit was reached for $\xi' = 1, 2, 5, 10, 20, 50$ for 7, 5, 4, 3, 3, 2 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. Note that the case $\xi = 1$ corresponds to the standard label correcting algorithm which is often used in the literature.

We developed a scheme for dynamic adjustment of parameter ξ which outperforms this best static parameterisation $\xi = 200$. 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 the left part of Figure 6, we show the performance profile which compares the total time for three variants of the labeling algorithm: the best fixed static $\xi = 200$, the dynamic adjustment of ξ presented above, and the best instance-wise static ξ . Note that the latter variant is an ‘‘a posteriori’’ parameterisation which is not practical and shown here only for comparison purposes.

6.4. Impact of reduced cost fixing of bucket arcs

In this experiment we assess the impact of reduced cost fixing of bucket arcs in comparison with reduced cost fixing of original arcs (as in Irnich et al. (2010)). In the latter 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. Note that the procedure for fixing original arcs runs faster, as once

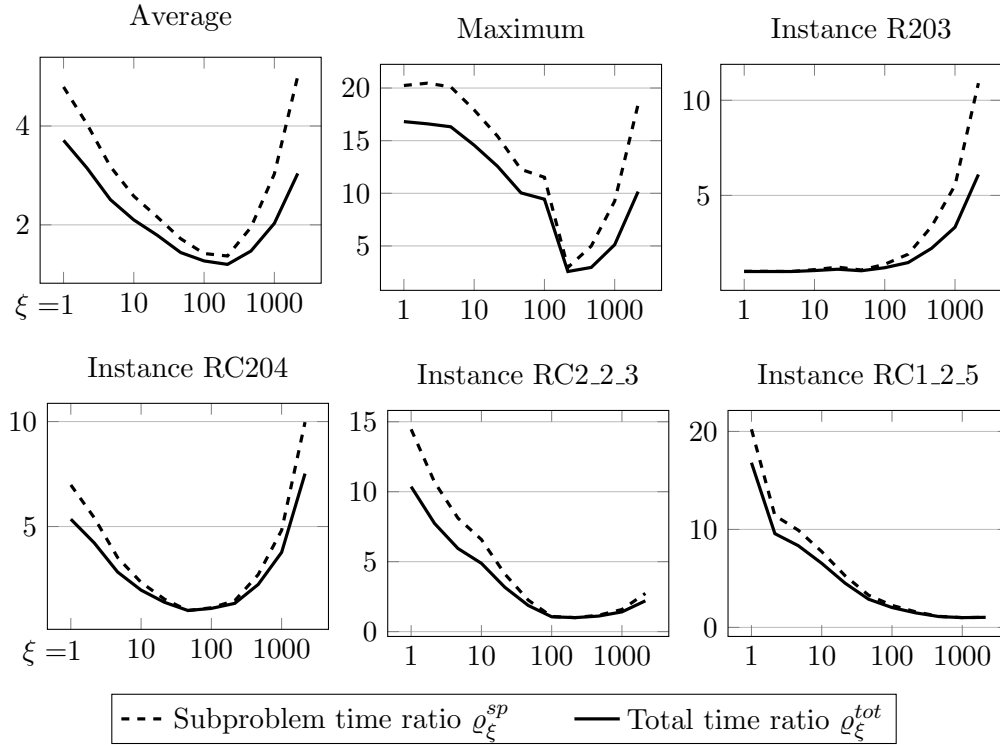


Figure 5: Impact of parameter ξ on the subproblem and total time

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 $a_{\gamma'} = a_{\gamma}$. We remark that the fixing procedure proposed in Pessoa et al. (2010), valid only for the single resource case with positive integer consumption, can be viewed as a fixing in a bucket graph with unitary step size ($\tilde{d} = 1$).

As in the previous experiment, to eliminate the impact of different sequences of primal-dual master solutions, we use two copies of the labeling algorithm, the first copy having always the same parameterization for a fixed instance. For both variants of the algorithm, we use dynamic adjustment of parameter ξ . In the right part of Figure 6 we show the performance profile which compares two variants of the algorithm: with reduced cost fixing of original arcs and reduced cost fixing of bucket arcs.

Although the average speed-up due to reduced cost fixing of bucket arcs is only 5%, for two instances it reaches the factor of two. These are instances RC2.2.8 and R2.2.4. In general, we noticed that one significantly benefits from reduced cost fixing of bucket arcs on hard instances with small primal-dual gap. Such instances have usually long routes.

6.5. Results for the branch-cut-and-price algorithm

In this section, we present the performance of the branch-cut-and-price 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 branch-cut-and-price algorithm for this problem

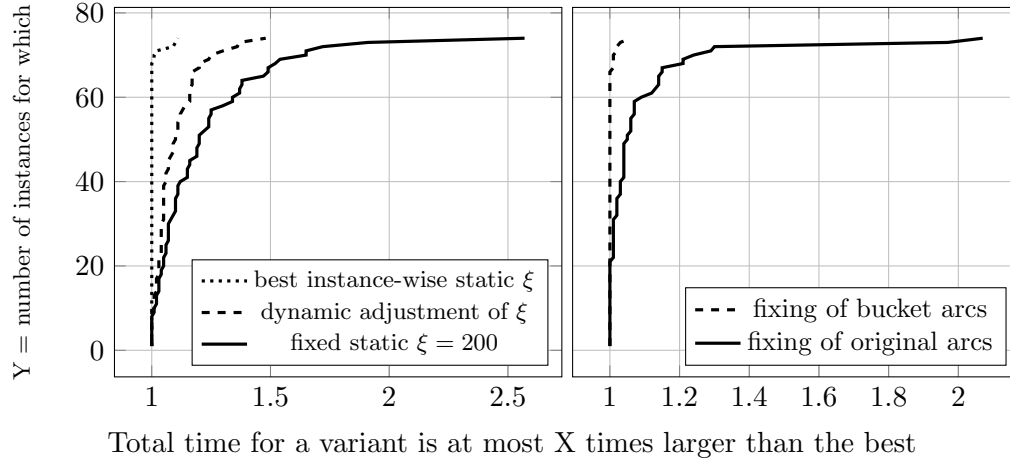


Figure 6: Performance profiles for variants with different parameterisation of ξ (left) and different reduced cost fixing (right)

by Pecin et al. (2017a). The difference between two algorithms mainly resides in different labeling routines included. Another difference is a better dual price smoothing stabilization with the automatically adjusted parameter (Pessoa et al., 2017) used by us. Also, we use only standard RCCs and limited-memory R1Cs with up to 5 rows, whereas algorithm in Pecin et al. (2017a) uses also elementary inequalities (a family of R1Cs) with 6 and 7 rows and clique cuts (after enumeration). Finally, different mechanisms to dynamically adjust ng -neighbourhoods are employed. Other components used in both algorithms are similar.

Note that the computer used in Pecin et al. (2017a) and our computer have practically the same speed, according to www.cpubenchmark.net/singleThread.html. In Table 2, we present the comparison of the two algorithms. Our algorithm outperforms the one by Pecin et al. for all classes of VRPTW instances, except class R1 for which the results are similar. Our algorithm proved optimality for 5 out of 9 open instances with 200 customers for the first time. When compared on the set of instances solved by the both algorithms, our approach is 3 times faster in terms of the average time, and 4 times faster in terms of the geometric mean time. Speed-up factor on individual instances reaches 47.5 as demonstrated in the detailed results presented in the Appendix. Interestingly, the largest improve factors are reached on instances for which the reduced cost fixing of bucket arcs had the most impact.

We have also tested our algorithm on VRPTW instances with 400 customers. Note that in the literature only one of these instances has been solved to optimality, by Kallehauge et al. (2006). We managed to solve to optimality 24 instances out of 60. There are three main shortcomings in our algorithm (in decreasing order of the number of impacted instances) that prevented it from solving 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 it is too large (ii) R1Cs with up to 5 rows can be added almost until convergence, but they still

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 2: Comparison of our branch-cut-and-price algorithm with the one in Pecin et al. (2017a) on the VRPTW instances

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:

- 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.

Note 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 Appendix, the solutions times of Contardo and Martinelli (2014) are divided by 1.6. In Table 3, 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 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 4 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

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 3: Comparison of our branch-cut-and-price algorithm with the one in Contardo and Martinelli (2014) on the distance-constrained MDVRP instances

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 4: Results for instances never attacked by exact methods before

6 out of 7 classic distance-constrained CVRP instances by Christofides et al. (1979). All best known solutions 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 7 out of 10 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 et al. (2011) who called these instances “nightmare”. Particularities of these instances are large capacities of vehicles, very heterogeneous customer demands, and many – up to 8 – 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 Appendix.

7. Conclusions

This article proposed a new bucket-graph based variant of the labeling algorithm for the RCSP. It can handle up to two resources and also the particularities of the RCSPs that arise as pricing subproblems 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 likely to perform better than existing variants when

the consumptions are given either by real numbers or by large integer numbers. Besides, a new procedure for removing bucket graph arcs by reduced costs 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. The new BCP outperformed another very recent BCP algorithm for the classical VRPTW. It also obtained very good results on instances from other important VRP variants, including classical DCVRP instances proposed almost 40 years ago, but only now solved to optimality.

Acknowledgments

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.

References

- 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., July 2017. A branch-and-price algorithm for the minimum latency problem. Tech. Rep. L-2017-4, Cadernos do LOGIS-UFF, Niterói, Brazil.
- Christofides, N., Mingozzi, A., Toth, P., 1979. *Combinatorial Optimization*. Wiley, Chichester, Ch. The vehicle routing problem, pp. 315–338.
- 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, J.-F., Gendreau, M., Laporte, G., 1997. A tabu search heuristic for periodic and multi-depot vehicle routing problems. *Networks* 30 (2), 105–119.
- Cordeau, J.-F., 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, J.-F., 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, J. C. D., 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, J. W., Linfati, R., Toth, P., Baldoquin, M. G., 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. *Column generation*, 33–65.
- 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.
- Kallehauge, B., Larsen, J., Madsen, O. B., 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, A. L., 2015. An exact algorithm for the elementary shortest path problem with resource constraints. *Transportation Science* 50 (1), 348–357.
- Lozano, L., Medaglia, A. L., 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. In: *International Conference on Integer Programming and Combinatorial Optimization*. Springer, pp. 393–403.
- Pecin, D., Pessoa, A., Poggi, M., Uchoa, E., Mar 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.
URL <http://www.sciencedirect.com/science/article/pii/S0167637717301049>
- Penna, P. H. V., Subramanian, A., Ochi, L. S., Vidal, T., Prins, C., 2017. A hybrid heuristic for a broad class of vehicle routing problems with heterogeneous fleet. *Annals of Operations Research* (Forthcoming).
- Pessoa, A., Sadykov, R., Uchoa, E., Vanderbeck, F., 2017. Automation and combination of linear-programming based stabilization techniques in column generation. *INFORMS Journal on Computing* Forthcoming.
- Pessoa, A., Uchoa, E., de Aragão, M. P., 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.
- Poggi de Aragão, M., Uchoa, E., 2003. Integer program reformulation for robust branch-and-cut-and-price. In: Wolsey, L. (Ed.), *Annals of Mathematical Programming in Rio*. Búzios, Brazil, pp. 56–61.
- Pugliese, L. D. P., 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, M. M., 1987. Algorithms for the vehicle routing and scheduling problems with time window constraints. *Operations Research* 35 (2), 254–265.
- Tilk, C., Rothenbächer, A.-K., 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, T. G., 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, T. G., 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.

Appendix A. Detailed results

In the Appendix, we present the detailed results for our branch-cut-and-price 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, 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 improve factor (ratio of the running times of the literature algorithm and ours). In Tables A.5–A.11, “Time PCDU” refers to the running time of the algorithm by Pecin et al. (2017a). In Tables A.18–A.19, “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	Improve
	<i>UB</i>		bound	bound					

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 A.5: Detailed results for harderst 100-customers Solomon VRPTW instances

Instance	Initial <i>UB</i>	Robust bound	Root			Nodes number	Total time	Final <i>UB</i>	Time PCDU	Improve factor
			bound	time	#R1C					
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 A.6: Detailed results for 200-customer Gehring and Homberger VRPTW instances of class C1

Instance	Initial <i>UB</i>	Robust bound	Root			Nodes number	Total time	Final <i>UB</i>	Time PCDU	Improve factor
			bound	time	#R1C					
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 A.7: Detailed results for 200-customer Gehring and Homberger VRPTW instances of class C2

Instance	Initial <i>UB</i>	Robust bound	Root		Nodes number	Total time	Final <i>UB</i>	Time PCDU	Improve factor	
			bound	time						#R1C
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 A.8: Detailed results for 200-customer Gehring and Homberger VRPTW instances of class R1

Instance	Initial <i>UB</i>	Robust bound	Root		Nodes number	Total time	Final <i>UB</i>	Time PCDU	Improve factor	
			bound	time						#R1C
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 A.9: Detailed results for 200-customer Gehring and Homberger VRPTW instances of class R2

Instance	Initial <i>UB</i>	Robust bound	Root		Nodes number	Total time	Final <i>UB</i>	Time PCDU	Improve factor	
			bound	time						#R1C
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 A.10: Detailed results for 200-customer Gehring and Homberger VRPTW instances of class RC1

Instance	Initial <i>UB</i>	Robust bound	Root		Nodes number	Total time	Final <i>UB</i>	Time PCDU	Improve factor	
			bound	time #R1C						
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 A.11: Detailed results for 200-customer Gehring and Homberger VRPTW instances of class RC2

Instance	Initial <i>UB</i>	Robust bound	Root		Nodes number	Total time	Final <i>UB</i>	
			bound	time #R1C				
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	7083.0*
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 et al. (2006)

Table A.12: Detailed results for 400-customer Gehring and Homberger VRPTW instances of class C1

Instance	Initial <i>UB</i>	Robust bound	Root			Nodes number	Total time	Final <i>UB</i>
			bound	time	#R1C			
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 A.13: Detailed results for 400-customer Gehring and Homberger VRPTW instances of class C2

Instance	Initial <i>UB</i>	Robust bound	Root			Nodes number	Total time	Final <i>UB</i>
			bound	time	#R1C			
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 A.14: Detailed results for 400-customer Gehring and Homberger VRPTW instances of class R1

Instance	Initial <i>UB</i>	Robust bound	Root			Nodes number	Total time	Final <i>UB</i>
			bound	time	#R1C			
R2.4.1	7522.1	7485.3	7520.7	42m42s	96	1	42m42s	<u>7520.7*</u>
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	<u>6067.8*</u>
R2_4.10	5665.1	5561.6	5619.5	4h09m	758	21	>60h	5665.1

Table A.15: Detailed results for 400-customer Gehring and Homberger VRPTW instances of class R2

Instance	Initial <i>UB</i>	Robust bound	Root		Nodes number	Total time	Final <i>UB</i>	
			bound	time #RIC				
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 A.16: Detailed results for 400-customer Gehring and Homberger VRPTW instances of class RC1

Instance	Initial <i>UB</i>	Robust bound	Root		Nodes number	Total time	Final <i>UB</i>	
			bound	time #RIC				
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 A.17: Detailed results for 400-customer Gehring and Homberger VRPTW instances of class RC2

Inst.	<i>n</i>	<i>m</i>	Initial <i>UB</i>	Robust bound	Root		Nod. num.	Total time	Final <i>UB</i>	Time CM	Impr. factor	
					bound	time #RIC						
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 A.18: Detailed results for Cordeau et al. MDVRP instances of class “p”

Inst.	<i>n</i>	<i>m</i>	Initial <i>UB</i>	Robust bound	Root		Nod. num.	Total time	Final <i>UB</i>	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	<u>2867.26*</u>	–	–

Table A.19: Detailed results for Cordeau et al. MDVRP instances of class “pr”

Instance	<i>n</i>	<i>m</i>	Initial <i>UB</i>	Robust bound	Root		Nodes number	Total time	Final <i>UB</i>	
					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	<u>2574.56*</u>
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	<u>4375.67</u>
pr06	288	4	4422.20	4326.08	4329.25	21m45s	21	353	>60h	<u>4421.95</u>
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	<u>2969.92*</u>
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 A.20: 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 A.21: Detailed results for Christofides et al. DCVRP instances

Instance	n	m	Initial UB	Robust bound	Root			Nodes number	Total time	Final UB
					bound	time	#R1C			
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 A.22: Detailed results for Duhamel et al. HFVRP instances with up to 100 customers

Instance	n	m	Initial UB	Robust bound	Root			Nodes number	Total time	Final UB
					bound	time	#R1C			
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	<u>21257.38*</u>
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	<u>6278.62*</u>
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	<u>10869.04*</u>
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	<u>11682.98*</u>
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	<u>10001.80*</u>
DLP_68	125	4	8971	8799.96	8852.89	5m39s	192	89	3h00m	<u>8889.03*</u>
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	<u>6406.16*</u>
DLP_88	127	5	12389	12328.90	12385.74	29s	25	1	29s	<u>12385.74*</u>
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	<u>11056.13*</u>
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	<u>5739.02*</u>
DLP_60	137	4	17037	16768.33	16934.97	5m40s	285	37	44m38s	<u>17012.42*</u>
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	<u>5525.90</u>
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	<u>8763.90*</u>
DLP_79	147	4	7260	7132.58	7165.36	8m22s	187	21	1h13m	<u>7257.97*</u>
DLP_66	150	4	12784	12484.59	12587.01	8m31s	104	999	>60h	<u>12776.24</u>

Table A.23: Detailed results for Duhamel et al. HFVRP instances with 101–150 customers

Instance	n	m	Initial UB	Robust		Root		Nodes number	Total time	Final UB
				bound	bound	time	#RIC			
DLP_69	152	4	9148	9003.97	9086.90	7m52s	335	147	3h36m	<u>9127.16*</u>
DLP_76	152	8	11995	11900.23	11933.72	5m28s	212	1195	>60h	<u>11994.22</u>
DLP_56	153	4	31031	30571.34	30764.30	8m26s	250	1591	58h50m	<u>30905.95*</u>
DLP_86	153	5	9028	8935.26	8957.90	8m23s	274	249	8h45m	<u>9020.63*</u>
DLP_37	161	5	6851	6751.15	6787.92	7m57s	354	2017	>60h	<u>6838.72</u>
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	<u>9132.03*</u>
DLP_09	167	5	7604	7543.04	7563.84	12m52s	274	135	2h16m	<u>7599.72*</u>
DLP_35	168	6	9556	9492.98	9521.16	2m55s	114	3	3m17s	<u>9522.45*</u>
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	<u>10772.81*</u>
DLP_63	174	5	19952	19670.12	19799.92	8m54s	108	97	5h24m	<u>19890.65*</u>
DLP_14	176	4	5645	5511.85	5511.85	4m37s	0	1023	>60h	<u>5639.98</u>
DLP_42	178	7	10818	10615.08	10615.08	7m00s	0	889	>60h	<u>10805.94</u>
DLP_02	181	4	11676	11603.82	11632.01	4m27s	104	25	9m34s	<u>11649.81*</u>
DLP_04	183	4	10749	10582.45	10673.39	9m41s	277	985	13h53m	<u>10714.84*</u>
DLP_95	183	2	6176	6100.98	6119.06	6m53s	290	1709	>60h	<u>6170.20</u>
DLP_71	186	3	9835	9711.77	9771.86	10m30s	292	87	4h08m	<u>9798.06*</u>
DLP_72	186	4	5904	5780.39	5831.66	9m35s	322	365	31h31m	<u>5870.43*</u>
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	<u>9333.39</u>
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	<u>6348.36</u>

Table A.24: Detailed results for Duhamel et al. HFVRP instances with 151–200 customers

Instance	n	m	Initial UB	Robust		Root		Nodes number	Total time	Final UB
				bound	bound	time	#R1C			
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	<u>22987.90</u>
DLP_22	239	2	13069	12923.02	12956.02	13m12s	357	1193	>60h	<u>13050.00</u>
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	<u>9649.05</u>

Table A.25: Detailed results for Duhamel et al. HFVRP instances with more than 200 customers