EURO J Transp Logist (2019) 8:147–168
DOI 10.1007/s13676-017-0112-9

RESEARCH PAPER

Selective pricing in branch-price-and-cut algorithms for vehicle routing

Guy Desaulniers[1] · Diego Pecin[1,3] · Claudio Contardo[2]

Received: 12 November 2016 / Accepted: 6 September 2017 / Published online: 25 September 2017

Abstract Branch-price-and-cut is a leading methodology for solving various vehicle routing problems (VRPs). For many VRPs, the pricing subproblem of a branch-price-and-cut algorithm is highly time consuming, and to alleviate this difficulty, a relaxed pricing subproblem is used. In this paper, we introduce a new paradigm, called selective pricing, that can be applied in this context to reduce the time required for solving hard-to-solve VRPs by branch-price-and-cut. This paradigm requires the development of a labeling algorithm specific to the pricing subproblem. To illustrate selective pricing, we apply it to a branch-price-and-cut algorithm for the VRP with time windows, where the relaxed pricing subproblem is a shortest *ng*-path problem with resource constraints. We develop a labeling algorithm for this subproblem and show through computational experiments that it can yield significant time reductions (up to 32%) to reach a good lower bound on certain very-hard-to-solve VRPTW instances with 200 customers. We also introduce a new labeling heuristic which also leads to computational time reductions.

✉ Claudio Contardo
claudio.contardo@gerad.ca

Guy Desaulniers
guy.desaulniers@gerad.ca

Diego Pecin
diego.pecin@isye.gatech.edu

1 GERAD and Department of Mathematics and Industrial Engineering Polytechnique Montréal, Montréal, Canada

2 GERAD, CIRRELT and Department of Management and Technology, ESG UQÀM, Montréal, Canada

3 Present Address: H. Milton Stewart School of Industrial and Systems Engineering, Georgia Institute of Technology, Atlanta, GA, USA

Keywords Column generation · Vehicle routing problem · Branch-and-price · Shortest path problem under resource constraints · Selective pricing

1 Introduction

Vehicle routing problems (VRPs, see Toth and Vigo 2014) consist of determining least-cost vehicle routes that visit a set of customers to service them (typically, to deliver or pickup merchandise). Widely studied for more than 50 years, these problems remain challenging, especially with new variants that incorporate complex features from real-world applications such as split deliveries (Irnich et al. 2014), pickups and deliveries (Battarra et al. 2014; Doerner and Salazar-González 2014), and stochastic demands (Gendreau et al. 2014), to name just a few.

A generic integer programming model for the VRPs is as follows:

$$\min \sum_{r \in R} c_r x_r \tag{1}$$

$$\text{s.t.} \sum_{r \in R} a_{hr} x_r = b_h, \forall h \in H \tag{2}$$

$$x_r \in \{0, 1\}, \forall r \in R, \tag{3}$$

where R denotes the set of feasible vehicle routes, c_r is the cost of route $r \in R$, H is the set of constraints, a_{hr} is the contribution of route $r \in R$ to constraint $h \in H$, and b_h is the right-hand side of this constraint. Each binary variable x_r indicates whether or not route $r \in R$ is selected in the solution. The objective function (1) aims at minimizing the total routing costs. Constraint set (2) may include various types of constraints, depending on the problem variant considered. For example, when each customer must be visited exactly once, they include set partitioning constraints (for such a constraint h, $b_h = 1$ and $a_{hr} \in \{0, 1\}$ for all $r \in R$) to force one route to visit each customer. This set of constraints can also include inequalities, for instance, to limit the number of routes selected to the number of available vehicles. Finally, let us mention that additional variables not directly associated with routes (e.g., a binary variable indicating whether or not a customer is serviced by an external provider) may also be part of this model.

Over the years, various exact algorithms have been developed to solve VRPs. Currently, branch-price-and-cut algorithms are the most successful ones for a wide variety of VRPs. A branch-price-and-cut algorithm (see Barnhart et al. 1998; Lübbecke and Desrosiers 2005) is a branch-and-bound algorithm in which the linear relaxation of a mixed-integer linear problem at each node of the search tree is solved using column generation and tightened with the addition of cuts. In this context, such a linear relaxation is called a master problem (MP). Column generation is an iterative algorithm that solves at each iteration a restricted master problem (RMP) and a pricing subproblem (PS). For models (1)–(3), this RMP corresponds to the MP (augmented by the cuts and branching decisions applicable at the corresponding node) restricted to a small subset of the routes in R, and this PS is often an elementary shortest path problem with resource constraints (ESPPRC) (see Irnich and Desaulniers 2005). Indeed, elementarity requirements forbidding multiple visits to the same customer along a route is often enforced in the PS (see Feillet et al. 2004). At a given iteration, the RMP is solved to yield a primal and a dual solution. Then, the PS is solved to find negative reduced cost columns (variables) with respect to this dual solution. If such columns are found, they are added to the RMP before starting a new iteration. Otherwise, the algorithm stops and the cost of the current RMP solution provides a lower bound at the current node of the search tree.

For many VRPs, the PS turns out to be a problem with high complexity. For instance, the ESPPRC arising for the vehicle routing problem with time windows (VRPTW) is known to be strongly *NP*-hard (Dror 1994). To alleviate the difficulty of solving a time-consuming PS at each iteration, a relaxation of it is often solved instead, yielding the generation of infeasible routes, for instance, routes visiting the same customer more than once. These routes can be part of linear relaxation solutions, reducing the quality of the computed lower bounds, but they are discarded from the solution afterwards, through cutting and branching.

Several PS relaxations have been proposed for various VRPs. The most notorious ones are relaxations of the ESPPRC, which can be applied to several VRPs. To replace a special case of the ESPPRC considering a single resource, namely, time, Desrosiers et al. (1984) introduced the shortest path problem with time windows that omit the elementarity requirements. This relaxation was generalized by Desrochers et al. (1992) who developed a branch-and-price algorithm for the VRPTW. The resulting PS was called the shortest path problem with resource constraints (SPPRC). To strengthen this relaxation, these authors also considered the SPPRC with 2-cycle elimination that forbids cycles of the form $i - j - i$ (where i and j represent customer nodes). Later, Irnich and Villeneuve (2006) extended this idea to eliminate all cycles of length k or less, resulting in the SPPRC with k-cycle elimination. More recently, Baldacci et al. (2011) introduced the *ng*-route relaxation that rapidly became the state-of-the-art when routes must be elementary like in the ESPPRC. An *ng*-route is a route that may contain certain cycles if they meet certain conditions. More precisely, let NG_i be a neighborhood of customer i that contains i and a subset of other customers (typically, the closest ones or those identified as having high chances of being part of a cycle starting and ending at customer i). An *ng*-route can visit a customer i twice if at least one customer j, such that $i \notin NG_j$ is visited between the two visits to i. Using this route relaxation, the PS becomes a shortest *ng*-path problem with resource constraints (*ng*-SPPRC) which can be solved by a labeling algorithm. If the size of the neighborhoods is small, the labeling algorithm is fast, but the lower bounds may be weak. On the other hand, larger neighborhoods yield better lower bounds, but the PS is harder to solve.

In this paper, we present a new paradigm, called *selective pricing*, that can be applied to branch-price-and-cut algorithms for VRPs when a relaxed PS is used to generate columns. Its application to a specific VRP requires the development of a new algorithm to solve the PS. We illustrate its usefulness by concentrating on the *ng*-route relaxation. However, we believe that our main contribution is not the development of a new labeling algorithm for the *ng*-SPPRC but rather the introduction of this selective-pricing paradigm that opens up a new way of designing algorithms for solving relaxed PSs. Note also that this paradigm may be applicable to other problem types that are solvable by branch-price-and-cut.

This paper is structured as follows. Section 2 presents the selective-pricing paradigm. Its application when the *ng*-SPPRC is used as a relaxed PS is described in Sect. 3. Computational results on the VRPTW are reported in Sect. 4. Conclusions are drawn in Sect. 5.

2 Selective pricing

In this section and the next one, we consider an arbitrary MP that can arise at any node of the search tree and is used to compute a lower bound at this node. Therefore, this MP might not be the linear relaxation of (1)–(3) as branching decisions and cutting planes might have led to a different linear relaxation. Nevertheless, we assume without loss of generality that the linear relaxation of (1)–(3) corresponds to a generic formulation of any MP.

When using a PS relaxation, infeasible routes may be generated. Various infeasibility types might be considered. For instance, for the ESPPRC, one can relax path-structural constraints such as the elementarity requirements and allow routes that visit a same customer more than once. Alternatively, some resource constraints can be relaxed, and for example, routes exceeding vehicle capacity can be accepted. Consequently, the set of routes R considered in the MP is enlarged to a set $\hat{R}$. In the following, the routes in R are called the *feasible routes*, while those in $\hat{R}\backslash R$ are termed the *relaxed routes*. Traditionally, the role of the PS is to find negative reduced cost columns in $\hat{R}$, if at least one exists, even if such routes are all relaxed ones. The PS is then formulated as the problem of finding a route in $\hat{R}$ that has the least reduced cost. More precisely, let π_h, $h \in H$, be the dual variables associated with constraints (2) and let $\bar{c}_r = c_r - \sum_{h\in H} a_{hr}\pi_h$ be the reduced cost of a route $r \in \hat{R}$. The traditional PS is defined as

$$z^{PS}(\hat{R}) \quad = \quad \min_{r\in\hat{R}} \quad \bar{c}_r. \tag{4}$$

To guarantee convergence of the column generation algorithm, it is not necessary to always find a least reduced cost route as long as negative reduced cost routes are identified. Consequently, the PS can rather be stated as

$$\text{If } z^{PS}(\hat{R}) < 0, \text{ find at least one route } r \in \hat{R} \text{ with } \bar{c}_r < 0. \tag{5}$$

The labeling algorithm used for solving the PS is designed to do so. The column generation algorithm stops when $z^{PS}(\hat{R}) \geq 0$, i.e., when there are no more routes in $\hat{R}$ with a negative reduced cost. In this case, the optimal value z^{RMP} of the current RMP provides a lower bound at the current node of the branch-and-bound search tree. This lower bound is denoted $\underline{z}^{MP(\hat{R})}$ to highlight that it is equal to the optimal value of the MP when it considers the set of routes $\hat{R}$.

Observe, however, that this stopping condition is not necessary to ensure that z^{RMP} is a valid lower bound. Indeed, one can replace $\hat{R}$ by R to obtain the sufficient condition $z^{PS}(R) \geq 0$ which does not imply $\bar{c}_r \geq 0$ for all $r \in \hat{R} \backslash R$, i.e., there might exist a relaxed route $r \in \hat{R} \backslash R$ with $\bar{c}_r < 0$. The sufficiency of this condition is proven in the following proposition.

Proposition 1 *The optimal value z^{RMP} of the RMP at a given column generation iteration is a valid lower bound at the current node of the search tree if $z^{PS}(R) \geq 0$ or, equivalently, if $\bar{c}_r \geq 0$, $\forall r \in R$.*

Proof Let $R' \subseteq \hat{R} \backslash R$ be the subset of relaxed routes that have been generated and that are taken into account in the current RMP. Because $R' \cup R \supseteq R$, considering only the set of routes $R' \cup R$ in the MP (instead of $\hat{R}$) yields a relaxation of the original models (1)–(3), and thus, an optimal solution to this MP provides a valid lower bound. The optimal solution of the current RMP, extended by setting to zero all feasible route variables that are not yet generated, forms such an optimal solution of cost z^{RMP}. Indeed, for this solution, $\bar{c}_r \geq 0, \forall r \in R$, because $z^{PS}(R) \geq 0$ by assumption and $\bar{c}_r \geq 0$, $\forall r \in R'$, because all variables x_r, $r \in R'$, are in the current RMP. □

In the following, we denote by $\underline{z}^{MP(R' \cup R)} = z^{RMP}$ this lower bound, where R' is defined as in the above proof. This notation highlights again that the lower bound is equal to the optimal value of the MP involving only the set of routes $R' \cup R$.

Following Proposition 1, we can re-state the PS as follows:

$$\text{If } z^{PS}(R) < 0, \text{ find at least one route } r \in \hat{R} \text{ with } \bar{c}_r < 0. \tag{6}$$

The difference between the traditional PS definition (5) and this new PS definition (6) is subtle ($\hat{R}$ is replaced by R in the condition), but may allow to define pricing algorithms that are more efficient than the existing ones. In fact, these algorithms may discard relaxed routes in $\hat{R} \backslash R$ even if it cannot be proved that they do not lead to a route of minimum reduced cost. To ensure the exactness of the column generation algorithm, these pricing algorithms must, however, identify a negative reduced cost route in $\hat{R}$ (i.e., feasible or relaxed) if there exists at least one feasible route in R that has a negative reduced cost. For these reasons, we say that the pricing process is *selective* or that *selective pricing* is applied.

Let us make some remarks about the usage of selective pricing in a branch-price-and-cut algorithm.

1. At a node of the search tree, the MP might not be solved to proven optimality as the reduced cost of some relaxed route variables may be negative even if $z^{\mathrm{PS}}(R) \geq 0$.
2. At a node of the search tree, selective pricing can yield a better lower bound than the traditional pricing. Indeed, if R' corresponds to the set of relaxed routes in the last RMP solved in this node, then $\underline{z}^{\mathrm{MP(R' \cup R)}} \geq \underline{z}^{MP(\hat{R})}$ because $R' \cup R \subseteq \hat{R}$.
3. The lower bound $\underline{z}^{MP(R' \cup R)}$ computed at a node depends on R'. Therefore, different parameter configurations for the same algorithm might yield different lower bounds at a same node.
4. In a search tree, the lower bound achieved at a node may be less than the lower bound achieved at its parent node because relaxed routes that were not

generated in the parent node may be generated in the child node. If this occurs, one can always assign the parent node’s lower bound to the child node.

To the best of our knowledge, only Cherkesly et al. (2015) have used a special case of selective pricing in the vehicle routing literature. They introduced different branch-price-and-cut algorithms for the pickup and delivery problem with time windows and last-in-first-out loading constraints. In the algorithms based on an *ng*-path relaxation, the labeling algorithm used to solve the PS may discard routes with cycles without proving that they are dominated by other routes. Cherkesly et al. (2015) only gave a very short justification for allowing this. In this section, we have presented selective pricing in a general context and fully justified it.

3 Selective pricing with the *ng*-SPPRC

To illustrate selective pricing, we develop a new labeling algorithm for solving[1] the *ng*-SPPRC. We present this algorithm in the context of the VRPTW which is used as an illustrative example for the rest of this paper. The proposed ideas are, however, applicable to other VRPs.

This section is divided into three subsections. First, we define the VRPTW and the resulting *ng*-SPPRC PS. Second, we introduce the new labeling algorithm. Finally, we discuss a heuristic version of it that is less time consuming and can be used to try to generate negative reduced cost columns rapidly.

3.1 The vehicle routing problem with time windows

The VRPTW (see Desaulniers et al. 2014) involves a set of customers N and a sufficiently large set of vehicles. With each customer, $i \in N$ are associated a demand d_i, a service time s_i, and a time window $[e_i, \ell_i]$ within which the service must start. The vehicles are all identical with a capacity Q and their routes must start and end at a common depot. For each pair of locations i and j, the travel time t_{ij} from i to j and the travel cost c_{ij} are assumed to be known. The VRPTW consists of finding feasible routes, such that each customer is visited once and the total routing costs are minimized. A route is said to be feasible if the sum of the demands of the visited customers does not exceed Q and if their time windows are respected. The cost of a route is computed as the sum of the costs of the arcs forming it.

For the VRPTW, models (1)–(3) becomes

$$\min \sum_{r \in R} c_r x_r \tag{7}$$

$$\text{s.t.} \sum_{r \in R} a_{ri} x_r = 1, \forall i \in N \tag{8}$$

[1] We continue to use the expression “solving the PS” even if we define the PS according to (6).

$$x_r \in \{0,1\}, \forall r \in R, \tag{9}$$

where a_{ri} is an integer constant equal to the number of times that route $r \in R$ visits customer $i \in N$ ($a_{ri} \in \{0,1\}$ if r is elementary). Objective function (7) minimizes the total routing costs, whereas constraints (8) ensure that each customer is visited by a single route. The dual variables associated with constraints (8) are denoted π_i, $i \in N$, and the reduced cost of route variable x_r is given by $\bar{c}_r = c_r - \sum_{i \in N} a_{ri}\pi_i$. In a classical branch-price-and-cut algorithm for the VRPTW, the PS is an ESPPRC, which is often replaced by an *ng*-SPPRC PS in the most recent works.

The *ng*-SPPRC PS can be defined on a directed graph $G = (V,A)$ with node set $V = N \cup \{o, \bar{o}\}$ and arc set A. Nodes o and $\bar{o}$ represent the depot at the beginning and the end of a route, respectively. With these nodes, we also associate time windows $[e_o, \ell_o] = [0,0]$ and $[e_{\bar{o}}, \ell_{\bar{o}}] = [0, \mathcal{T}]$, where $\mathcal{T}$ is the horizon length, and define without loss of generality $s_o = d_o = d_{\bar{o}} = \pi_o = 0$. The arc set is given by

$$A = \{(i,j) \in V \times V \mid i \neq \bar{o}, j \neq o, e_i + s_i + t_{ij} \leq \ell_j, d_i + d_j \leq Q\}.$$

For each arc $(i,j) \in A$, we define its reduced cost as $\bar{c}_{ij} = c_{ij} - \pi_i$. Finally, let $NG_i \subseteq N$ be the neighborhood of node $i \in N$, i.e., a subset of customers including i, as described in Sect. 1.

Let $p = (v_0, v_1, \ldots, v_k)$, $v_j \in V$ for all $j = 0, 1, \ldots, k$, be a path in G. It corresponds to a route in $\hat{R}$, hereafter an *ng*-route, if it

- Starts and ends at the depot: $v_0 = o$ and $v_k = \bar{o}$;
- Satisfies vehicle capacity: $\sum_{j=1}^{k-1} d_{v_j} \leq Q$;
- Satisfies time windows: for every node v_j, $j = 0, 1, \ldots, , k$, the earliest start of service time $T_{v_j} \in [e_{v_j}, \ell_{v_j}]$, where $T_{v_0} = e_{v_0} = 0$ and $T_{v_{j+1}} = \max\{e_{v_{j+1}}, T_{v_j} + s_{v_j} + t_{v_j, v_{j+1}}\}$ for all $j = 1, \ldots, k$;
- Does not contain cycles that violate the *ng*-route requirements: for every pair of integers j_1 and j_2 , such that $0 < j_1 < j_2 < k$ and $v_{j_1} = v_{j_2}$, there exists another integer j_3 , such that $j_1 < j_3 < j_2$ and $v_{j_1} \notin NG_{v_{j_3}}$.

This path is elementary (i.e., it belongs to R) if $v_{j_1} \neq v_{j_2}$ for all pairs of integers j_1 and j_2 , such that $0 < j_1 < j_2 < k$. The reduced cost of p is given by $\bar{c}_p = \sum_{j=0}^{k-1} \bar{c}_{v_j, v_{j+1}}$.

In the following, *ng*-routes (or *ng*-paths) that are elementary are said to be *feasible paths*, while the others are called *relaxed paths*. We use *ng*-path when it can be either feasible or relaxed.

According to definition (6), the selective *ng*-SPPRC PS consists of finding a negative reduced cost *ng*-path in G if there exists at least one negative reduced cost feasible path in G, or proving that no such feasible path exists.

3.2 Labeling algorithm

We start by describing a (mono-directional) labeling algorithm for solving the *ng*-SPPRC without selective pricing. A labeling algorithm starts from an initial label at node o of G and extends labels forwardly in G using resource extension functions.

Each label L is associated with a node $n(L)$ and represents an ng-path $p(L)$ from o to $n(L)$. To avoid enumerating all ng-paths, a dominance rule is applied to eliminate labels that cannot yield an optimal ng-path.

For the ng-SPPRC arising from the VRPTW, a label L contains the following components: $n(L)$, its resident node; $Z(L)$, the reduced cost of $p(L)$; $D(L)$, the load accumulated along $p(L)$; $T(L)$, the earliest service start time at $n(L)$ if reached using $p(L)$; and $M(L) \subseteq NG_{n(L)}$, the subset of nodes in $NG_{n(L)}$ to which label L cannot be extended, because it would be infeasible with regard to vehicle capacity, time windows or the ng-route constraints. This label writes as $L = (n(L), Z(L), D(L), T(L), M(L))$. The initial label at node o is given by $L_o = (o, 0, 0, 0, \emptyset)$. A label L with $n(L) = i$ can be extended along an arc $(i,j) \in A$ only if $j \notin M(L)$. When performing this extension, the following resource extension functions are used to create a new label L':

$$n(L') = j \tag{10}$$

$$Z(L') = Z(L) + \bar{c}_{ij} \tag{11}$$

$$D(L') = D(L) + d_j \tag{12}$$

$$T(L') = \max\{e_j, T(L) + s_i + t_{ij}\} \tag{13}$$

$$M(L') = \left(\{j\} \cup M(L) \cup U(n(L'), D(L'), T(L'))\right) \cap NG_j, \tag{14}$$

where $U(n, D, T) = \{v \in N \cup \{\bar{o}\} > | > D + d_v > Q \mathrm{or} \mathrm{T} + \mathrm{s_n} + \mathrm{t_{n}}_v > \ell_v\}$ is the set of unreachable nodes from a label at node n associated with a load of D and an earliest service start time of T. In the above definition of $U(n, D, T)$, we assume that the service and travel time consumptions along the arcs satisfy the triangle inequality. If this is not the case, $s_n + t_{nv}$ must be replaced by the shortest path duration between n and v. Finally, a label L_1 (or ng-path $p(L_1)$) is said to dominate a label L_2 (or ng-path $p(L_2)$) if

$$n(L_1) = n(L_2) \tag{15}$$

$$Z(L_1) \leq Z(L_2) \tag{16}$$

$$D(L_1) \leq D(L_2) \tag{17}$$

$$T(L_1) \leq T(L_2) \tag{18}$$

$$M(L_1) \subseteq M(L_2). \tag{19}$$

All dominated labels are discarded except when two or more labels dominate each other (i.e., all the previous relations hold at equality for each pair of labels). In this case, one of these labels is kept.

Note that (15)–(19) are sufficient conditions which guarantee that any feasible extension χ of $p(L_2)$ is also feasible for $p(L_1)$ and that the reduced cost of $p(L_1) \oplus \chi$

is less than or equal to that of $p(L_2) \oplus \chi$, where the symbol $\oplus$ denotes the concatenation of the two paths it links.

In the rest of this section, we propose to modify the above algorithm to perform selective pricing. In this new algorithm, partial relaxed paths can be discarded even if it is not proven that they cannot yield a shortest *ng*-path between o and $\bar{o}$. The description of this algorithm requires the following definition.

Definition 2 Let $p = (o, v_1, \ldots, v_k)$ be an *ng*-path. The *dominated path set* (DPS) of p contains the following *ng*-paths:

1. The *ng*-path p;
2. All *ng*-paths $(o, w_1, \ldots, w_m) = (o, u_1, \ldots, u_{j_1} = v_{j_2}) \oplus (v_{j_2}, \ldots, v_k)$ (for some $j_2 \in \{1, \ldots, k\}$) obtained by concatenating an *ng*-path $(o, u_1, \ldots, u_{j_1})$ identified by the algorithm as dominated by the prefix[2] $(o, v_1, \ldots, v_{j_2})$ of p and the suffix $(v_{j_2}, \ldots, v_k)$ of p; and
3. Recursively, all *ng*-paths built by concatenating an *ng*-path identified by the algorithm as dominated by a prefix of an *ng*-path $(o, w_1, \ldots, w_m = v_k)$ already belonging to the DPS of p and a suffix of this *ng*-path.

We denote by $S(L)$ the DPS of an *ng*-path $p(L)$, represented by label L. This DPS contains $p(L)$ and all *ng*-paths that are not represented by a label at node $n(L)$ at the end of the algorithm, because one of their prefixes was dominated during the course of the algorithm by a prefix of an *ng*-path in this DPS. Observe that, in the above definition, a DPS contains *ng*-paths. Because it is time consuming to check the feasibility of an extension of a dominated *ng*-path, we rather use a weaker version of this definition by also accepting relaxed *ng*-paths. Observe also that the definition of a DPS implies that all *ng*-paths dominated by prefixes of the *ng*-paths in this DPS are known. Given that these prefixes may continue to dominate *ng*-paths after the creation of a label L, set $S(L)$ might not be complete when L is created and might be updated afterwards.

Figure 1 gives an example of a DPS $S(L)$ for a label L associated with the *ng*-path $p(L) = (o, 1, 2, 3, 4)$. It is assumed that the *ng*-paths (o, 1, 6), (o, 8, 9), (o, 9, 2), and (o, 5, 6, 7, 3), i.e, the four dashed paths depicted in Fig. 1 that are red, black, blue, and green, are dominated by *ng*-paths (o, 5, 6), (o, 9), (o, 1, 2), and (o, 1, 2, 3), respectively. Because (o, 9, 2) is dominated by the prefix (o, 1, 2) of $p(L)$, the *ng*-path $(o, 9, 2) \oplus (2, 3, 4) = (o, 9, 2, 3, 4) \in S(L)$. Furthermore, because ($o$, 8, 9) is dominated by the prefix (o, 9) of (o, 9, 2), the *ng*-path (o, 8, 9, 2, 3, 4) also belongs to $S(L)$. Three other *ng*-paths, including $p(L)$, belong to $S(L)$ as listed in this figure. Table 1 completes this example by presenting the $S(L)$ set of every *ng*-path $p(L)$ represented by a non-dominated label L in the network, as shown in Fig. 1. The last column in this table gives for each *ng*-path $p(L)$ the subset of customers $C(L)$ visited by all *ng*-paths in $S(L)$. The utility of these subsets will be discussed below.

[2] Note that a path is considered as a prefix of itself.

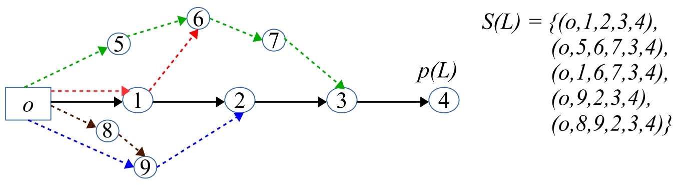

Fig. 1 Example of a DPS $S(L)$ for *ng*-path $p(L) = (o, 1, 2, 3, 4)$

Table 1 Sets $S(L)$ and $C(L)$ for the non-dominated *ng*-paths of the example

$p = p(L)$	$S(L)$	$C(L)$
(o)	$\{(o)\}$	$\emptyset$
$(o, 1)$	$\{(o, 1)\}$	$\{1\}$
$(o, 5)$	$\{(o, 5)\}$	$\{5\}$
$(o, 8)$	$\{(o, 8)\}$	$\{8\}$
$(o, 9)$	$\{(o, 9), (o, 8, 9)\}$	$\{9\}$
$(o, 1, 2)$	$\{(o, 1, 2), (o, 8, 9, 2), (o, 9, 2)\}$	$\{2\}$
$(o, 5, 6)$	$\{(o, 5, 6), (o, 1, 6)\}$	$\{6\}$
$(o, 5, 6, 7)$	$\{(o, 5, 6, 7), (o, 1, 6, 7)\}$	$\{6, 7\}$
$(o, 1, 2, 3)$	$\{(o, 1, 2, 3), (o, 8, 9, 2, 3), (o, 9, 2, 3),$ $(o, 5, 6, 7, 3), (o, 1, 6, 7, 3)\}$	$\{3\}$
$(o, 1, 2, 3, 4)$	$\{(o, 1, 2, 3, 4), (o, 5, 6, 7, 3, 4), (o, 1, 6, 7, 3, 4),$ $(o, 9, 2, 3, 4), (o, 8, 9, 2, 3, 4)\}$	$\{3, 4\}$

In the proposed algorithm, a partial relaxed path can be discarded in two cases. The first case can occur when a partial *ng*-path $p(L) = (o, v_1, \ldots, v_k)$, with $v_k \neq \bar{o}$ and represented by label L, is extended to a node v_{k+1} that has already been visited, i.e., $v_{k+1} = v_j$ for some $j \in \{1, \ldots, k-1\}$. The *ng*-path $q = (o, v_1, \ldots, v_k, v_{k+1})$ resulting from this extension is, therefore, a relaxed path that may be discarded. However, one cannot exclude this extension for the sole reason that it yields a cycle. Indeed, there might exist an *ng*-path $p' = (o, w_1, \ldots, w_m = v_k) \in S(L) \setminus \{p(L)\}$ that can be feasibly extended to v_{k+1} without yielding a cycle (i.e., $v_{k+1} \neq w_j$ for all $j \in \{1, \ldots, m\}$). Given that p' or one of its prefixes has been dominated and discarded, $p(L)$ must be kept to ensure the exactness of the algorithm. This issue is illustrated through an example, as shown in Fig. 2. In this example, $p(L) = (o, 1, 2, 3)$, $q = (o, 1, 2, 3, 1)$, and $p' = (0, 4, 2, 3)$. We assume that $4 \notin \mathrm{NG}_2$, $(o,\ 1,\ 2)$ dominates $(o,\ 4,\ 2)$, and the (non-generated) paths p' and $q' = (o, 4, 2, 3, 1)$ are feasible. Clearly, q' is dominated by q, but if q is discarded, it is not possible to extend q' to possibly yield a shortest o-$\bar{o}$-*ng*-path as q' has not been generated. Nevertheless, the extension of a path $p(L) = (o, v_1, \ldots, v_k)$ to a previously visited node v_{k+1} can be forbidden if every *ng*-path in $S(L)$ contains v_{k+1}.

The second case can arise when two partial *ng*-paths $p(L_1)$ and $p(L_2)$, represented by labels L_1 and L_2, respectively, end at the same node $n(L_1) = n(L_2)$ and L_1 does not dominate L_2 for the only reason that $M(L_1) \nsubseteq M(L_2)$. If the customers in

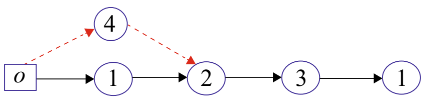

Fig. 2 Example of a relaxed path (o, 1, 2, 3, 1) that cannot be discarded

$M(L_1)\backslash M(L_2)$ have been visited by $p(L_2)$, then $p(L_2)$ may be discarded, because any feasible extension χ of $p(L_2)$ that contains a node in $M(L_1)\backslash M(L_2)$ results in a path $p(L_2) \oplus \chi$ that contains a cycle. However, as in the first case, this property must not only hold for $p(L_2)$, but also for all ng-paths in $S(L_2)$.

In both cases, the complete knowledge of the ng-paths in the DPS is not necessary. Indeed, in the first case, it is only important to know if v_{k+1} has been visited by all ng-paths in a DPS. For the second case, we need to know if the nodes in $M(L_1)\backslash M(L_2)$ have been visited by all ng-paths in a DPS. Therefore, for each label L, instead of storing the set $S(L)$, we store the customer nodes $C(L)$ that are common to all paths in $S(L)$. This set of nodes is given by $C(L) = \bigcap_{p\in S(L)} W(p)$, where $W(p)$ is the set of customer nodes visited along path p. Like for $S(L)$, the composition of a node subset $C(L)$ might evolve throughout the algorithm. In fact, nodes in $C(L)$ may be removed as new ng-paths belonging to $S(L)$ are identified. Table 1 provides the $C(L)$ subsets obtained at the end of the algorithm assuming that only the ng-paths discussed in the example have been generated.

Using the $C(L)$ subsets, the above discussion on the two cases when a partial relaxed path can be discarded results in the following modifications to the labeling algorithm for solving the ng-SPPRC:

1. A label L is not extended along an arc $(n(L), j)$ if $j \in C(L)$, even if $j \notin M(L)$.
2. In the dominance test, condition (19) is relaxed to $M(L_1) \subseteq M(L_2) \cup C(L_2)$.

The conditions $j \in C(L)$ and $M(L_1) \subseteq M(L_2) \cup C(L_2)$ are valid only if the subsets $C(L)$ and $C(L_2)$ do not diminish after they have been tested. This will be guaranteed through a more restrictive dominance rule.

In the modified algorithm, a label is defined as $L = (n(L), Z(L), D(L), T(L), M(L), C(L))$. In the initial label L_o, $C(L_o) = \emptyset$. A label L with $n(L) = i$ is extendable along an arc $(i,j) \in A$ only if $j \notin M(L) \cup C(L)$. When extending label L along this arc to create a new label L', its components $n(L')$, $Z(L')$, $D(L')$, $T(L')$, and $M(L')$ are computed using the extension functions (10)–(14). The computation of $C(L')$ is special. Indeed, it is initially computed when extending L along (i, j) as follows:

$$C(L') = C(L) \cup \{j\}. \tag{20}$$

Then, every time that L' dominates a label L'', $C(L')$ is updated as follows:

$$C(L') = C(L') \cap C(L''). \tag{21}$$

The dominance rule stipulates that L_1 dominates L_2 if

$$n(L_1) = n(L_2) \tag{22}$$

$$Z(L_1) \leq Z(L_2) \tag{23}$$

$$D(L_1) \leq D(L_2) \tag{24}$$

$$T(L_1) \leq T(L_2) \tag{25}$$

$$M(L_1) \subseteq M(L_2) \cup C(L_2) \tag{26}$$

$$L_1 \text{ is not yet extended or } C(L_1) \subseteq C(L_2). \tag{27}$$

Again, all dominated labels are discarded except when two or more labels dominate each other (one of them must be kept). Note that condition (27) ensures that, once L_1 has been extended, set $C(L_1)$ does not change due to newly dominated paths added to set $S(L_1)$. This is necessary to guarantee the validity of (20)–(21) and, therefore, that of every set $C(\hat{L})$ associated with a label $\hat{L}$ resulting from an extension of L_1.

Observe that, without condition (27), the modified algorithm would never generate more labels than the standard *ng*-path labeling algorithm. It would, in general, generate less labels due to the stricter rule for extension and the relaxed condition (26). However, with this condition that restricts the dominance rule, it might happen that the new algorithm generates more labels.

Dominance rule (22)–(27) is valid in the sense stated in the following proposition. In this proposition, we also associate with each label L the set $W(L) \subseteq N$ of the customer nodes visited along path $p(L)$.

Proposition 3 *Let $p_2 = (v_0 = o, v_1, v_2, \ldots, v_k)$ be a feasible path from o to a node $v_k \in V$ that would be associated with a label L_2 if it was generated. The labeling algorithm with dominance rule (22)–(27) generates at least one ng-path p_1 associated with a non-discarded label L_1, such that $n(L_1) = n(L_2)$, $p_2 \in S(L_1)$, $D(L_1) \leq D(L_2)$, $T(L_1) \leq T(L_2)$, $M(L_1) \subseteq W(L_2)$, and $Z(L_1) \leq Z(L_2)$.*

Proof Let $p_2^j = (v_0, v_1, \ldots, v_j)$ for $j = 0, 1, \ldots, k$, be the prefixes of $p_2 = (v_0, v_1, \ldots, v_k)$ starting in v_0 and denote by L_2^j, $j = 0, 1, \ldots, k$, the associated labels if they were generated. The proof proceeds by induction on the value of k. The case $k = 0$ is trivial because $p_1 = p_2 = (o)$ is the only *ng*-path generated that ends in node o. Now, assume that there exists an *ng*-path p_1^{j-1} associated with a label L_1^{j-1}, such that

$$n(L_1^{j-1}) = n(L_2^{j-1}) \tag{28}$$

$$p_2^{j-1} \in S(L_1^{j-1}) \tag{29}$$

$$D(L_1^{j-1}) \leq D(L_2^{j-1}) \tag{30}$$

$$T(L_1^{j-1}) \leq T(L_2^{j-1}) \tag{31}$$

$$M(L_1^{j-1}) \subseteq W(L_2^{j-1}) \tag{32}$$

$$Z(L_1^{j-1}) \leq Z(L_2^{j-1}). \tag{33}$$

Because $p_2^{j-1} \in S(L_1^{j-1})$, we have $C(L_1^{j-1}) \subseteq W(L_2^{j-1})$. Furthermore, $v_j \notin M(L_1^{j-1}) \cup C(L_1^{j-1})$ because p_2^j is feasible. Therefore, label L_1^{j-1} can be extended along arc (v_{j-1}, v_j) to yield a label L_1^j with

$$n(L_1^j) = n(L_2^j) \tag{34}$$

$$p_2^j \in S(L_1^j) \tag{35}$$

$$D(L_1^j) = D(L_1^{j-1}) + d_{v_j} \leq D(L_2^{j-1}) + d_{v_j} = D(L_2^j) \leq Q \tag{36}$$

$$\begin{aligned} T(L_1^j) &= \max\{e_{v_j}, T(L_1^{j-1}) + s_{v_{j-1}} + t_{v_{j-1},v_j}\} \\ &\leq \max\{e_{v_j}, T(L_2^{j-1}) + s_{v_{j-1}} + t_{v_{j-1},v_j}\} = T(L_2^j) \leq \ell_{v_j} \end{aligned} \tag{37}$$

$$M(L_1^j) \subseteq M(L_1^{j-1}) \cup \{v_j\} \subseteq W(L_2^j) \tag{38}$$

$$Z(L_1^j) = Z(L_1^{j-1}) + c_{ij} \leq Z(L_2^{j-1}) + c_{ij} = Z(L_2^j). \tag{39}$$

Thus, L_1^j is feasible and satisfies all conditions of the proposition if it is not discarded. In this case $L_1 = L_1^j$. Now, assume that L_1^j is dominated by a feasible label L_3^j and discarded. In this case,

$$n(L_3^j) = n(L_1^j) = n(L_2^j) \tag{40}$$

$$p_2^j \in S(L_3^j) \tag{41}$$

$$D(L_3^j) \leq D(L_1^j) \leq D(L_2^j) \tag{42}$$

$$T(L_3^j) \leq T(L_1^j) \leq T(L_2^j) \tag{43}$$

$$Z(L_3^j) \leq Z(L_1^j) \leq Z(L_2^j). \tag{44}$$

Furthermore, because $p_2^j \in S(L_1^j)$, observe that $C(L_1^j) \subseteq W(L_2^j)$. Thus, $M(L_3^j) \subseteq M(L_1^j) \cup C(L_1^j) \subseteq W(L_2^j)$. Consequently, L_3^j has the same characteristics as L_1^j, i.e., L_3^j can replace L_1^j in relations (34)–(39). If L_3^j is not discarded, then $L_1 = L_3^j$. Otherwise, the process is repeated until obtaining a non-discarded label L_1. This occurs in a finite number of iterations, because the number of possible labels is finite. $\square$

Corollary 4 *If there exists a feasible path from o to $\bar{o}$ that has a negative reduced cost, then the labeling algorithm with dominance rule (22)–(27) generates at least one ng-path from o to $\bar{o}$ that has a negative reduced cost.*

It ensues from this corollary that the proposed labeling algorithm solves the selective PS according to Definition 6.

Finally, note that the above algorithm can easily be converted into a backward labeling algorithm that extends labels backwardly in G starting from an initial label at node $\bar{o}$. Consequently, a bi-directional labeling algorithm (see Righini and Salani 2006) can also be devised to improve computational efficiency.

3.3 Heuristic labeling algorithm

Without condition (27) in the dominance rule, the new labeling algorithm would generate at most the same number of labels as the standard algorithm (and often less) for solving the *ng*-SPPRC. This condition is, however, necessary to ensure the exactness of the algorithm. We propose a heuristic version of the new labeling algorithm that omits condition (27) in the dominance rule and replaces condition (26) by $M(L_1) \subseteq W(L_2)$. This latter condition is a relaxation of (26), because $M(L_2) \cup C(L_2) \subseteq W(L_2)$. The rationale behind this relaxed condition when considered together with the other conditions (22)–(25) is that any feasible extension χ of $p(L_2)$ that is not feasible for $p(L_1)$ would yield an *ng*-path $p(L_2) \oplus \chi$ that contains a cycle. Thus, such an extension does not need to be considered for $p(L_2)$. Nevertheless, Proposition 3 would not be true with this condition, because there might exist an *ng*-path $\tilde{p} \in S(L_2)$ for which χ is feasible, $\tilde{p} \oplus \chi$ does not contain a cycle, and $\tilde{p} \oplus \chi$ is not dominated by $p(L_1) \oplus \chi$.

As proposed by several authors (see Desaulniers et al. 2014), one or several heuristic algorithms can be called first at each or some column generation iterations. If one of them succeeds to find negative reduced cost columns, these columns are added to the RMP before starting a new iteration. Otherwise (i.e., if all heuristics fail to find such columns), the exact version of the labeling algorithm is invoked. The heuristic labeling algorithm presented above will be used in this way.

4 Computational results

In this section, we provide computational evidence of the effectiveness of the selective-pricing strategy when incorporated into a branch-price-and-cut algorithm, namely, the one developed by Pecin et al. (2016) for the VRPTW. This algorithm relies on an *ng*-SPPRC PS to generate negative reduced cost columns. Its labeling algorithm can thus be replaced by the new labeling algorithm introduced in Sect. 3. The algorithm of Pecin et al. (2016) uses three fast heuristic labeling algorithms to generate negative reduced cost columns as much as possible: the first heuristic keeps only the least reduced cost label for each pair of time and load values. The second and third heuristics rely on a network containing a subset of arcs, ensuring that at least 7 arcs (or 12 arcs for the third heuristic) enter and exit each customer node as proposed by Desaulniers et al. (2008). For our tests, these second and third heuristics were each merged with the heuristic proposed in Sect. 3.3, i.e., they apply the relaxed dominance rule described in that section. We can thus compare the performance of two branch-price-and-cut algorithms, namely, the algorithm of Pecin et al. (2016) with the classical *ng*-SPPRC PS (hereafter labeled `StdP`) and the same algorithm but with the selective *ng*-SPPRC PS (labeled `SelP`). For both algorithms, we use neighborhoods containing 15 customers (namely, the customer itself and its 14 closest customers). This is a compromise between the values used in Pecin et al. (2016) (20 for certain instances and 10 for others), although 15 is most probably not the best value for any of the tested instances. Furthermore, both algorithms use the same cuts and cut separation procedure as in Pecin et al. (2016). The cuts considered are the rounded capacity cuts (Laporte and Nobert 1983) and some rank-1 Chvátal–Gomory cuts (see Jepsen et al. 2008; Petersen et al. 2008), namely, limited-memory rank-1 cuts (Pecin et al. 2016, 2017a, b) which have shown to be essential for solving large-sized instances of the capacitated vehicle routing problem and the VRPTW. The former are qualified as *robust* cuts, because handling their dual variables in the PS does not change the nature of the PS. The latter, hereafter called the lm-rank-1 cuts, are said *non-robust*, because they complicate the PS. Consequently, in the algorithms, non-robust cuts are not separated as long as robust cuts can be found. The algorithms are coded in C++ and rely on the CPLEX, version 12.6, linear programming solver.

Preliminary tests showed that selective pricing was not helpful (but neither harmful) for VRPTW instances that are not very hard to solve or for which cycles are not problematic. Consequently, we have decided to concentrate our experiments on the most difficult VRPTW instances from the dataset of Gehring and Homberger (2001) with 200 customers for which cycles can be problematic, namely, the seven instances in classes C2, RC2, and R2 that could not be solved by Pecin et al. (2016) and two additional instances in class RC2. Given that our goal was not to solve to optimality these instances but rather to illustrate the impact of applying selective pricing and the proposed labeling heuristic, we focused only on the root node results. All tests were executed on an Intel Xeon ES-2637 3.5GHz with 128GB RAM, running Linux Oracle Server 6.7.

We performed four sets of experiments to assess the effectiveness of the selective-pricing strategy and the proposed labeling heuristic. The first series of tests aims at comparing the `SelP` and `StdP` algorithms with the currently best-known algorithm, namely, that of Pecin et al. (2016) (labeled `PCDU16`) which does not incorporate selective pricing nor the new labeling heuristic. The second series of experiments compares the performance of the exact labeling algorithm with and without selective pricing by solving the same subset of PS with both algorithms. The third set of tests focuses on the new labeling heuristic and aims at showing its usefulness in both `SelP` and `StdP` algorithms. Finally, the last experiments allow to compare the lower bounds obtained at the root node when applying each algorithm.

In the first experiments, we compared the results of the algorithms `SelP`, `StdP`, and `PCDU16`. These experiments were conducted as follows. First, we solved the root node for each instance using the `SelP` algorithm. The algorithm was stopped when one of the following two conditions was met: (1) no more cuts were found and (2) the time required by the labeling algorithm to solve the PS at a given iteration exceeded the minimum between 200 s and three times the time required to solve the last PS before adding non-robust cuts. This second condition indicates that solving the PS becomes too difficult and it is preferable to stop adding cuts that increase the difficulty of solving the PS. In any case, the lower bound achieved at the root node is equal to the value of the RMP at the last iteration when the PS was solved using the exact labeling algorithm and no negative reduced cost columns were generated. Then, we ran the `StdP` algorithm and stopped it when it reached the lower bound obtained by `SelP`. For `PCDU16`, we simply collected from the experimental results of Pecin et al. (2016) the time it took to achieve the same bound or the best bound if the same bound was not reached.

The results of these experiments are reported in Table 2. The columns *Time* give the total computational times required by the three algorithms. The columns *Gap* indicate the optimality gap (in percentage) between the computed lower bound and the upper bound corresponding to the cost of the best solution found by the hybrid genetic algorithm of Vidal et al. (2013). The last two columns specify the time reduction in percentage yielded by `SelP` with respect to the other two algorithms. When comparing `SelP` with `StdP`, we observe that `SelP` yields an average time gain of 16.3% for these instances. This average gain is modest, but the results show that a substantial time reduction of 32.7% can be achieved for one instance, while no time increase has been recorded. Next, when comparing `SelP` with `PCDU16` for the nine tested instances, we compute an average time reduction of 35.4% and an impressive maximum reduction of 59.9%, for instance, RC2-2-10. Note that the average reduction should be larger, because `PCDU16` did not reach the lower bound achieved by `SelP`, for instance, R2-2-8 and RC2-2-8. In fact, for instance, R2-2-8, the solution process was killed after running for 4.5 days. For RC2-2-8, the heuristic separation procedure could not find any violated cut. We observe that `SelP` yielded a time increase for only one instance, namely, RC2-2-3. This time increase is,

Table 2 Results of the three algorithms

`Instance`	`SelP`		`StdP`		`PCDU16`		Time gain (%)	
	Time (s)	Gap (%)	Time (s)	Gap (%)	Time (s)	Gap (%)	vs `StdP`	vs `PCDU16`
C2-2-3	19,809	0.70	22,633	0.70	24,189	0.70	12.4	18.1
C2-2-4	51,273	1.82	54,735	1.82	110,540	1.82	6.3	53.6
R2-2-4	90,186	0.18	122,627	0.18	141,827	0.18	26.5	36.4
R2-2-8	224,930	0.39	240,499	0.39	396,535	0.52	6.5	43.3
RC2-2-3	36,316	0.02	42,828	0.02	34,425	0.02	15.2	-5.5
RC2-2-4	224,861	0.26	333,896	0.26	357,851	0.26	32.7	37.2
RC2-2-8	42,042	0	56,466	0	83,057	0.04	25.5	49.4
RC2-2-9	144,597	0.89	174,460	0.89	196,346	0.89	17.1	26.4
RC2-210	133,951	1.14	140,123	1.14	334,078	1.14	4.4	59.9
Average							16.3	35.4

however, very small. Consequently, we can say that `SelP` clearly outperforms `PCDU16` for the tested instances.

The second set of experiments, focusing on the exact labeling algorithms with and without selective pricing, was performed on the five instances for which `SelP` required less than 100,000 s, as shown in Table 2. For each of these instances, we ran again `SelP` and collected some statistics about the solution process of the PS when the exact labeling algorithm with selective pricing is executed at ten specific iterations, namely, the last five iterations before starting to generate lm-rank-1 cuts when the exact pricing algorithm was executed (labeled `Averages without lm-rank-1 cuts`) and the last five iterations of the whole solution process (labeled `Averages with lm-rank-1 cuts`). At these iterations, we also execute the standard labeling algorithm on the same PS, but the (potentially) found columns are not inserted in the RMP and its computational time is stored separately. The results of these experiments are given in Table 3. In this table, we report: under columns labeled NL, the average number of non-dominated labels generated by each algorithm; under columns labeled T, the average computational time (in seconds) spent by each algorithm; under columns labeled RC, the average minimum reduced cost computed by each algorithm; and under the column labeled C, the average number of binding lm-rank-1 cuts. The results show that, compared to `SelP`, the average minimum reduced cost obtained with `StdP` is much smaller in five cases, smaller in two other cases, and identical for the last three cases. This highlight that, in most cases, convergence has not completed with `StdP`, indicating that the MP is not solved to optimality, as mentioned in the first remark of Sect. 2. In fact, the minimum reduced cost computed with `StdP` is always negative except for the three cases C2-2-3, C2-2-4, and RC2-2-8 without lm-rank-1 cuts. Consequently, these results show that selective pricing was effective to abort the execution of the algorithm and that would have continued (most probably for several iterations in some cases); otherwise, if the standard pricing had been used. Looking at the average number of generated non-dominated labels, we observe that, on average, less labels are generated by the selective-pricing algorithm in seven cases and more in the other three cases. The variations are, however, relatively small in general (between -10.6 and 4.1% except for the case RC2-2-3 with lm-rank-1 cuts, where the number of labels increased by more than 33% on average). In terms of computational time, there is no clear dominance between the two algorithms; however, on some cases with the largest average computational times induced by the handling of the lm-rank-1 cut duals, selective pricing seems to take the lead by some interesting margins.

The third series of experiments consisted of solving the root node relaxation considering only capacity cuts using four different algorithms: `SelP`, `StdP`, and these two algorithms without incorporating the new labeling heuristic. These tests were run on the nine instances considered, as shown in Table 2. For each of these instances, the four algorithms produced almost the same lower bound at the end of the solution process, and therefore, it is fair to compare their computational times. The results of these tests are synthesized in Table 4. For each algorithm and each instance, we report the computational time in seconds. Furthermore, we indicate in the fourth and seventh columns the relative gain in time obtained using the new

Table 3 Average results over the last five exact pricings

Instance	Averages without lm-rank-1 cuts						Averages with lm-rank-1 cuts						
	SelP			StdP			SelP			StdP			*C*
	NL	*T*	RC	NL	*T*	RC	NL	*T*	NL	RC	*T*	RC	
C2-2-3	18,231.8	17.4	−4.2	18,920.2	17.5	−4.2	152,081.4	56.9	−1.3	170,111.6	65.5	−5.6	173.0
C2-2-4	102,644.2	54.0	−1.0	106,728.8	53.9	−1.0	448,918.8	178.7	−1.2	463518.8	185.3	−7.0	222.8
R2-2-4	48,608.6	29.7	−0.7	46,690.2	29.5	−21.5	69,987.8	45.2	−0.3	70287.2	44.4	−41.6	116.2
RC2-2-3	26,567.6	12.3	−1.0	27,280.2	12.4	−76.1	121,248.6	52.7	−0.1	91002.6	33.8	−57.0	262.2
RC2-2-8	70,244.8	9.4	−2.5	68,255.2	9.1	−2.5	394,500.0	73.1	−0.1	396,803.2	74.1	−19.0	126.0

Table 4 Results with capacity cuts only

Instance	SelP			StdP			Time gain (%)	
	Time (s) with Heu	Time (s) no Heu	Time gain (%)	Time (s) with Heu	Time (s) no Heu	Time gain (%)	SelP vs with Heu	StdPno Heu
C2-2-3	8244	8356	1.3	6696	9113	26.5	−23.1	8.3
C2-2-4	28,815	35,990	19.9	25,532	38,651	33.9	−12.9	6.9
R2-2-4	29,622	33,818	12.4	33,572	36,664	8.4	11.8	7.8
R2-2-8	40,815	68,610	40.5	38,635	65,152	40.7	−5.6	−5.3
RC2-2-3	11,972	12,584	4.9	11,707	13,342	12.3	−2.3	5.7
RC2-2-4	54,189	86,622	37.4	43,677	103,835	58.0	−24.1	16.6
RC2-2-8	7381	7618	3.1	7470	8121	8.0	1.2	6.2
RC2-2-9	9570	11,493	16.7	9400	9842	4.5	−1.8	−16.8
RC2-210	36,286	29,138	−24.5	29,385	31,573	6.9	−23.5	7.7
Average			12.4			22.1	−8.9	4.1

labeling heuristic in SelP and StdP, respectively. Finally, the last two columns give the time reduction in percentage yielded by SelP with respect to StdP with and without the new heuristic, respectively. From these results obtained for a small subset of very-hard-to-solve instances, we first observe that using the heuristic is always profitable except for one case (RC2-210 with SelP). When used in SelP (resp. StdP), this heuristic reduces the average computational time by 12.4% (resp. 22.1%) and the reduction can be quite large (up to 58%) for certain instances. When looking at the time gains given in the next-to-last column, we observe that, compared to the standard labeling algorithm, selective pricing deteriorates the average computational time by 8.9% on average. This shows that selective pricing can be helpful when the PS is very hard to solve, that is, when non-robust cuts are generated in our case (see the results in Table 2).

Finally, we performed experiments to check if, as stated in the second remark of Sect. 2, SelP can yield at the root node better lower bounds than those obtained by StdP. We solved again the same nine instances with both exact pricing algorithms but without adding any cuts. The results obtained from these tests are reported in Table 5. For each instance and each pricing algorithm, we provide the lower bound achieved at the root node and the required computational time in seconds. We observe that, for six of these nine instances, both pricing algorithms yield the same lower bound and that the lower bound increase obtained using SelP is marginal for the other three instances. This is not surprising given that many columns are generated by the first heuristic algorithm which does not involve any specific strategy to discard paths containing cycles like the selective-pricing strategy. These

Table 5 Lower bounds achieved without cuts

Instance	SelP		StdP	
	LB	Time (s)	LB	Time (s)
C2-2-3	1736.5	6966.7	1736.5	7821.3
C2-2-4	1645.8	31,172.4	1645.8	28,856.2
R2-2-4	1909.6	36,896.4	1909.5	35,678.6
R2-2-8	1802.9	40,760.4	1802.9	46,438.6
RC2-2-3	2186.7	11,813.6	2186.0	12,182.3
RC2-2-4	1815.8	47,603.6	1815.6	55,316.7
RC2-2-8	2130.7	9138.2	2130.7	11,386.2
RC2-2-9	2048.7	9271.2	2048.7	8874.2
RC2-210	1933.5	23,398.7	1933.5	22,772.6

results show again that `SelP` does not always yield faster computational times given that the PS are not too difficult to solve when no cuts have been generated.

5 Conclusion

In this paper, we introduced a new paradigm, called selective pricing, that can be applied to improve the solution process of hard-to-solve VRPs by branch-price-and-cut when a relaxation of the PS is used to reduce its complexity. This new paradigm requires the development of a labeling algorithm specific to the PS which can find a (relaxed or not) route with a negative reduced cost when at least one feasible route with a negative reduced cost exists or prove that no such feasible routes exist even if some relaxed routes have a negative reduced cost. To illustrate selective pricing, we applied it to a branch-price-and-cut algorithm for the VRPTW, where the relaxed PS is an *ng*-SPPRC. We developed a labeling algorithm and showed through computational experiments that it can yield significant time reductions (up to 32%) to reach a good lower bound on certain very-hard-to-solve VRPTW instances with 200 customers. We also introduced a new labeling heuristic which reduces the overall computational times. Combining selective pricing and this heuristic produced, for the nine tested instances, an average time reduction of at least 35% compared to a state-of-the-art branch-price-and-cut algorithm.

This new paradigm opens up several research avenues. In particular, instead of using selective pricing to bridge the gap between *ng*-route pricing and elementary route pricing as presented above, one may use it to close the gap between two sets of *ng*-routes defined by different neighborhood sizes. Another avenue would be to devise new labeling algorithms based on the selective-pricing strategy for various VRPs. For instance, one might think about various ways of relaxing the PS for the pickup and delivery problem and, consequently, of defining new labeling algorithms that might yield improved lower bounds if selective pricing is well-defined. Finally, although the computational results presented in this paper are promising, it is a first attempt at using selective pricing for improving the performance of a branch-price-

and-cut algorithm. We believe that new ideas can be proposed to enhance our work. For example, it might be interesting to devise a column management procedure that would remove relaxed routes from the RMP in the hope to not generate them again due to selective pricing. This mechanism would help to obtain better lower bounds.

Acknowledgements This work has been partially financed by the Natural Sciences and Engineering Research Council of Canada (NSERC), the Fonds de recherche du Québec-Nature et technologies (FRQNT) and the GERAD.

References

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

Barnhart C, Johnson E, Nemhauser G, Savelsbergh M, Vance PH (1998) Branch-and-price: column generation for solving huge integer programs. Oper Res 46(3):316–329

Battarra M, Cordeau JF, Iori M (2014) Pickup-and-delivery problems for goods transportation. In: Toth P, Vigo D (eds) Vehicle routing: problems, methods, and applications, MOS-SIAM series on optimization, vol 6, 2nd edn. SIAM, Philadelphia, pp 161–191

Cherkesly M, Desaulniers G, Laporte G (2015) Branch-price-and-cut algorithms for the pickup and delivery problem with time windows and LIFO loading. Transp Sci 49(4):752–766

Desaulniers G, Lessard F, Hadjar A (2008) Tabu search, partial elementarity, and generalized k-path inequalities for the vehicle routing problem with time windows. Transp Sci 42(3):387–404

Desaulniers G, Madsen O, Ropke S (2014) The vehicle routing problem with time windows. In: Toth P, Vigo D (eds) Vehicle routing: problems, methods, and applications, MOS-SIAM series on optimization, vol 5, 2nd edn. SIAM, Philadelphia, pp 119–159

Desrochers M, Desrosiers J, Solomon M (1992) A new optimization algorithm for the vehicle routing problem with time windows. Oper Res 40:342–354

Desrosiers J, Soumis F, Desrochers M (1984) Routing with time windows by column generation. Networks 14:545–565

Doerner K, Salazar-González J (2014) Pickup-and-delivery problems for people transportation. In: Toth P, Vigo D (eds) Vehicle routing: problems, methods, and applications, MOS-SIAM series on optimization, vol 7, 2nd edn. SIAM, Philadelphia, pp 193–212

Dror M (1994) Note on the complexity of the shortest path models for column generation in VRPTW. Oper Res 42:977–979

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

Gehring H, Homberger J (2001) A parallel two-phase metaheuristic for routing problems with time windows. Asia Pac J Oper Res 18:35–47

Gendreau M, Jabali O, Rei W (2014) Stochastic vehicle routing problems. In: Toth P, Vigo D (eds) Vehicle routing: problems, methods, and applications, MOS-SIAM series on optimization, chapter 8, vol 8, 2nd edn. SIAM, Philadelphia, pp 213–239

Irnich S, Desaulniers G (2005) Shortest path problems with resource constraints. In: Desaulniers G, Desrosiers J, Solomon M (eds) Column generation, vol 2. Springer, Berlin, pp 33–65

Irnich S, Villeneuve D (2006) The shortest path problem with resource constraints and k-cycle elimination for $k \geq 3$. INFORMS J Comput 18(3):391–406

Irnich S, Schneider M, Vigo D (2014) Four variants of the vehicle routing problem. In: Toth P, Vigo D (eds) Vehicle routing: problems, methods, and applications, MOS-SIAM series on optimization, vol 9, 2nd edn. SIAM, Philadelphia, pp 241–271

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

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

Lübbecke M, Desrosiers J (2005) Selected topics in column generation. Oper Res 53(6):1007–1023

Pecin D, Contardo C, Desaulniers G, Uchoa E (2016) New enhancements for the exact solution of the vehicle routing problem with time windows. Technical Report G-2016-13, Cahiers du GERAD

Pecin D, Pessoa A, Poggi M, Uchoa E (2017a) Improved branch-cut-and-price for capacitated vehicle routing. Math Progr Comput 9:61–100

Pecin D, Pessoa A, Poggi M, Uchoa E, Haroldo S (2017b) Limited memory rank-1 cuts for vehicle routing problems. Oper Res Lett 45:206–209

Petersen B, Pisinger D, Spoorendonk S (2008) Chvátal-Gomory rank-1 cuts used in a Dantzig-Wolfe decomposition of the vehicle routing problem with time windows. Veh Routing Probl Latest Adv New Chall 20(43):397–419

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

Toth P, Vigo D (eds) (2014) Vehicle routing: problems, methods, and applications, MOS-SIAM series on optimization, 2nd edn. SIAM, Philadelphia

Vidal T, Crainic T, Gendreau M, Prins C (2013) A hybrid genetic algorithm with adaptive diversity management for a large class of vehicle routing problems with time windows. Comput Oper Res 40:475–489