INFORMS Journal on Computing

Publication details, including instructions for authors and subscription information:
http://pubsonline.informs.org

New Enhancements for the Exact Solution of the Vehicle Routing Problem with Time Windows

Diego Pecin, Claudio Contardo, Guy Desaulniers, Eduardo Uchoa

To cite this article:
Diego Pecin, Claudio Contardo, Guy Desaulniers, Eduardo Uchoa (2017) New Enhancements for the Exact Solution of the Vehicle Routing Problem with Time Windows. INFORMS Journal on Computing 29(3):489-502. https://doi.org/10.1287/ijoc.2016.0744

http://pubsonline.informs.org/journal/ijoc/

INFORMS JOURNAL ON COMPUTING
Vol. 29, No. 3, Summer 2017, pp. 489–502
ISSN 1091-9856 (print), ISSN 1526-5528 (online)

New Enhancements for the Exact Solution of the Vehicle Routing Problem with Time Windows

Diego Pecin,[a,b] **Claudio Contardo,**[b,c] **Guy Desaulniers,**[a,b] **Eduardo Uchoa**[d]

[a] Département de mathématiques et de génie industriel, Polytechnique Montréal, Montréal QC H3T 1J4, Canada; [b] Group for Research in Decision Analysis (GERAD), HEC Montréal 3000, QC H3T 2A7, Canada; [c] Département de management et technologie, ESG UQÀM, Montréal, QC H2X 3X2, Canada; [d] Departamento de engenharia de produção, Universidade Federal Fluminense, Niterói, RJ, 24220-900, Brazil

Contact: diego.pecin@gerad.ca (DP); claudio.contardo@gerad.ca (CC); guy.desaulniers@gerad.ca (GD); uchoa@producao.uff.br (EU)

Received: May 26, 2016
Revised: October 22, 2016
Accepted: November 7, 2016
Published Online: June 26, 2017

http://dx.doi.org/10.1287/ijoc.2016.0744

Abstract. The vehicle routing problem with time windows (VRPTW) consists of finding least-cost vehicle routes to satisfy the demands of customers that can be visited within specific time windows. We introduce two enhancements for the exact solution of the VRPTW by branch-price-and-cut (BPC). First, we develop a sharper form of the limited-memory subset-row inequalities by representing the memory as an arc subset rather than a node subset. Second, from the elementary inequalities introduced by Balas in 1977, we derive a family of inequalities that dominate them. These enhancements are embedded into an exact BPC algorithm that includes state-of-the-art features such as bidirectional labeling, decremental state-space relaxation, completion bounds, variable fixing, and route enumeration. Computational results show that these enhancements are particularly effective for the most difficult instances and that our BPC algorithm can solve all 56 Solomon instances with 100 customers and 51 of 60 Gehring and Homberger instances with 200 customers.

History: Accepted by Karen Aardal, Area Editor for Design and Analysis of Algorithms.

Funding: D. Pecin, C. Contardo, and G. Desaulniers have been partially financed by the Natural Sciences and Engineering Research Council of Canada [Grants 157935-2012 and 435824-2013], the Fonds de recherche du Québec–Nature et technologies [Grant 181909], and GERAD. E. Uchoa has been partially financed by Conselho Nacional de Pesquisa of Brazil (CNPq).

Supplemental Material: The online appendix is available at https://doi.org/10.1287/ijoc.2016.0744.

Keywords: vehicle routing • time windows • branch-price-and-cut • elementary inequalities • limited-memory subset-row inequalities

1. Introduction

The capacitated vehicle routing problem (CVRP) is one of the most studied problems in operations research. Given a set of customers, each with a demand expressed by, e.g., a volume, and a fleet of identical vehicles associated with a single depot, the CVRP calls for finding feasible vehicle routes such that each customer is visited exactly once and the total routing cost (typically proportional to the total traveled distance) is minimized. A route is deemed feasible if the sum of the demands of the visited customers does not exceed the vehicle capacity. The vehicle routing problem with time windows (VRPTW) is a generalization of the CVRP in which the vehicle routes must also satisfy hard time window constraints at the visited customers. It can be formally defined as follows.

Let $V = \{0, 1, \ldots, n+1\}$ be a set of nodes, where nodes 0 and $n+1$ are two copies of the depot and $V^+ = V\backslash\{0, n+1\}$ is the set of customers. With each node $i \in V^+$ are associated a demand $d_i > 0$, a time window $[e_i, l_i]$, $0 \le e_i \le l_i$, and a service time $s_i > 0$. We also associate time windows $[e_0, l_0] = [e_{n+1}, l_{n+1}] = [0, H]$ with the depot nodes, where $H > 0$ represents the planning horizon length. Furthermore, to simplify notation, we introduce $d_0 = d_{n+1} = s_0 = s_{n+1} = 0$. We assume that an unlimited fleet of identical vehicles of capacity Q is available to service these customers. For each pair of nodes $(i, j) \in V \times V$, denote by c_{ij} and t_{ij} the traveling cost and traveling time from i to j, respectively, with $c_{ij}, t_{ij} \ge 0$. Let $A = \{(i, j)\colon i \in V\backslash\{n+1\}, j \in V\backslash\{0\}, i \ne j, d_i + d_j \le Q, e_i + s_i + t_{ij} \le l_j\}$ be the set of feasible arcs. A route $r = (i_0 = 0, i_1, \ldots, i_k, i_{k+1} = n+1)$ visiting $k \ge 1$ customers $i_1, i_2, \ldots, i_k \in V^+$ is said to be feasible if (i) it satisfies vehicle capacity, i.e., $\sum_{h=1}^{k} d_{i_h} \le Q$; (ii) the earliest start of service time T_h at every visited customer node i_h, $1 \le h \le k$, falls within the corresponding time window, i.e., $e_{i_h} \le T_h \le l_{i_h}$; and (iii) the route end time T_{k+1} does not exceed H. The earliest start of service time T_h is defined recursively as $T_0 = 0$, $T_h = \max\{e_{i_h}, T_{h-1} + s_{i_{h-1}} + t_{i_{h-1}, i_h}\}$ for $h > 0$, and the route end time is given by $T_{k+1} = T_k + s_{i_k} + t_{i_k, n+1}$. Note that a vehicle can arrive at a customer i before the opening e_i of its time window and wait, but it is not acceptable to arrive after its closing l_i. The cost of route r is given by $c_{0i_1} + \sum_{h=1}^{k-1} c_{i_h, i_{h+1}} + c_{i_k, n+1}$. The VRPTW consists of constructing feasible vehicle routes such that each customer is visited exactly once and the total cost is minimized.

1.1. Literature Review

The VRPTW has many applications in the distribution (or collection) of goods to customers. It has been widely studied since the 1970s. Indeed, numerous heuristics and exact solution algorithms have been developed, as surveyed in Desaulniers et al. (2014). Here we briefly review the literature on exact algorithms for the VRPTW.

Since the beginning of the 1990s, most successful algorithms for the VRPTW have been based on column generation. This technique decomposes the problem into a restricted master problem that selects the best routes from a subset of routes and a pricing subproblem that generates new routes to add dynamically to the restricted master problem. In principle, the generated routes should be elementary and thus, the resulting pricing subproblem will correspond to an elementary shortest path problem with resource constraints (ESPPRC) (see Irnich and Desaulniers 2005) that is known to be strongly NP-hard (Dror 1994). In their seminal paper, Desrochers et al. (1992) introduced the first branch-and-price algorithm (i.e., a column-generation algorithm embedded in a branch-and-bound framework) in which the pricing subproblem is a pseudopolynomial relaxation of the ESPPRC that allows the generation of nonelementary routes. Allowing cycles in the pricing subproblem reduces its complexity but also weakens the computed lower bounds in the branch-and-bound search tree.

To improve the lower bounds, two research avenues have been explored. The first consists of adding valid inequalities dynamically to strengthen the linear relaxations encountered in the search tree, yielding a branch-price-and-cut (BPC) algorithm. Kohl et al. (1999) proposed the first family of valid inequalities specific to the VRPTW. These inequalities are called 2-path cuts (2PCs) and impose a lower bound of 2 on the flow entering any subset of customer nodes that cannot be serviced by a single vehicle because of the time windows. Another family of valid inequalities that was very efficient for the VRPTW is called the subset-row cuts (SRCs). These inequalities were introduced by Jepsen et al. (2008) and correspond to Chvátal-Gomory inequalities of rank 1. Their addition produces an important reduction of the integrality gap. Some problems, traditionally beyond the reach of exact methods, now become possible to solve thanks to them. However, these cuts can make the pricing subproblem much more difficult to solve and may even stall the solution process when too many of them are added to the master problem. To alleviate this difficulty, Pecin et al. (2014, 2017a) developed a weaker version of these inequalities for the CVRP, which are called the limited-memory SRCs. One contribution of this paper is to test these inequalities in the VRPTW context and to propose a memory that is defined by a subset of arcs rather than a subset of nodes.

The second line of research aims at strengthening the pricing subproblem. Feillet et al. (2004) developed a labeling algorithm that can handle the ESPPRC when the feasible routes do not contain too many customers. Their experiments clearly show that much better lower bounds can be achieved with an ESPPRC pricing subproblem. Desaulniers et al. (2008) later suggested using heuristics intensively to generate routes, avoiding as much as possible the use of an exact labeling algorithm to solve the ESPPRC. Despite these efforts, solving the ESPPRC exactly remains a bottleneck for the difficult VRPTW instances. Baldacci et al. (2011) thus developed a new relaxation for the pricing subproblem for the CVRP and adapted it to the VRPTW (Baldacci et al. 2012). It allows the generation of routes with cycles, called *ng*-routes, but certain cycles are forbidden according to predefined node neighborhoods. The size of these neighborhoods is used to determine a tradeoff between better lower bounds and the difficulty of solving the subproblem. When the neighborhoods are large enough, only elementary routes are generated but the subproblem is hard to solve. In contrast, small neighborhoods yield easy-to-solve subproblems but weak lower bounds. In practice, medium-sized neighborhoods are sufficient to obtain high-quality bounds without spending too much time solving the pricing subproblem.

The algorithm proposed by Baldacci et al. (2012) is not a BPC algorithm. It can be seen as a three-step method. In the first step, a lower bound is computed using a dual-ascent method based on column and cut generation. Given this lower bound, the corresponding dual solution, and an upper bound (obtained from a heuristic solution), a dynamic programming procedure enumerates a subset of the feasible routes—all routes whose reduced cost with respect to the given dual solution is less than or equal to the gap between the upper and the lower bound. It can be proven that this subset contains the routes of all optimal solutions. Finally, in the third step, the VRPTW is formulated as an integer program in which a binary variable is associated with each route in this subset. This program is then solved using a commercial mixed integer programming (MIP) solver to yield an optimal solution. The performance of this three-step algorithm highly depends on the quality of the lower bound computed in the first step and that of the upper bound. Indeed, if the gap between these bounds is too large, the route enumeration procedure applied in the second step might fail because of the very large number of routes to enumerate.

Several other algorithms have been proposed for the VRPTW, including branch-and-cut and Lagrangean relaxation algorithms. For a review of these algorithms, we invite the interested readers to consult the surveys

of Cordeau et al. (2002), Baldacci et al. (2012), and Desaulniers et al. (2014).

So far, the computational experiments performed for testing the proposed exact algorithms for the VRPTW have concentrated on the Solomon benchmark instances. To test heuristics on larger instances, Gehring and Homberger (2001) introduced instances involving 200–1,000 customers each. To our knowledge, only Kallehauge et al. (2006) have published test results obtained on these instances by an exact algorithm. Their Lagrangean relaxation-based BPC algorithm was able to solve to optimality seven instances involving 200 customers, one instance with 400 customers, and one instance with 1,000 customers.

1.2. Contributions

The contributions of this paper are as follows. First, we propose a state-of-the-art exact BPC algorithm that incorporates various features from the most effective solution algorithms recently developed for the CVRP and the VRPTW. Putting these algorithmic components together and configuring them to obtain the most efficient algorithm is a complex task. Second, we revisit the limited-memory SRCs (more generally, rank-1 Chvátal-Gomory cuts) by defining the memory according to an arc subset instead of a node subset. This redefinition helps reduce the negative impact on the computational time that the dual values of these cuts incur on the labeling algorithm used to solve the column-generation pricing subproblem for the hard-to-solve instances. Third, we derive from the elementary inequalities of Balas (1977) a new family of inequalities that dominate them. Note that these two enhancements are also applicable to other problems that can be formulated as set-partitioning problems and solved by BPC where the pricing subproblem corresponds to an elementary shortest path problem. Finally, through extensive computational experiments, we show the efficiency of the proposed algorithm on the hardest-to-solve Solomon instances with 100 customers and on the Gehring and Homberger instances with 200 customers.

This paper is structured as follows. Section 2 provides a mathematical formulation for the VRPTW. Section 3 presents the rank-1 inequalities and introduces the arc-based memory. Section 4 discusses the variant of the elementary inequalities that we use and how we separate them. Next, the proposed BPC algorithm is described in Section 5. Section 6 reports computational results and Section 7 presents conclusions.

2. Mathematical Formulation

Let Ω be the set of all feasible routes. Denote by c_r the cost of route $r \in \Omega$ and by a_i^r the number of visits to customer $i \in V^+$ in route r. Furthermore, let λ_r be a binary variable indicating whether route r is selected in the solution.

The set partitioning formulation (SPF) of the VRPTW is given by

$$\min \sum_{r\in\Omega} c_r\lambda_r \tag{1}$$

$$\text{subject to: } \sum_{r\in\Omega} a_i^r\lambda_r = 1, \quad \forall\, i \in V^+, \tag{2}$$

$$\lambda_r \in \{0,1\}, \quad \forall\, r \in \Omega. \tag{3}$$

The objective function (1) minimizes the overall cost of the selected routes. Constraints (2) guarantee that each customer is serviced by exactly one route. Finally, constraints (3) force the variables to be binary. The SPF has an exponential number of variables and, in practice, its linear relaxation cannot be solved directly by a linear programming solver. To overcome this difficulty, column generation is usually applied to handle the large number of variables.

Column generation (see, e.g., Barnhart et al. 1998, Lübbecke and Desrosiers 2005) is an iterative method that can be used to solve the linear relaxation of model (1)–(3), which is then called the master problem. At each iteration, it solves the master problem restricted to a small subset of the variables and a pricing subproblem that aims to find negative reduced-cost columns to add to the restricted master problem. The algorithm stops when no such column exists.

For the VRPTW, the subproblem consists of finding a route $r \in \Omega$ with a negative reduced cost $\bar{c}_r = c_r - \sum_{i\in V^+} a_i^r \pi_i$, where π_i, $i \in V^+$ are the dual variables associated with constraints (2). It can be formulated as an ESPPRC defined on network $G = (V, A)$ and involves two resources—time and load—to impose route feasibility with respect to time windows and vehicle capacity. For each arc $(i, j) \in A$, let $\bar{c}_{ij} = c_{ij} - \pi_i$ (with $\pi_0 = 0$); define a binary variable x_{ij} that is equal to 1 if arc (i, j) is used in the route and is 0 otherwise. For each node $i \in V$, define a variable T_i that specifies the service start time at node i if $i \in V^+$, the start time from the depot if $i = 0$, and the arrival time at the depot if $i = n + 1$. With this notation, the pricing subproblem can be formulated as follows:

$$\min \sum_{(i,j)\in A} \bar{c}_{ij}x_{ij} \tag{4}$$

$$\text{subject to: } \sum_{(0,j)\in A} x_{0j} = \sum_{(i,n+1)\in A} x_{i,n+1} = 1, \tag{5}$$

$$\sum_{(i,j)\in A} x_{ij} - \sum_{(j,i)\in A} x_{ji} = 0, \quad \forall\, i \in V^+ \tag{6}$$

$$\sum_{(i,j)\in A} d_i x_{ij} \le Q, \tag{7}$$

$$x_{ij}(T_i + s_i + t_{ij}) \le T_j, \quad \forall\,(i,j) \in A \tag{8}$$

$$e_i \le T_i \le l_i, \quad \forall\, i \in V, \tag{9}$$

$$x_{ij} \in \{0,1\}, \quad \forall\,(i,j) \in A. \tag{10}$$

The objective function (4) seeks to minimize the sum of the reduced costs of the selected arcs. Constraints (5) and (6) define the structure of a path from source node 0 to sink node $n+1$. Vehicle capacity is imposed by constraint (7), whereas time window constraints are enforced through constraints (8) and (9). Assuming that there exist no cycles of zero duration, the uniqueness of the variable T_i for each node $i \in V$ ensures that any solution to (4)–(10) corresponds to a feasible elementary route.

The elementary requirements stem from constraints (2) of the SPF that impose a single visit to each customer and induce a strongly NP-hard pricing subproblem. However, as mentioned in the introduction, path relaxations allowing multiple visits to the same customer in a route r are often used to yield a more tractable pricing subproblem. In this paper, we use the *ng*-route relaxation introduced by Baldacci et al. (2011, 2012) and enlarge the set Ω to include all feasible *ng*-routes. For each customer $i \in V^+$, let $N_i \subseteq V^+$ be the neighborhood of i that is defined as the NG closest customers to i. An *ng*-route can only revisit a customer i (forming a cycle) if it passes first by another customer j such that $i \notin N_j$. The resulting pricing subproblem is a shortest *ng*-path problem with resource constraints (*ng*-SPPRC) and is solved by a labeling algorithm, to be discussed in Section 5.1.

To derive integer solutions, column generation is embedded into a branch-and-cut framework to yield a BPC algorithm. We consider the following rounded capacity cuts (RCCs; Laporte and Nobert 1983) and 2PCs. For every subset of customers $S \subseteq V^+$, the RCCs impose a lower bound $r(S) = \lceil \sum_{i \in S} d_i / Q \rceil$ on the number of vehicles required to service the customers in S. Denoting by $\delta^-(S) = \{(i,j) \in A \mid i \notin S, j \in S\}$ the subset of arcs entering S and by b^r_{ij} the number of times that arc $(i,j) \in A$ is traversed in route $r \in \Omega$, these cuts can be expressed as

$$\sum_{r \in \Omega} \left(\sum_{(i,j) \in \delta^-(S)} b^r_{ij} \right) \lambda_r \geq r(S), \quad \forall S \subseteq V^+. \tag{11}$$

Next, let $\mathcal{U}$ be the set of minimal subsets of V^+ with the property that a single vehicle cannot serve all the customers in it because of the time windows. For every subset of customers $S \in \mathcal{U}$, the 2PCs (Kohl et al. 1999) indicate that at least two vehicles are required to service the customers in S. These cuts can be written as

$$\sum_{r \in \Omega} \left(\sum_{(i,j) \in \delta^-(S)} b^r_{ij} \right) \lambda_r \geq 2, \quad \forall S \in \mathcal{U}. \tag{12}$$

Given that, for a route $r \in \Omega$ and a subset S of customers, $\sum_{(i,j) \in \delta^-(S)} b^r_{ij}$ can be expressed in terms of the pricing subproblem arc-flow variables (i.e., $\sum_{(i,j) \in \delta^-(S)} b^r_{ij} = \sum_{(i,j) \in \delta^-(S)} x_{ij}$), the RCCs and the 2PCs are classified as *robust* cuts according to the classification proposed in Poggi de Aragão and Uchoa (2003). This means that they do not change the pricing subproblem complexity because their dual variables only affect the arc costs in the subproblem. Denoting by α_S, $S \subseteq V^+$, and β_S, $S \in \mathcal{U}$ the duals associated with the RCCs (11) and the 2PCs (12), respectively, the subproblem arc costs become

$$\bar{c}_{ij} = c_{ij} - \pi_i - \sum_{\substack{S \subseteq V^+: \\ (i,j) \in \delta^-(S)}} \alpha_S - \sum_{\substack{S \in \mathcal{U}: \\ (i,j) \in \delta^-(S)}} \beta_S, \quad \forall (i,j) \in A, \tag{13}$$

where π_i is the dual variable associated with constraint (2) for customer i.

In Sections 3 and 4, we present two families of nonrobust cuts whose corresponding dual variables cannot be incorporated in the subproblem arc costs.

3. Rank-1 Inequalities with an Arc Memory

Jepsen et al. (2008) introduced a family of valid inequalities (the SRCs) defined over the SPF. Given a customer subset $S \subseteq V^+$ and a multiplier $p = 1/k$ for some positive integer k ($1 < k < |S|$), the SRC

$$\sum_{r \in \Omega} \left\lfloor p \sum_{i \in S} a^r_i \right\rfloor \lambda_r \leq \lfloor p|S| \rfloor \tag{14}$$

is valid since it can be obtained by a Chvátal-Gomory rounding of the corresponding rows (2). Petersen et al. (2008) experimented with more general rank-1 cuts, where the row corresponding to each customer $i \in S$ has its own multiplier p_i ($0 < p_i < 1$):

$$\sum_{r \in \Omega} \left\lfloor \sum_{i \in S} p_i a^r_i \right\rfloor \lambda_r \leq \left\lfloor \sum_{i \in S} p_i \right\rfloor. \tag{15}$$

Recently, Pecin et al. (2017b) performed a study of the set-partitioning polyhedron and determined that the optimal multiplier vectors for rank-1 cuts with up to five rows are the following.

- For $|S| = 3$, (1/2, 1/2, 1/2), which means that the SRCs with $k = 2$ are already the best possible rank-1 cuts for this case.
- For $|S| = 4$, (2/3, 1/3, 1/3, 1/3) and its permutations.
- For $|S| = 5$, (1/3, 1/3, 1/3, 1/3, 1/3), (2/4, 2/4, 1/4, 1/4, 1/4), (3/4, 1/4, 1/4, 1/4, 1/4), (3/5, 2/5, 2/5, 1/5, 1/5), (1/2, 1/2, 1/2, 1/2, 1/2), (2/3, 2/3, 2/3, 1/3, 1/3), (3/4, 3/4, 2/4, 2/4, 1/4), and its permutations.

When nonelementary routes can be generated, one can also define SRCs (14) with $|S| = 1$. In particular, to forbid routes with cycles, one can apply SRCs (14) with $|S| = 1$ and $p = 1/2$.

Computational experiments by a number of authors made clear that SRCs (more general rank-1 cuts were

only used by Petersen et al. 2008) have a big potential for reducing the integrality gaps of the SPF. However, the cuts defined directly over the SPF variables are nonrobust, each added cut changing the pricing subproblem and making it harder to solve (see Desaulniers et al. 2011). Essentially, the labels in the labeling algorithm used to solve the subproblem should have an additional dimension for each active cut. Consequently, only a limited number of SRCs can be effectively used and the full potential gap reductions cannot always be obtained.

Contrary to the general guideline found in most integer programming textbooks (e.g., Wolsey and Nemhauser 1988) that favors the use of the strongest valid inequalities available, Pecin et al. (2014) proposed a weaker version of the SRCs, called limited-memory SRCs, that reduces their impact on the pricing subproblem. Even if a larger number of them may be required to reach the lower bound obtained with the standard SRCs, they can yield a substantial reduction of the overall computational time.

The limited-memory mechanism proposed by Pecin et al. (2014) associates with each rank-1 cut indexed by k a customer subset $S(k) \subseteq V^+$, a multiplier vector $(p_i^k)_{i \in S(k)}$, and a memory node set $M(k)$ with $S(k) \subseteq M(k) \subseteq V^+$. The idea (somehow inspired by the *ng*-route relaxation) is that, when a route $r \in \Omega$ leaves the memory set $M(k)$, it may "forget" previous visits to nodes in $S(k)$, yielding a coefficient for λ_r in the cut k that may be smaller than the original $\lfloor \sum_{i \in S} p_i^k a_i^r \rfloor$ coefficient. The memory set $M(k)$ is defined during the cut separation procedure as a minimal set preserving the coefficients of the route variables λ_r that have a positive value in the current linear relaxation solution. Those limited-memory rank-1 cuts, while still nonrobust, can be handled much better by the labeling algorithm used for solving the pricing subproblem because an additional dimension is needed only for the labels corresponding to a partial path ending in $M(k)$. This may be very advantageous computationally if $|M(k)| \ll n$, as usually happens. We remark that the weak 3-SRCs proposed and used in Baldacci et al. (2012) correspond to particular limited-memory rank-1 cuts, where $|S(k)| = 3$ and $M(k) = S(k)$.

For the VRPTW instances with wide time windows and loose vehicle capacity that allow long feasible routes (with up to 60 customers), our preliminary tests showed that the size of some memory node sets is too large and yields an intractable pricing subproblem after the addition of a few limited-memory rank-1 cuts. To overcome this difficulty, we propose a sharper definition of the limited memory concept. Given $S \subseteq V^+$, a multiplier vector p of dimension $|S|$, and a memory arc set $AM \subseteq A$, the limited-arc-memory rank-1 cut is defined as

$$\sum_{r \in \Omega} \gamma(S, p, AM, r) \lambda_r \leq \left\lfloor \sum_{i \in S} p_i \right\rfloor, \tag{16}$$

where the coefficient $\gamma(S, p, AM, r)$ of the route variable λ_r, $r \in \Omega$, is computed as described in Algorithm 1. This algorithm can be explained as follows. Whenever a route r visits a node $j \in S$, the multiplier p_j is added to the *state* variable. When *state* ≥ 1, *state* is decremented and γ is incremented. If $AM = A$, the procedure returns a value of γ equal to $\lfloor \sum_{i \in S} p_i a_i^r \rfloor$ and the limited-memory cut would be equivalent to the original cut. On the other hand, if $AM \subset A$, it may happen that r uses an arc $(i, j) \notin AM$ when *state* > 0, causing *state* to be reset to zero and "forgetting" some previous visits to nodes in S. In this case, the returned coefficient may be less than the original coefficient.

Algorithm 1 (Computing the coefficient $\gamma(S, p, AM, r)$ of λ_r in a limited-arc-memory rank-1 cut)

$\gamma \leftarrow 0$, *state* $\leftarrow 0$
For every arc (i, j) in route r (in order) **do**
  If $(i, j) \notin AM$ **then**
    state $\leftarrow 0$
  If $j \in S$ **then**
    state $\leftarrow$ *state* $+ p_j$
    If *state* ≥ 1 **then**
      $\gamma \leftarrow \gamma + 1$, *state* $\leftarrow$ *state* $- 1$
return γ.

The limited-node-memory scheme in Pecin et al. (2014) can be viewed as a particular case of the arc memory, where an AM set is composed of all arcs $(i, j) \in A$ such that j belongs to the corresponding node set M. The advantage of the more general scheme is to allow the cut to have the same coefficients on the relevant route variables (those with positive values) considering fewer arcs in AM and impacting less the pricing subproblem labeling algorithm. As will be discussed in Section 6, computational experiments show that some hard-to-solve instances are much better handled using limited-arc-memory rank-1 cuts. However, not so hard-to-solve instances can be better treated with limited-node-memory cuts.

As for the limited-node-memory cuts, the limited-arc-memory rank-1 cuts are separated by complete enumeration for $|S| \leq 3$ and restricted enumeration for $|S| \in \{4, 5\}$. In the latter case, the procedure explores only the subsets of customers S such that the distance between any pair of customers in S does not exceed a predefined maximum distance. For each subset S and multiplier vector p listed, we check if a violated cut can be found with a full memory. If so, an arc-memory AM of minimal size such that the cut violation remains the same is identified. When a cut defined for the same subset S and vector p but for a different arc memory AM' already exists, this cut is simply enhanced by enlarging its memory to $AM' \cup AM$. Otherwise, a new cut is added for S, p, and AM. Note that we also considered not merging the arc-memory sets, that is, creating a new cut for each arc memory. However, preliminary tests showed that it is better to merge the arc memories.

4. Elementary Inequalities

In this section we propose an extension of the elementary cuts introduced by Balas (1977) for the set-partitioning problem and we describe a heuristic separation algorithm based on local search.

4.1. Definition

The elementary cuts of Balas (1977) are defined as follows. Given a route $r \in \Omega$, let $V^+(r)$ be the set of customers visited by route r. Given a customer subset $C \subset V^+$ and a customer $i \in V^+ \backslash C$, let $\Omega^\perp(i,C) = \{r \in \Omega \mid a_i^r > 0, a_j^r = 0, \forall\, j \in C\}$ be the subset of routes that visit i but no customers in C. Given a route $r \in \Omega$ and a customer $i \in V^+ \backslash V^+(r)$, the following elementary inequality is valid for the SPF:

$$\lambda_r \le \sum_{q \in \Omega^\perp(i, V^+(r))} \lambda_q. \tag{17}$$

This inequality can be interpreted as follows: If route r is used, then the route visiting customer i cannot visit any customer in $V^+(r)$, as at least one customer in $V^+(r)$ would be visited twice.

The elementary inequalities have received little attention since Sherali and Adams (1990) showed that they can be implied by reformulating the set-partitioning problem using the reformulation-linearization technique. This reformulation involves a quadratic number of variables in terms of the size of Ω. For the VRPTW, this would be equivalent to considering variables λ_{rq} representing the use of two different vehicle routes in a single column. Pricing on these variables could, however, rapidly become prohibitive for difficult instances that already pose a computational challenge for solving the single-route pricing subproblem.

Here we describe a new family of inequalities that dominate the elementary inequalities of the form (17). Because they pursue the same purpose, we also call them elementary inequalities. Given a customer set $C \subset V^+$ and a customer $i \in V^+ \backslash C$, we define an elementary inequality as being a rank-1 Chvátal-Gomory cut with multipliers $p_i^C = (|C| - 1)/|C|$ and $p_j^C = 1/|C|$, $\forall\, j \in C$:

$$\sum_{r \in \Omega} \left\lfloor p_i^C a_i^r + \sum_{j \in C} p_j^C a_j^r \right\rfloor \lambda_r \le 1. \tag{18}$$

The following proposition and corollary establish the dominance of inequality (18) over inequality (17) in the case of elementary routes.

Proposition 1. *For a given route $r \in \Omega$ and a customer $i \in V^+ \backslash V^+(r)$, let $\nu(i,r) = \lambda_r - \sum_{q \in \Omega^\perp(i, V^+(r))} \lambda_q$ be the violation of an elementary cut (17). Similarly, let $C = V^+(r)$ and let $\mu(i,r) = \sum_{q \in \Omega} \lfloor p_i^C a_i^q + \sum_{j \in C} p_j^C a_j^q \rfloor \lambda_q - 1$ be the violation of the corresponding elementary cut (18). If Ω contains only elementary routes, then $\mu(i,r) \ge \nu(i,r)$ and this inequality might be strict.*

Proof. Let $\Omega(i) \subseteq \Omega$ be the subset of feasible routes that visit customer i and let $\Omega(i, V^+(r)) = \Omega(i) \backslash \Omega^\perp(i, V^+(r))$ be the subset of routes visiting i and at least one customer in C. If Ω contains only elementary routes, then $\sum_{q \in \Omega(i)} \lambda_q = \sum_{q \in \Omega^\perp(i, V^+(r))} \lambda_q + \sum_{q \in \Omega(i, V^+(r))} \lambda_q = 1$ and $\nu(i,r)$ can be rewritten as $\nu(i,r) = \lambda_r + \sum_{q \in \Omega(i, V^+(r))} \lambda_q - 1$. Because $\lfloor p_i^C a_i^q + \sum_{j \in C} p_j^C a_j^q \rfloor \ge 1$ for all $q \in \Omega(i, V^+(r)) \cup \{r\}$, it follows that $\mu(i,r) \ge \nu(i,r)$.

The inequality might be strict because there might exist a route $q \notin \Omega(i, V^+(r)) \cup \{r\}$ with a strictly positive coefficient in inequality (18). This might be the case, for instance, of a route visiting all customers in $V^+(r)$ and just one more customer in $V^+ \backslash (V^+(r) \cup \{i\})$. □

Corollary 1. *If an inequality (17) is violated for a route $r \in \Omega$, then the corresponding inequality (18) for set $C = V^+(r)$ is also violated.*

Note that the elementary cuts correspond to rank-1 cuts of the form (15). Indeed, with each elementary cut indexed by k and defined for a customer i and a customer subset C, we can associate a customer set $S(k) = C \cup \{i\}$, and a vector of multipliers $(p_j^k)_{j \in S(k)}$ (with $p_i^k = p_i^C$ and $p_j^k = p_j^C$, $\forall\, j \in C$). They can also be combined with a node-memory $M^n(k)$ or an arc-memory $M^a(k)$ to reduce their impact on the computational time required for solving the pricing subproblem. If a limited arc memory is used, the coefficient $\gamma(S(k), p^k, M^a(k), r) = \lfloor p_i^C a_i^r + \sum_{j \in C} p_j^C a_j^r \rfloor$ of a route variable λ_r, $r \in \Omega$, can be computed using Algorithm 1. We observe that the elementary cuts with $|C| = 2, 3, 4$ correspond to the rank-1 cuts of the form (15) with $|S| = 3$, 4, 5 and $p = (1/2, 1/2, 1/2)$, $(2/3, 1/3, 1/3, 1/3)$, $(3/4, 1/4, 1/4, 1/4, 1/4)$ (and their permutations), respectively. In the rest of the paper (except in Section 4.2), we consider the elementary cuts as part of the rank-1 cuts.

4.2. Separation

In vehicle routing applications, rank-1 Chvátal-Gomory cuts have not often been used and have been separated using constrained explicit enumeration; for instance, Jepsen et al. (2008) enumerate all subsets of three customers to find violated SRCs associated with three-customer subsets. Preliminary tests showed that such enumeration becomes prohibitively time consuming for larger subsets of rows. Consequently, we developed a new search strategy especially tailored for separating the elementary inequalities of the form (18). Our algorithm does not rely on explicit enumeration, but rather on local search. Its outline is as follows.

For a given customer $i \in V^+$, let $\Omega(i)$ be again the set of all routes visiting i. Furthermore, let $\Omega^\perp(i)$ be the set of routes that do not visit i. For each route $r \in \Omega^\perp(i)$, define $C(r,i) = V^+(r) \cap (\bigcup_{q \in \Omega(i)} V^+(q))$ as the subset of customers visited in r and by at least one route visiting i. Finally, let $W(i) = \bigcup_{q \in \Omega(i)} V^+(q)$ be the set of all customers that are visited by at least one route visiting customer i. The following local search procedure is executed for each possible pair of customer $i \in V^+$ and route $r \in \Omega^\perp(i)$. For each execution, it starts with an

initial customer subset $C = C(r, i)$ and applies a series of moves in the hope of finding subsets S yielding violated inequalities.

At each iteration of this procedure, three possible neighborhoods are explored:

1. The ADD neighborhood selects a customer $j \in W(i)\backslash(C \cup \{i\})$ and adds it to C.
2. The REMOVE neighborhood selects a customer $j \in C$ and removes it from C.
3. The SWAP neighborhood selects two customers $j \in C$ and $k \in W(i)\backslash(C \cup \{i\})$, adds k to C, and removes j from it.

For each possible move in each neighborhood, the procedure evaluates the violation improvement (the violation being computed as $\mu(i, C) = \sum_{r\in\Omega} \lfloor p_i^C a_i^r + \sum_{j\in C} p_j^C - a_j^r \rfloor \lambda_r - 1)$ and applies the move producing the largest improvement. The search is stopped when no improvement is possible. We found that this strategy performs much better than a simpler first-improvement criterion or a best-improvement criterion on the first neighborhood that strictly improves the violation on the incumbent. Indeed, this procedure with a complete exploration of the neighborhoods in each iteration requires a computational time that is still some orders of magnitude inferior to the time required for the execution of the exact pricing and is capable of finding several violated cuts that would not be found using a simpler strategy. Finally, based on preliminary test results, we chose to limit the cardinality of C to a maximum size (seven for our tests) throughout the procedure.

Our algorithm stores all violated cuts found along each execution of the local search. The number of cuts found may be large. To reduce this number, we proceed in three sequential additional steps. First, the cuts are sorted according to their violation. Second, we check whether any two cuts are equivalent, in which case we arbitrarily discard one. Two cuts are said to be equivalent with respect to the current fractional solution if the coefficients in the two cuts of every route associated with a basic variable are the same. Finally, if there are more than κ cuts left, we keep the κ cuts with the largest violations, where κ is a predefined parameter. After performing a series of preliminary tests to calibrate this parameter, we have chosen to set $\kappa = 150$.

5. A Branch-Price-and-Cut Algorithm for the VRPTW

This section describes the components of the full BPC algorithm that we implemented. Note that the hybrid genetic algorithm of Vidal et al. (2013) is run first to provide a feasible solution and an upper bound on the optimal value. This upper bound is used as discussed in Sections 5.2 and 5.4.

5.1. Solving the Pricing Subproblem

The pricing subproblem can be solved exactly using the following forward labeling algorithm. In this algorithm, a partial or complete path $P = (0, \ldots, i)$, $i \in V^+ \cup \{n+1\}$, is represented by a *label* $L(P) = (\bar{c}(P), n(P) = i, t(P), q(P), \Pi(P), E(P))$ that stores, respectively, its reduced cost, node, earliest start of service time at i, accumulated load (including demand d_i), set of nodes forbidden as immediate extensions because of ng-route restrictions, and vector of states associated with the limited-memory rank-1 cuts with nonzero dual values in the current restricted master problem solution (hereafter called the active nonrobust cuts). The algorithm starts with the single unprocessed label $(0, 0, 0, 0, \varnothing, \mathbf{0})$ representing the null path, an empty vehicle at time zero at the depot. An unprocessed label should be processed by extending it to all its feasible successors. More formally, a label $L(P)$ with $n(P) = i$ is extended along every arc $(i, j) \in A$ such that $j \notin \Pi(P)$, $t(P) + s_i + t_{ij} \le l_j$ and $q(P) + d_j \le Q$, generating a new label

$$L(P') = (\bar{c}(P'), j, \max\{t(P) + s_i + t_{ij}, e_j\}, q(P) + d_j, (\Pi(P) \cap N_j) \cup \{j\}, E(P')),$$

where:

- $E(P')$ is the vector of variable states updated from $E(P)$ after the visit to j according to Algorithm 1. This means that, for the active nonrobust cut k, $E(P')[k]$ first receives the value $E(P)[k]$, then it is reset to zero if $(i, j) \notin M^a(k)$, is increased by p_j^k if $j \in S(k)$, and is decremented by 1 if it becomes larger than or equal to 1;
- $\bar{c}(P') = \bar{c}(P) + \bar{c}_{ij} - \sum_{k\in DEC} \sigma_k$, where $\bar{c}_{ij}$ is the arc reduced cost (13), DEC is the subset of the active nonrobust cuts whose states were decremented by one when calculating $E(P')$, and σ_k is the dual variable associated with cut k. As those dual variables are negative, this last term can be viewed as a penalization on the route. When all labels are processed, the routes with minimum reduced cost are found among the labels P representing paths ending in $n+1$. To retrieve the complete paths, the labels also include a pointer to their unique predecessor.

To avoid enumerating all feasible routes, a dominance rule is applied to identify dominated labels that can be discarded. A label $L(P_1)$ is said to dominate a label $L(P_2)$ if every feasible completion of P_2 is also feasible for P_1 and yields a route whose reduced cost is not greater than that of the route obtained by applying the same completion to P_1. The following five conditions, together, are sufficient to ensure such domination:

(i) $n(P_1) = n(P_2)$, (ii) $t(P_1) \le t(P_2)$, (iii) $q(P_1) \le q(P_2)$,
(iv) $\Pi(P_1) \subseteq \Pi(P_2)$, and
(v) $\bar{c}(P_1) - \sum_{\substack{k\in K:\\ E(P_1)[k]>E(P_2)[k]}} \sigma_k \le \bar{c}(P_2)$,

where K is the set of the indices of the active nonrobust cuts. The second term in the left-hand side of (v) is a bound on the extra penalizations that a completion of P_1 can incur over the same completion of P_2, if it revisits sets $S(k)$ for cuts $k \in K$ in which $E(P_1)[k] > E(P_2)[k]$. Dominance can only occur if the reduced cost of $\bar{c}(P_1)$ is sufficiently less than $\bar{c}(P_1)$ to compensate for the potential extra penalizations. The traditional SRCs or rank-1 cuts have a large potential for disrupting the label dominance by making condition (v) much less likely to be satisfied. The limited-memory mechanism mitigates that difficulty. As the state of given cut k is reset to zero whenever a route passes by an arc not in $M^a(k)$, its dual variable σ_k is involved in fewer label comparisons.

Being the most critical part of the BPC algorithm, the labeling algorithm is enhanced with the following acceleration techniques.

- *Bidirectional Search.* The labeling algorithm can be implemented in a backward manner, starting from node $n+1$ and extending partial paths backwards until reaching node 0. That implementation in itself is not computationally more efficient. However, Righini and Salani (2006) realized that bidirectional search yields significant gains. The idea is to execute the forward algorithm for generating only labels with a time component that does not exceed half of the time horizon H and the backward algorithm for generating labels with a time larger than half of H. The routes with negative reduced costs are obtained by performing an additional concatenating step.
- *Completion Bounds.* In the pricing context, one is only interested in routes with negative reduced cost. Given a partial path $P = (0, \ldots, i)$, let $CB(P)$ be a completion bound on P, that is, a lower bound on the reduced cost of all complete routes that can be obtained as extensions of P. If $CB(P) \geq 0$, then label $L(P)$ can be eliminated. As proposed in Contardo and Martinelli (2014), good completion bounds for the forward algorithm can be obtained by running the backward algorithm considering only a fraction of the nonrobust cuts and underestimating the effect of the remaining nonrobust cuts. Conversely, completion bounds for the backward algorithm are obtained by running the forward algorithm with a similar relaxation.
- *Decremental State Space Relaxation* (*DSSR*). As suggested by Martinelli et al. (2014), we apply DSSR on the neighborhoods N_i, $i \in V^+$. This means that we start by running the labeling algorithm with empty neighborhoods; then we increase them (up to their maximum size NG) to forbid the detected cycles and we run the labeling algorithm again, and so on, until negative reduced cost feasible *ng*-routes are found or it is not possible to remove further cycles without exceeding the NG limit. For most nodes $i \in V^+$, the final N_i set has a cardinality significantly less than NG, yielding performance gains. As suggested in Desaulniers et al. (2008), the convergence of DSSR is accelerated by initializing it with the final neighborhoods obtained at the previous column-generation iteration in the same node of the search tree. The first iteration of a node inherits the sets N_i, $i \in V^+$ from the last iteration of its father node. The neighborhoods are reset to empty only once during the whole algorithm—just before separating nonrobust cuts for the first time at the root node. In this case, the last pricing subproblem is solved a second time starting with empty neighborhoods.
- *Heuristic Labeling.* Even with those enhancements, a single call to the exact pricing algorithm can still be very time consuming for the most challenging instances. Therefore, it is also essential to rely on fast and effective heuristic pricing routines. In our algorithm, we use three different heuristics for each linear relaxation to solve. In order, each heuristic is applied for a certain number of iterations until it fails to generate negative reduced cost routes. The exact labeling algorithm is then called to complete the linear relaxation solution process. The three heuristics correspond to heuristic versions of the above described labeling algorithm. The first keeps only the least-cost label for each pair of time and load values at each node. Proposed by Desaulniers et al. (2008), the second and third heuristics rely on a network containing a subset of the arcs in A, ensuring that a minimum number of arcs (7 in the second heuristic and 12 in the third) exit and enter each customer node.

5.2. Fixing by Reduced Costs

After solving a linear relaxation, the BPC algorithm uses the procedure of Irnich et al. (2010) to fix the flow on some arcs to 0 (i.e., the arcs are removed) according to the reduced costs of the routes traversing them. This procedure requires complete runs of both the forward and backward labeling algorithm. To test if an arc $(i, j) \in A$ can be removed, all pairs of labels—one from the forward algorithm associated with a path ending in i and another from the backward algorithm associated with a path starting in j, such that their concatenation using (i, j) leads to a feasible route—must be considered. If the reduced cost of each of these routes is larger than or equal to the gap between the current upper and lower bounds, then (i, j) cannot appear in any route of an improving solution and can thus be discarded.

5.3. Cutting

After solving each linear relaxation, the BPC algorithm searches for violated robust RCCs (11), separated by the heuristic procedure of Lysgaard (2003). When no violated RCCs are found, violated rank-1 cuts of the forms (16) and (17) are sought using the separation procedure described in Sections 3 and 4.2. Note that

2PCs (12) were considered in preliminary tests, which showed that they were not useful. As explained next, classical clique cuts may also be separated using the routines of Santos et al. (2016) after a successful route enumeration.

5.4. Enumerating Routes and Branching

Given an upper bound derived from a feasible solution, Baldacci et al. (2011, 2012) use a route enumeration approach to close the integrality gap after solving the root node by column and cut generation. The key observation is that a route $r \in \Omega$ can only be part of a solution that improves the best known upper bound if it is elementary and its reduced cost with respect to the computed linear relaxation solution is less than the gap. The enumeration of those elementary routes can be performed by a label-setting algorithm, similar to the one described above, producing a route subset $\hat{\Omega} \subseteq \Omega$. A general MIP solver is then used to solve the SPF restricted to $\hat{\Omega}$, using the upper bound as a cutoff. If the restricted model is infeasible, it proves that the solution providing the upper bound is optimal. Otherwise, the optimal solution of the restricted model is optimal for the complete problem. If the gap is small enough to produce a set $\hat{\Omega}$ of reasonable cardinality this may work very well; that is, the route enumeration approach solves the instance in much less time than traditional branching would take. However, large gaps lead to values of $|\hat{\Omega}|$ that cause the MIP solver to take too much time ($|\hat{\Omega}| > 100{,}000$ is often unpractical).

Contardo and Martinelli (2014) proposed a strategy to profit from route enumeration, even with larger gaps. Routes are enumerated and stored in a pool if $|\hat{\Omega}|$ is less than a few million. The linear relaxation of the resulting MIP is solved by column generation where, instead of using a labeling algorithm, the columns are priced by inspection of the pool. Then, given that non-robust cuts have little impact on the pricing complexity, arbitrary cuts for the SPF (including rank-1 cuts without memory limits and clique cuts) are separated aggressively. These cuts typically substantially increase the lower bound, allowing reductions in the pool size by fixing route variables by reduced costs. When the set of the remaining routes becomes small enough, the restricted model is solved by a MIP solver. The proposed BPC algorithm uses this enumeration strategy, allowing pools with up to 30,000,000 routes.

For difficult instances, there might be too many routes to enumerate, even to store them in a pool. If this is the case, enumeration is aborted and the BPC algorithm proceeds to branching on the flow entering a subset of customers as in Lysgaard et al. (2004). Because deeper nodes in the search tree yield smaller gaps, enumeration is attempted at every node. The overall effect of this hybrid strategy (first used by Pessoa et al. 2009) may be a substantially reduced-size search tree.

6. Computational Experiments

This section provides computational results obtained by the proposed BPC algorithm on the VRPTW instances of Solomon (1987) and Gehring and Homberger (2001). There are 56 100-customer Solomon benchmark instances, which are divided into six classes: C1, RC1, R1, C2, RC2, and R2. The locations of the customers are clustered in the C1 and C2 classes, random in R1 and R2, and mixed in RC1 and RC2. In the C1, RC1, and R1 classes, the time windows are relatively narrow, naturally favoring the appearance of short, near-elementary routes even if the route relaxation used in the pricing subproblem is theoretically poor. In contrast, in the C2, RC2, and R2 classes, the time windows are relatively wide and the vehicle capacity is not very constraining, allowing the generation of long routes (with up to 60 customers in some cases) that may contain many cycles unless a strong route relaxation is used to forbid them explicitly. Designed similarly, the 60 Gehring and Homberger instances with 200 customers are divided into the same six classes (C1, RC1, R1, C2, RC2, and R2). Each class contains 10 instances. The Solomon instances are denoted as ccnn, where cc represents the class and nn the instance number in this class, whereas the Gehring and Homberger instances are denoted cc-snn where $s = 2$ indicates that there are 200 customers.

The BPC algorithm settings are as follows. For all Solomon instances except R208 and all C1, RC1, and R1 Gehring and Homberger instances, we have set $NG = 10$, which means that the neighborhood of each customer used to define the feasible *ng*-routes is composed of its 10 nearest customers. For the other instances where allowing more cycles can be very harmful for the lower bound, we used $NG = 20$. Furthermore, for all instances except the C1, RC1, and R1 Gehring and Homberger instances, we observed that the capacity constraint is not really binding. Therefore, to ease the solution of the pricing subproblem, we did not include this constraint, which was enforced through the RCCs whenever needed. The choice of the memory type for the rank-1 cuts is discussed next.

The BPC algorithm was coded in C++ and solves the linear and integer programs using IBM CPLEX Optimizer 12.6. All experiments were conducted on an Intel Xeon ES-2637 3.5 GHz with 128 GB RAM, running Linux Oracle Server 6.7.

In Section 6.1, we present a comparison between the arc- and the node-memory scheme for the rank-1 cuts. In Section 6.2, we provide computational results showing the usefulness of the elementary cuts. Finally, in Sections 6.3 and 6.4, we report summary results of our experiments on the Solomon and the Gehring and Homberger instances.

6.1. Comparison Between Arc- and Node-Memory Rank-1 Cuts

We performed computational tests to compare the usage of a limited arc memory for rank-1 cuts with that of a node memory. The comparison bears only on the results obtained at the root node of the search tree for a selected subset of hard-to-solve instances and considers the following elements: (i) How long it takes to reach similar lower bounds with each memory scheme or, alternatively, which lower bound can be obtained within similar times with each memory scheme (depending on the instance, one of these two alternatives may be more convenient for the comparison); (ii) the number of rounds of separation of rank-1 cuts and the total number of active cuts the different memory schemes require to reach their respective lower bounds; and (iii) the time at the root node for the last exact pricing in both cases. This last information is very important, since the exact pricing is the most critical part of our BPC algorithm and must be kept tractable along the whole execution of the algorithm.

The results are given in Table 1, which provides the following statistics for each instance: **UB**, the computed upper bound; **RLB**, the root-node lower bound after adding robust cuts only; **RPT** the time (in seconds (s)) for the last exact pricing before starting the separation of rank-1 cuts; **RI**, the number of column-generation iterations (except those with the fast first pricing heuristic) to reach RLB; **RTT**, the total time to reach RLB; **LB**, the root-node lower bound reached after separating rank-1 cuts; **C**, the final number of active rank-1 cuts; **S**, the number of rounds of separation of rank-1 cuts; **PT**, the time of the last exact pricing performed; **I**, the total number of column generation iterations (except those with the fast pricing heuristic) to reach LB; **PAT**, the average pricing time over those iterations after the separation of rank-1 cuts starts; and **TT**, the total time to reach LB. Note that before the separation of rank-1 cuts start, all information provided is the same for both memory schemes. Obviously, when the separation starts, the runs diverge from each other (column labels ending with 1 refer to arc memory and with 2 to node memory).

For each of the six tested instances, we first solved its root node using a node memory and stopped the solution process when no more cuts could be generated (instance RC2-2-8) or when the first exact pricing took more than 1,000 s. In the latter case, we report in Table 1 the results obtained before adding the last round of nonrobust cuts. Second, we solved the instance using an arc memory. The solution process stopped when reaching a comparable lower bound for the first three instances and when reaching a similar computational time for the last three instances.

These results clearly show that the labeling algorithm can better handle the duals of the rank-1 cuts when using an arc memory instead of a node memory. Indeed, the average time for solving the pricing subproblem once these cuts have been added (columns **PAT**) is much shorter with an arc memory than with a node memory, despite there being a much larger number of active rank-1 cuts with the arc memory (columns **C**). Observe also that the last exact pricing times are much shorter with the arc memory, which shows that the BPC algorithm is in a better position to complete the solution process in less computational time. Consequently, even if a larger number of rank-1 cuts with a limited-arc-memory is needed to reach the same lower bound or to exceed it, the gains in time realized while solving the pricing subproblem largely compensate for the additional number of cut separation calls and column-generation iterations.

For the R208 Solomon instance, we were able to obtain a lower bound of 698.1 after adding additional arc-memory rank-1 cuts. This bound was reached in 13,828 s, and the last exact pricing required a still acceptable time of 125 s. Therefore, these cuts were decisive to solve this instance in 64,105 s. Indeed, using a node memory, it takes several days to solve, as suggested by the results given in Table 1. For instance, C2-2-3, a lower bound of 1,752.1, was obtained in 30,283 s with a final exact pricing time of 163 s after the addition of more arc-memory rank-1 cuts. Nevertheless, we cannot close this instance within four days of computation.

The results presented in this section show that using an arc memory for the rank-1 cuts can be useful for the instances that are very hard to solve. For the easier instances, that is not necessarily the case, as using a node memory may be more efficient. In the following, we use an arc memory for all instances to be consistent across our experiments.

6.2. Usefulness of the Elementary Cuts

We ran tests to assess the usefulness of the elementary cuts of the form (18). These tests were performed on a selected subset of 8 instances, namely, the hardest 200-customer instances of classes R1 and RC1 that exhibit the largest integrality gaps among the solved instances with 200 customers. The linear relaxation at the root node of each instance was solved twice. In the first run, we considered all nonrobust cuts, including the elementary cuts that were separated using the local search procedure proposed in Section 4.2. In the second run, no elementary cuts except the rank-1 cuts of the form (2) with $|S| = 3$ (previously used in the literature) were separated. This means that all rank-1 cuts (2) with $|S| = 4$ and $|S| = 5$ and multiplier vectors corresponding to elementary cuts were not considered.

The results of these tests are reported in Table 2. The labels common to this table and Table 1 have the

Table 1. Comparison Between Arc- and Node-Memory Schemes for Rank-1 Cuts

Ins	UB	RLB	RPT	RI	RTT	LB1	C1	S1	PT1	I1	PAT1	TT1	LB2	C2	S2	PT2	I2	PAT2	TT2
R208	701.1	691.1	7	70	1,622	697.5	139	10	70	226	25.1	9,621	697.3	100	7	796	189	254.0	63,864
C2-2-3	1,763.5	1,737.1	8	228	2,506	1,747.0	131	13	24	809	11.8	12,849	1,747.1	90	8	536	1,176	36.9	74,973
RC2-2-8	2,151.2	2,138.4	16	262	11,152	2,150.3	95	19	99	2,458	28.2	83,057	2,150.3	44	4	82	1,274	119.1	14,7585
C2-2-4	1,695.0	1,652.3	63	460	66,541	1,664.7	207	17	367	802	107.6	128,684	1,656.6	39	2	364	578	204.0	135,024
RC2-2-9	2,086.7	2,053.6	15	463	22,721	2,069.6	97	16	213	6,219	34.7	275,225	2,065.6	47	3	295	1,555	199.9	270,216
RC2-210	1,989.3	1,946.0	129	274	58,572	1,966.5	109	20	284	2,218	115.7	335,007	1,962.0	53	4	486	1,186	257.7	347,495

same meaning. Labels ending with 1 refer to the run including elementary cuts; those ending with 2 refer to the second run. The section dedicated to the results of the run with elementary cuts includes the following four additional columns. Label **EC** indicates the number of elementary cuts active in the last column-generation iteration, whereas label **ES** shows the average size of the set $S = C \cup \{i\}$ in these elementary cuts. Label **pLB1** gives the lower bound reached within the time **pTT1**. This time is less than the time given in column **TT2**.

From the results in columns **pLB1**, **pTT1**, **LB2**, and **TT2**, we observe that, for all these instances, the elementary cuts reach a similar or slightly better lower bound in less computational time. The columns **LB1** and **TT1** indicate that allowing more time can further improve the lower bound except for instance RC1-210. For all instances except RC1-210, the total number of nonrobust cuts required to reach the final lower bound is larger with elementary cuts. Nevertheless, the times of the last exact pricing (columns **PT**) are very similar with or without elementary cuts. These results show that the elementary cuts, which represent between 3% and 19% of the active nonrobust cuts, have a positive impact on the solution process for these difficult instances. Finally, column **ES** indicates that a large proportion of the active elementary cuts correspond to rank-1 cuts defined over sets S with $|S| \geq 6$.

6.3. Summary Results on the 100-Customer Solomon Instances

Table 3 summarizes the performance of the new BPC algorithm on the Solomon instances by comparing it against state-of-the-art algorithms. Column **NI** specifies the number of instances in each class. Columns **Opt** and **Time** indicate the number of instances solved to optimality and the average computational time in seconds (on the specified machine), computed only over the solved instances. Labels **JPSP08**, **DLH08**, and **BMR11** refer to the algorithms of Jepsen et al. (2008), Desaulniers et al. (2008), and Baldacci et al. (2011), respectively. Label **PCDU16** refers to the algorithm proposed in this paper.

These results indicate that the proposed BPC algorithm was able to solve the 56 Solomon instances to optimality in an average computational time of 1,375 s. All instances, except R208, were closed without any branching. Clearly, our algorithm outperforms the state-of-the-art algorithms in general. However, some C2 instances turned out to be difficult for our method, considering the average time reported by Baldacci et al. (2011) for this class. For example, for C204, the hardest instance in this set, our algorithm spent 1,648 s compared to 182 s for their algorithm. Even if all C2 instances were closed without separating any rank-1 cut, we noticed that the use of $NG = 10$ is not a good strategy for the instances C203 and , as the exact pricing needs to be performed several times because of the slow convergence of column generation. In fact, a much less constrained *ng*-route definition with $NG = 2$ allows us to reduce the average time for the C2 instances to 133 s. This suggests that a straightforward pricing based on *q*-route without 2-cycle elimination (i.e., only capacity constraint is considered) would be enough to handle this class efficiently. In contrast, our BPC algorithm solved hard instances much faster than the previous algorithms, as detailed in Table 5 in the online Appendix A.

Table 2. Improved Results Obtained Using Elementary Cuts

Ins	UB	RLB	RPT	RTT	pLB1	pTT1	LB1	C1	EC	ES	PT1	TT1	LB2	C2	PT2	TT2
R1-2-3	3,373.9	3,320.5	0.9	179	3,350.4	1,660	3,351.3	430	84	6.4	9.3	2,307	3,348.4	371	9.0	1,674
R1-2-4	3,047.6	3,000.0	1.4	274	3,031.6	1,728	3,032.5	486	85	6.7	8.7	2,135	3,030.1	414	9.0	1,784
RC1-2-2	3,221.6	3,176.7	0.5	141	3,203.3	596	3,204.2	394	66	6.4	2.7	825	3,202.3	337	2.0	607
RC1-2-3	3,001.4	2,949.5	1.1	282	2,979.2	1,717	2,981.3	492	84	6.4	11.1	2,560	2,979.3	476	7.5	1,774
RC1-2-7	3,177.8	3,112.3	0.6	179	3,151.2	4,872	3,151.6	552	69	6.4	38.6	5,439	3,150.4	480	34.2	4,976
RC1-2-8	3,060.0	3,017.4	0.8	226	3,043.9	1,322	3,044.9	426	28	6.9	9.7	1,716	3,043.9	391	6.2	1,355
RC1-2-9	3,074.8	3,000.8	0.9	253	3,035.4	5,324	3,037.2	545	49	6.4	39.5	11,300	3,035.4	521	37.4	8,508
RC1-210	2,990.5	2,932.5	1.1	261	2,964.1	3,747	2,964.1	477	18	6.6	15.1	3,747	2,963.8	487	15.4	4,144

Table 3. Comparison of Recent Algorithms on the 100-Customer Solomon Instances

		JPSP08		DLH08		BMR11		PCDU16	
Class	NI	Opt	Time	Opt	Time	Opt	Time	Opt	Time
C1	9	9	468	9	18	9	25	9	15
RC1	8	8	11,004	8	2,150	8	276	8	52
R1	12	12	27,412	12	2,327	12	251	12	31
C2	8	7	2,795	8	2,093	8	40	8	328
RC2	8	5	3,204	6	15,394	8	3,767	8	337
R2	11	4	35,292	8	63,068	10	28,680	11	6,432
Total	56	45		51		55		56	
Average			13,288		12,920		5,867		1,375
Processor		Opteron 2.6 GHz		Pentium 4 3 GHz		X7350 2.93 GHz		X-ES2637 3.5 GHz	

6.4. Summary Results on the 200-Customer Gehring and Homberger Instances

The main motivation behind this article is to assess the scalability of a BPC algorithm for solving larger VRPTW instances than the Solomon instances, which have now become all very tractable. We concentrated our tests on the Gehring and Homberger instances with 200 customers. Furthermore, to reduce the impact of the rank-1 cuts, a maximum of 410 rank-1 cuts are generated in each round of separation: at most 200 cuts with $|S| = 1$; at most 150 with $|S| = 3$; at most 30 with $|S| = 5$; and at most 30 additional elementary cuts.

Table 4 gives average results for the 200-customer instances (detailed results can be found in the online Appendix B). For each class, label **Opt** indicates the total number of instances solved (out of 10 instances). Labels **RGP**, **GP**, and **T** report, respectively, the average gap between the upper bound and the lower bound at the root node after adding only robust cuts, the average final root-node gap and the average time (in seconds) to solve the root node. These three statistics are computed over all instances (the *All-root* columns) and over the solved instances (the *Solved* columns). For the solved instances, we also report under labels **B** and **TT** the average number of nodes in the search tree (including the root node) and the average total time (in seconds), respectively. Finally, labels **Max** and **Min** specify the maximum and minimum gap at the root node for the unsolved instances.

As the table shows, of 30 instances in the classes C1, RC1, and R1, the BPC algorithm could solve a total of 28 instances to optimality. Although the gap at the root node for the hardest RC1 and R1 instances is typically high, route enumeration usually succeeds for nodes with a gap of 0.40% or less. Recall that after route enumeration, the solution of a node and its children is substantially accelerated for three main reasons: (i) only elementary routes in a pool are considered, (ii) stronger cuts (full-memory rank-1 cuts and clique cuts) are separated, and (iii) variable fixing is performed over the enumerated routes instead of arcs.

Among the instances solved in these three classes, only R1-2-3 could not be solved within the time limit using node-memory rank-1 cuts. The BPC algorithm failed to solve it because the variable fixing procedure that requires a complete forward labeling and a complete backward labeling stalled throughout the search tree. For example, at the root node, it took 2,042 s with 296 active limited-node-memory rank-1 cuts. When using limited-arc-memory rank-1 cuts, it required only 174 s with 371 active cuts. Therefore, the arc memory was crucial to solve this hard instance in an acceptable time. For the other 27 solved instances, the node-memory cuts yielded better results in general

Table 4. Summary Results on the 200-Customer Gehring and Homberger Instances

		All-root			Solved					Unsolved	
Class	Opt	RGP	GP	T	RGP	GP	T	B	TT	Max	Min
C1	10	0.10	0.04	88	0.10	0.04	88	1.60	133	—	—
RC1	8	1.68	0.57	4,674	1.56	0.45	3,889	46.00	38,940	1.27	0.82
R1	10	1.24	0.25	1,451	1.24	0.25	1,451	25.20	12,854	—	—
C2	8	0.83	0.24	20,552	0.54	0.00	4,096	1.00	4,096	1.75	0.64
RC2	7	1.29	0.24	123,251	0.77	0.05	26,362	1.57	27,375	1.13	0.28
R2	8	0.86	0.08	59,503	0.79	0.00	13,397	1.25	13,645	0.59	0.18
Total	51										
Average		1.00	0.24	34,920	0.82	0.13	7,274	13.03	15,195		

than the arc-memory cuts. Indeed, with the former cuts, the total computational time for these 27 instances is 239,157 seconds while it is 348,024 seconds when applying arc-memory cuts instead.

For classes C2, R2, and RC2, 23 of 30 instances were solved to optimality. For the closed instances, the gap after adding nonrobust cuts at the root node was typically very small, and the size of the resulting search tree was never larger than three nodes. It seems clear that the instances allowing very long routes (those in the C2, RC2, and R2 classes) typically exhibit small gaps if enough rank-1 cuts are separated. Nevertheless, these instances are, in general, difficult to solve because the labeling algorithms used for pricing the routes, for fixing variables, and for enumerating routes are less effective when routes can be very long and many dual variables from nonrobust cuts must be handled. Consequently, the limited-arc-memory rank-1 cuts that have a reduced negative impact on the computational time of the labeling algorithms constitute an improvement for solving these instances more efficiently.

7. Concluding Remarks

In this paper, we proposed two enhancements for the exact solution of the VRPTW by BPC. The first is a new form of the limited-memory rank-1 inequalities that reduce their negative impact on the pricing subproblem. The second is a new family of inequalities that dominate known inequalities. Through computational experiments, we showed that these two enhancements can be particularly effective for solving the most difficult instances. Overall, the proposed BPC algorithm produced the best known results for the 100-customer Solomon instances (closing all 56 instances) and for the 200-customer Gehring and Homberger instances (closing 51 of 60 instances). Future research will concentrate on improving the pricing mechanism and also on identifying new valid inequalities that are efficient and do not impact much the solution of the pricing subproblem.

Acknowledgments

The authors thank the associate editor and the two anonymous referees for their valuable comments. They also thank Thibaut Vidal for providing the upper bounds the algorithm uses. The authors gratefully acknowledge the funding support as reported earlier.

References

Balas E (1977) Some valid inequalities for the set partitioning problem. Hammer PL, Johnson EL, Korte BH, Nemhauser G, eds. *Studies in Integer Programming*, Annals of Discrete Mathematics (North-Holland, New York), 13–47.

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

Baldacci R, Mingozzi A, Roberti R (2012) Recent exact algorithms for solving the vehicle routing problem under capacity and time window constraints. *Eur. J. Oper. Res.* 218(1):1–6.

Barnhart C, Johnson EL, Nemhauser GL, Savelsbergh MWP, Vance PH (1998) Branch-and-price: Column generation for solving huge integer programs. *Oper. Res.* 46(3):316–329.

Contardo C, Martinelli R (2014) A new exact algorithm for the multi-depot vehicle routing problem under capacity and route length constraints. *Discrete Optim.* 12(1):129–146.

Cordeau J, Desaulniers G, Desrosiers J, Solomon M, Soumis F (2002) VRP with time windows. Toth P, Vigo D, eds. *The Vehicle Routing Problem* (SIAM Monographs on Discrete Mathematics and Applications, Philadelphia), 309–324.

Desaulniers G, Desrosiers J, Spoorendonk S (2011) Cutting planes for branch-and-price algorithms. *Networks* 58(4):301–310.

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 Sci.* 42(3): 387–404.

Desaulniers G, Madsen O, Ropke S (2014) The vehicle routing problem with time windows. Toth P, Vigo D, eds. *Vehicle Routing: Problems, Methods, and Applications*, MOS-SIAM Series on Optimization (SIAM, Philadelphia), 119–159.

Desrochers M, Desrosiers J, Solomon M (1992) A new optimization algorithm for the vehicle routing problem with time windows. *Oper. Res.* 40(2):342–354.

Dror M (1994) Note on the complexity of the shortest path models for column generation in VRPTW. *Oper. Res.* 42(5):977–978.

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(3):216–229.

Gehring H, Homberger J (2001) A parallel two-phase metaheuristic for routing problems with time windows. *Asia-Pacific J. Oper. Res.* 18(1):35–47.

Irnich S, Desaulniers G (2005) Shortest path problems with resource constraints. Desaulniers G, Desrosiers J, Solomon M, eds. *Column Generation*, Chap. 2 (Springer, New York), 33–65.

Irnich S, Desaulniers G, Desrosiers J, Hadjar A (2010) Path-reduced costs for eliminating arcs in routing and scheduling. *INFORMS J. Comput.* 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. *Oper. Res.* 56(2):497–511.

Kallehauge B, Larsen J, Madsen O (2006) Lagrangian duality applied to the vehicle routing problem with time windows. *Comput. Oper. Res.* 33(5):1464–1487.

Kohl N, Desrosiers J, Madsen OBG, Solomon MM, Soumis F (1999) 2-path cuts for the vehicle routing problem with time windows. *Transportation Sci.* 33(1):101–116.

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

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

Lysgaard J (2003) CVRPSEP: A package of separation routines for the capacitated vehicle routing problem. Accessed May 25, 2016, http://www.asb.dk/~lys.

Lysgaard J, Letchford A, Eglese R (2004) A new branch-and-cut algorithm for the capacitated vehicle routing problem. *Math. Programming* 100(2):423–445.

Martinelli R, Pecin D, Poggi M (2014) Efficient elementary and restricted non-elementary route pricing. *Eur. J. Oper. Res.* 239(1): 102–111.

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

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

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

Pessoa A, Uchoa E, Poggi de Aragão M (2009) A robust branch-cut-and-price algorithm for the heterogeneous fleet vehicle routing problem. *Networks* 54(4):167–177.

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. Golden B, Raghavan S, Wasil E, eds. *The Vehicle Routing Problem: Latest Advances and New Challenges* (Springer, New York), 397–419.

Poggi de Aragão M, Uchoa E (2003) Integer program reformulation for robust branch-and-cut-and-price. Wolsey L, ed. *Annals of Mathematical Programming in Rio, Búzios, Brazil*, 56–61.

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

Santos H, Toffolo T, Gomes R, Ribas S (2016) Integer programming techniques for the nurse rostering problem. *Ann. Oper. Res.* 239(1):225–251.

Sherali H, Adams W (1990) A hierarchy of relaxations between the continuous and convex hull representations for zero-one programming problems. *SIAM J. Discrete Math.* 3(3):411–430.

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

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(1):475–489.

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