Math. Prog. Comp. (2017) 9:61–100
DOI 10.1007/s12532-016-0108-8

FULL LENGTH PAPER

Improved branch-cut-and-price for capacitated vehicle routing

Diego Pecin[1] · **Artur Pessoa**[2] · **Marcus Poggi**[1] · **Eduardo Uchoa**[2]

Received: 17 September 2015 / Accepted: 22 April 2016 / Published online: 1 June 2016

Abstract The best performing exact algorithms for the capacitated vehicle routing problem developed in the last 10 years are based in the combination of cut and column generation. Some authors only used cuts expressed over the variables of the original formulation, in order to keep the pricing subproblem relatively easy. Other authors could reduce the duality gaps by also using a restricted number of cuts over the master LP variables, stopping when the pricing becomes prohibitively hard. A particularly effective family of such cuts are the subset row cuts. This work introduces a technique for greatly reducing the impact on the pricing of these cuts, thus allowing much more cuts to be added. The newly proposed branch-cut-and-price algorithm also incorporates and combines for the first time (often in an improved way) several elements found in previous works, like route enumeration and strong branching. All the instances used for benchmarking exact algorithms, with up to 199 customers, were solved to optimality. Moreover, some larger instances with up to 360 customers, only considered before by heuristic methods, were solved too.

Keywords Integer programming · Column generation · Cut separation · Algorithmic engineering

Mathematics Subject Classification 90C10 Integer programming · 90C39 Dynamic programming · 90C57 Polyhedral combinatorics, branch-and-bound, branch-and-cut

An extended abstract from this article was already published in [23].

✉ Eduardo Uchoa
uchoa@producao.uff.br

[1] Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro, Rio de Janeiro, RJ, Brazil

[2] Departamento de Engenharia de Produção, Universidade Federal Fluminense, Niterói, RJ, Brazil

1 Introduction

The capacitated vehicle routing problem (CVRP) was defined by Dantzig and Ramser [10] as follows. The input consists of a set of $n+1$ points, a *depot* and n *customers*; an $(n+1) \times (n+1)$ matrix $[c_{ij}]$ with the travel *costs* between every pair of points i and j; an n-dimensional *demand* vector $[d_i]$ giving the amount to be collected from customer i; and a vehicle *capacity* Q. A solution is a set of routes, starting and ending at the depot, that visit every customer exactly once. The only constraint on a route is that the sum of the demands of its customers does not exceed the vehicle capacity Q. The objective is to find a solution with minimum total cost. Many authors also assume that the *number of routes* is fixed to an additional input number K. The CVRP is a widely studied problem due to its own applications, since it can model adequately a significant number of real logistic systems. Furthermore, it plays a particularly important role on general vehicle routing research, for being the most basic and prototypical VRP variant.

Column generation has been used in exact algorithms for the vehicle routing problem with time windows (VRPTW) since Desrochers et al. [11]. This technique performed very well on tightly constrained instances (those with narrow time windows). As the CVRP can be regarded as the particular case of VRPTW where time windows are arbitrarily large, column generation was viewed as a non-promising approach for the problem. In fact, in the early 2000s, the best performing algorithms for the CVRP were branch-and-cut algorithms that separated quite complex families of cuts identified by polyhedral investigation (see Naddef and Rinaldi [22]). In spite of their sophistication, some instances from the literature with only 50 customers could not be solved to optimality [20]. At that moment, the branch-cut-and-price algorithm (BCP) algorithm in Fukasawa et al. [12] showed that the combination of cut and column generation could be much more effective than each of those techniques taken alone.

According to the classification proposed in Poggi and Uchoa [28], the BCP algorithm in [12] only uses *robust cuts*. A cut is said to be robust when the value of the dual variable associated to it can be translated into costs in the pricing subproblem. Therefore, the structure and the size of that subproblem remain unaltered, regardless of the number of robust cuts added. On the other hand, *non-robust cuts* are those that change the structure and/or the size of the pricing subproblem, each additional cut makes it harder. Robustness is a desirable property of a BCP. Even with good pricing heuristics, at least one call to the exact pricing is necessary to establish a valid dual bound. With the addition of non-robust cuts, there is a risk of having to solve to optimality an intractable subproblem. Nevertheless, since Jepsen et al. [17] and Baldacci et al. [2] it is known that some non-robust cuts can be effectively used, at least if their separation is restricted in order to avoid an excessive impact on the pricing. While the adjective *non-robust* focuses on their negative aspect; some authors call them *strong cuts*, focusing on their positive aspect, a greater potential for significantly reducing the integrality gaps.

1.1 Literature review

We briefly review the most recent works proposing exact algorithms for the CVRP, all of them are based on the combination of column and cut generation. A deeper review can be found in [29].

1. Fukasawa et al. [12] presented a BCP algorithm where the columns are associated to the q-routes without k-cycles [16]. The separated cuts are the same used in previous algorithms over the edge CVRP formulation. Those cuts are robust with respect to q-route pricing. If the column generation at the root node is found to be too slow, the algorithm may automatically switch to a branch-and-cut. This may be advantageous in some instances. All benchmark instances from the literature with up to 134 customers could be solved to optimality, an expressive improvement over previous methods.
2. Baldacci et al. [2] presented an algorithm based on column and cut generation. The columns are associated to elementary routes. Besides cuts for the edge formulation, strengthened capacity cuts and clique cuts are separated. Those later cuts are effective but non-robust. An important new idea is introduced. Instead of branching, the algorithm finishes in the root node (therefore, it is not a BCP) by enumerating all elementary routes with reduced cost smaller than the duality gap. A set-partitioning problem containing all those routes is then given to a MIP solver. The algorithm could solve almost all instances solved in [12], usually taking much less time. However, the exponential nature of some algorithmic elements, in particular the route enumeration, made it fail on some instances with many customers per route.
3. Pessoa et al. [24] presented some improvements over [12]. Cuts from an extended formulation with load indices were also separated. Those cuts do not change the complexity of the pricing of q-routes by dynamic programming. Additionally, the idea of performing elementary route enumeration and MIP solving to finish a node was borrowed from [2]. However, in order to avoid a premature failure when the root gap is too large, it was hybridized with traditional branching. The overall improvement was not sufficient for solving larger instances.
4. The algorithm by Baldacci et al. [3] improved upon [2]. It introduces the concept of ng-routes, a route relaxation that is more effective than the q-routes without k-cycles. Instead of clique cuts, subset row cuts ([17]) and weak subset row cuts, that have less impact on the pricing, are separated. The resulting algorithm is not only faster on average, it is much more stable than the algorithm in [2], being able to solve even some instances with many customers per route.
5. Contardo [8] introduced new twists on the use of non-robust cuts and on route enumeration. The columns are associated with q-routes without two-cycles, a relatively poor relaxation. The partial elementarity of the routes is enforced by non-robust strong degree cuts. Robust cuts from edge formulation and non-robust strengthened capacity cuts and subset row cuts are also separated. The enumeration of elementary routes is directed to a pool of columns. As soon as the duality gap is sufficiently small to produce a pool with reasonable size (a few million routes), the pricing starts to be performed by inspection. From this point, an aggressive

separation of non-robust cuts can be performed, leading to very small gaps. The reported computational results are very consistent. In particular, the hard instance M-n151-k12 (150 customers, 12 routes) was solved to optimality for the first time (in 6 hours), setting a new record.

6. Røpke [32] went back to robust BCP. Its linear relaxation differs from [12] only by the use of the more effective *ng*-routes. The power of the algorithm comes mainly from a sophisticated and aggressive strong branching, able to reduce significantly the average size of the enumeration trees. The overall results are comparable with the results in [3,8]. A long run of that algorithm (5.5 days) could also solve M-n151-k12.
7. Finally, Contardo and Martinelli [9] improved upon [8]. Instead of *q*-routes without two-cycles, *ng*-routes are used. Moreover, the performance of the dynamic programming pricing was enhanced by the use of the DSSR technique [4,31] and edge variables are fixed by reduced costs, using the procedure proposed in [15]. The computational results were also improved. For example, instance M-n151-k12 could be solved in less than 3 h.

1.2 New branch-cut-and-price algorithm

The branch-cut-and-price algorithm proposed in this work contains elements from all those previous algorithms, usually enhanced and combined with new elements. The features of that BCP are listed below:

- It uses limited memory subset row cuts (lm-SRCs), the most important original contribution introduced here. While the traditional SRCs are known to be effective, their practical use has been restricted by their large impact on the pricing. The lm-SRCs are a weakening of the SRCs. This weakening can be controlled and dynamically adjusted, making the lm-SRCs as effective in improving the lower bounds as the traditional SRCs, but still much less costly in the pricing. In fact, in many instances from the literature, including quite large ones, it is possible to separate lm-SRCs to obtain bounds as good as those that would be obtained by separating all SRCs with cardinality up to 5.
- The underlying formulation used in the BCP has extended arc-load variables. This allows a new and effective scheme for fixing of variables by Lagrangean bounds (superior to the fixing in [15]), with direct benefits on the pricing.
- The columns in the BCP are associated with *ng*-routes. The corresponding pricing subproblem is solved by a labeling algorithm that must also consider the dual variables of the lm-SRCs. Its implementation is quite critical for the overall BCP performance. After experiments with a number of alternatives, the best performance was obtained by a bidirectional search that differs a little from that proposed in [30] because the concatenation of the labels is not necessarily performed at the half of the capacity. Completion bounds (similar to those in [8]) are also used for eliminating labels. Anyway, the exact pricing algorithm is called just a few times per BCP node, most of the iterations use effective heuristics.
- In some instances, usually those with an average of more than 15 customers per route, column generation is prone to severe convergence problems. When this

situation is detected, the BCP automatically starts performing dual stabilization, as described in [25].
- Like in [24], the BCP hybridizes branching with route enumeration. Actually, it performs an aggressive hierarchical strong branching, with up to n candidates (partially) evaluated in the root node. The strong branching effort in each node depends on an estimate of the size of the subtree rooted in that node. The branching mechanism also keeps the history of candidate evaluations for helping on future decisions.
- As soon as the gap of a BCP node is sufficiently small, the elementary routes that can be part of the optimal solution are enumerated into a pool. From that point, since the pricing will be performed by inspection, all lm-SRCs may be immediately lifted to SRCs and additional non-robust cuts, including cliques, may be separated. On larger instances this may not be enough to reduce the number of routes to a size tractable by a MIP solver. In those cases, a new idea is used: perform ordinary BCP branching. The pricing will continue to be performed by pool inspection, in both child nodes. Nodes are only finished by the MIP solver when the number of remaining routes is quite small.
- The lm-SRCs are still non-robust. There are cases where several hundreds such cuts are being normally handled by the pricing algorithm, and then, the separation of a few dozen additional lm-SRCs makes this algorithm 100 times slower. In this situation the BCP performs a rollback, where the offending cuts are removed even if the lower bound of the node is decreased. This is a completely original feature, not appearing in previous works.

Overall, we believe that this is possibly the most sophisticated BCP algorithm for a particular problem ever implemented. The techniques introduced in this work, including the lm-SRCs, have the potential of being applied on many other problems, including several other VRP variants, parallel machine scheduling or network design.

This article is organized as follows. Section 2 presents the used formulations. Section 3 introduces the limited memory SRCs and their separation strategy. Section 4 describes the labeling algorithms used for pricing, for fixing variables by reduced costs and for enumerating routes. Section 5 presents the strong branching procedure, fully hybridized with the enumeration. Section 6 reports computational experiments. Finally, Sect. 7 contains a number of conclusions.

2 Formulations

This work departs from an extended formulation for the Asymmetrical CVRP presented in [24] (also in [13] for the unitary demand case). Let $G = (V, A)$ be a complete directed graph where $V = \{0, \ldots, n\}$ is the vertex set and A is the arc set. Vertices in $V_+ = \{1, \ldots, n\}$ correspond to the customers, whereas vertex 0 corresponds to the depot. A cost c_a is associated with each arc $a = (i, j) \in A$, symmetrical CVRP instances have symmetric costs, i.e., $c_{ij} = c_{ji}$. A positive integral demand d_i is associated to each customer i for $i \in V_+, d_0$ is defined as 0. Let Q denote the vehicle capacity. We assume that the number of routes is fixed to K. Let $G_Q = (V, A_Q)$ be a multigraph where A_Q contains arcs $(i, j)^q$, for all

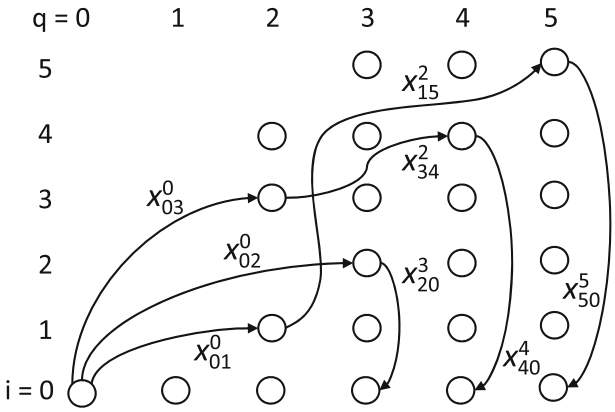

Fig. 1 Representation of a solution as a set of paths in $\mathcal{N}$

$i \in V_+$, $j \in V$ and for all $q = d_i, \ldots, Q$, plus arcs $(0, j)^0$, for all $j \in V_+$. For any set $S \subseteq V$, $\delta^-(S) = \{(i, j)^q \in A_Q : i \in V \setminus S, j \in S\}$, and $\delta^+(S)$ is defined in a similar way. For each $(i, j)^q \in A_Q$, a binary variable x^q_{ij} indicates that some vehicle goes from i to j carrying a load—the sum of the demands of vertex i and of its preceding vertices in the route—of exactly q units. The arc-load indexed formulation is:

$$\text{(ALF)} \quad \min \quad \sum_{a^q \in A_Q} c_a x^q_a \tag{1}$$

subject to

$$\sum_{a^0 \in \delta^+(\{0\})} x^0_a = K, \tag{2}$$

$$\sum_{a^q \in \delta^+(\{i\})} x^q_a = 1, \quad \forall i \in V_+, \tag{3}$$

$$\sum_{a^{q-d_i} \in \delta^-(\{i\})} x^{q-d_i}_a - \sum_{a^q \in \delta^+(\{i\})} x^q_a = 0, \quad \forall i \in V_+, q = d_i, \ldots, Q, \tag{4}$$

$$x^q_a \geq 0, \quad \forall a^q \in A_Q, \tag{5}$$

$$x \text{ integer}. \tag{6}$$

Equations (2) and (3) are depot and customer outdegree constraints. Balance equations (4) state that if an arc with index $q - d_i$ enters vertex i then an arc with index q must leave i. This formulation can be viewed as defining an acyclic network $\mathcal{N} = (V_Q, A_Q)$ with a set of nodes $V_Q = \{(i, q) : i \in V; q = d_i, \ldots, Q\}$. The set of arcs is also A_Q, but an arc $(i, j)^q \in A_Q$ is interpreted as going from (i, q) to $(j, q + d_i)$. Figure 1 gives an example of an integral solution (routes $(0 - 1 - 5 - 0)$, $(0 - 2 - 0)$, and $(0 - 3 - 4 - 0)$) depicted as a set of paths in such a network, where $n = Q = 5$, $K = 3$, $d_1 = d_3 = d_4 = 2$, and $d_2 = d_5 = 3$.

A q-route is a walk that starts at the depot, traverses a sequence of customers with total demand at most Q, and returns to the depot [7]. The q-routes are not necessarily elementary, but each time a customer is revisited its demand in counted again. Let Ω be the set of all q-routes. For each $r \in \Omega$ define a non-negative variable λ_r and binary coefficients a_{rq}^{ij}, for each $(i, j)^q \in A_Q$, indicating whether (i, j) is traversed with load q in route r. Equations (4) in ALF can be replaced by:

$$\sum_{r \in \Omega} a_{rq}^{ij} \lambda_r = x_{ij}^q, \forall (i, j)^q \in A_Q. \tag{7}$$

Substituting the x variables and relaxing the integrality, the Dantzig–Wolfe master LP is written as:

$$\text{(DWM)} \quad \min \sum_{r \in \Omega} \left(\sum_{(i,j)^q \in A_Q} a_{rq}^{ij} c_{ij} \right) \lambda_r \tag{8}$$

subject to

$$\sum_{r \in \Omega} \left(\sum_{(i,j)^q \in \delta^+(\{0\})} a_{rq}^{ij} \right) \lambda_r = K, \tag{9}$$

$$\sum_{r \in \Omega} \left(\sum_{(i,j)^q \in \delta^+(\{i\})} a_{rq}^{ij} \right) \lambda_r = 1, \quad \forall i \in V_+, \tag{10}$$

$$\lambda_r \geq 0 \quad \forall r \in \Omega. \tag{11}$$

A generic constraint l of format $\sum_{(i,j)^q \in A_Q} \alpha_{ij}^{lq} x_{ij}^q \geq b_l$ can also be included in the DWM, using the variable substitution (7), as $\sum_{r \in \Omega} (\sum_{(i,j)^q \in A_Q} \alpha_{ij}^{lq} a_{rq}^{ij}) \lambda_r \geq b_l$. Suppose that, at a given instant, there are n_R constraints over the x variables (including (9) and (10)) in the DWM. Equality (9) is numbered as 0, equalities (10) are numbered from 1 to n, according to the $i \in V_+$, other constraints are numbered from $n + 1$ to n_R. The dual variable of the constraint with number l is denoted by π_l. The reduced cost of an arc $(i, j)^q$ is defined as:

$$\bar{c}_{ij}^q = c_{ij} - \sum_{l=0}^{n_R - 1} \alpha_{ij}^{lq} \pi_l. \tag{12}$$

The pricing subproblem for solving the DWM consists in finding a shortest path in $\mathcal{N}$ from node $(0, 0)$ to nodes $(0, q)$, $1 \leq q \leq Q$, with respect to the arc reduced costs $\bar{c}_{ij}^q$. This can be done in $O(n^2 Q)$ time.

A significantly stronger linear relaxation would be obtained if Ω was redefined as the set of elementary routes. On the other hand, the pricing subproblem would become strongly NP-hard. While carefully designed labeling algorithms are now capable of pricing elementary routes on most instances from the literature with up to 199 customers [21], this is still too costly. A possible alternative is pricing q-routes without k-cycles [16], a relaxation of the elementary routes that allows multiple visits to a customer, on the condition that at least k other customers are visited between successive

visits. While pricing q-routes without three-cycles is not much more costly than pricing ordinary q-routes [12], using values of k larger than 4 can make the algorithm too slow. A more recent alternative for imposing partial elementarity, used in this work, are the ng-routes [3]. For each customer $i \in V_+$, let $NG(i) \subseteq V_+$ be the ng-set of i, defining its neighborhood. This may stand for the $|NG(i)|$ (this cardinality is decided *a priori*) closest customers and includes i itself. An ng-route allows multiple visits to a customer i, on the condition that at least one costumer j such that $i \notin NG(j)$ is visited between successive visits. Extensive experiments performed in [21] showed that the use of ng-sets with size 8 already yields lower bounds comparable with the use of q-routes without 5-cycles, but spending significantly less time. From now on, Ω is redefined to be a set of ng-routes.

3 Cuts

Even if Ω only contains elementary routes, the bounds given by (8)–(11) are *not* good enough to be the basis of efficient exact algorithms (gaps between 1 and 4 % are typical in the instances from the literature). For this purpose, the formulation must be reinforced with additional cuts. Cuts for the undirected edge formulation can be included in the DWM by using the transformation $x_{ij} = \sum_{(i,j)^q \in A_Q} x^q_{ij} + \sum_{(j,i)^q \in A_Q} x^q_{ji}$. In this work, rounded capacity cuts and strengthened combs are separated by the CVRPSEP package [19]. All those cuts are robust, the effect of their dual variables is captured in the arc-load reduced costs (12).

[17] introduced a family of inequalities defined over the route variables. Let $a^r_i = \sum_{(i,j)^q \in A_Q} a^{ij}_{rq}$ be the number of times that vertex i appears in route r. Given a base set $C \subseteq V_+$ and a multiplier $p, 0 < p < 1$, the following (C, p)-Subset Row Cut (SRC)

$$\sum_{r \in \Omega} \left\lfloor p \sum_{i \in C} a^r_i \right\rfloor \lambda_r \leq \lfloor p|C| \rfloor \tag{13}$$

is valid, since it can be obtained by a Chvátal-Gomory rounding of the corresponding constraints in (10). For each integer $d, 1 \leq d \leq n$, define a non-negative integer variable y^d_C as the sum of all variables λ_r such that $\sum_{i \in C} a^r_i = d$. Variables with $d > |C|$ can only be non-zero if Ω contains non-elementary routes. The interesting SRCs, for sets C with cardinality up to 5, are the following:

- The cuts where $|C| = 3$ and $p = 1/2$ are called 3-subset row cuts (3SRCs) and can be expressed as $y^2_C + y^3_C + 2y^4_C + 2y^5_C + \ldots \leq 1$. Although they are very effective in improving the lower bounds, only a relatively small number of those cuts could be separated in [3,8], and [9], in order to keep the pricing tractable. The first authors also used the Weak 3SRCs, a weakening of the 3SRCs where only the variables corresponding to routes that use an edge (i, j) such that $i, j \in C$ have coefficient 1.
- Taking $|C| = 1$ and $p = 1/2$, the 1-subset row cuts (1SRCs) $y^2_C + y^3_C + 2y^4_C + \ldots \leq 0$ are obtained. They are equivalent to the strong degree cuts $y^1_C \geq 1$ introduced in

[8], in the sense that both families forbid cycles over a vertex i ($C = \{i\}$). Contardo also defined the weaker k-cycle elimination cuts, that only forbid cycles over i of size k or less. Of course, these cuts can only be useful when the Ω set contains non-elementary routes.
- The cuts where $|C| = 4$ and $p = 2/3$ are 4SRCs, expressed as $y_C^2 + 2y_C^3 + 2y_C^4 + 3y_C^5 + 4y_C^6 + \ldots \leq 2$.
- There are two interesting families of cuts with $|C| = 5$. Those with $p = 1/3$ will be called 5,1SRCs, $y_C^3 + y_C^4 + y_C^5 + 2y_C^6 \ldots \leq 1$; whereas those with $p = 1/2$ are 5,2SRCs, having the format $y_C^2 + y_C^3 + 2y_C^4 + 2y_C^5 + 3y_C^6 \ldots \leq 2$. The latter family was already used in [8], and [9].

The newly proposed limited memory subset row cuts (lm-SRCs) correspond to a generalization of the SRCs. Each such cut is defined by a triplet (C, M, p), where M is the memory set ($C \subseteq M \subseteq V_+$). Remark that each lm-SRC has its own memory set. An lm-SRC is written as:

$$\sum_{r \in \Omega} \alpha(C, M, p, r)\lambda_r \leq \lfloor p|C| \rfloor , \tag{14}$$

where the coefficients α are a function of C, M, p and r computed by algorithm 1.

Algorithm 1 Procedure that calculates the coefficient of a route r in a lm-SRC

```
1: function α(C, M, p, r)
2: coeff ← 0, state ← 0
3: for every vertex i in route r (in order) do
4:     if i ∉ M then
5:         state ← 0
6:     else if i ∈ C then
7:         state ← state + p
8:         if state ≥ 1 then
9:             coeff ← coeff + 1, state ← state − 1
10: return coeff
```

Variable $coeff$ stores the coefficient to be returned. Each time a vertex in C is visited, variable $state$ is increased by p. When $state$ becomes larger or equal to 1, its value is reduced by 1 unit and $coeff$ is incremented. When $M = V_+$, the Function α will return $\lfloor p \sum_{i \in C} a_i^r \rfloor$ and the lm-SRC will be identical to an SRC. On the other hand, when M is strictly contained in V_+, the lm-SRC may be a weakening of its corresponding SRC. This happens because every time the route r leaves M, the variable $state$ is reset to zero, potentially decreasing the returned coefficient. Function α indicates how the lm-SRCs should be taken into account in the labeling algorithms used in the pricing. In fact, that procedural function is executed along the algorithm. Each label should have an additional dimension for each lm-SRC, storing their states in the corresponding partial paths. However, the coefficients do not need to be stored in the labels. Instead, whenever a label extension causes the increment of the coefficient of an lm-SRC, according to function α, the value of its dual variable is immediately subtracted from the reduced cost of the new label. We remark that the number of

possible states of an lm-SRC depends on its p. For example, for the frequent case where $p = 1/2$, the state can be only 0 or $1/2$. Therefore, it can be represented by a single bit.

The potential advantage of the lm-SRCs over classical SRCs is their much reduced impact on the labeling algorithm used in the pricing, when $|M| << |V_+|$. The reasons for that reduction will be explained in Sect. 4.1.1. In order to obtain small memory sets, we propose the following separation strategy for the lm-SRCs. First, identify a violated (C, p)-SRC. Then, function calculate M is used to obtain a minimal memory such that the lm-(C, M, p)-SRC has the same violation. For example, suppose that,

Algorithm 2 Procedure to calculate the memory set of a separated lm-SRC

```
1: function Calculate M(C, p, λ)
2: M ← C
3: for each route r such that λ_r > 0 and ⌊p Σ_{i∈C} a_i^r⌋ > 0 do
4:     state ← 0, Aux ← ∅
5:     for every vertex i in route r (in order) do
6:         if i ∈ C then
7:             state ← state + p
8:             if state ≥ 1 then
9:                 M ← M ∪ Aux, Aux ← ∅, state ← state − 1
10:        else if state > 0 then
11:            Aux ← Aux ∪ {i}
12: return M
```

in a given fractional solution, the paths that visit $C = \{1, 2, 3\}$ at least twice are: $r_1 = (0-1-4-5-3-6-2-7-1-0)$ with $\lambda_{r_1} = 0.2$, $r_2 = (0-7-2-8-3-0)$ with $\lambda_{r_2} = 0.3$, and $r_3 = (0-5-3-4-1-7-9-2-0)$ with value $\lambda_{r_3} = 0.4$. The $(C, 1/2)$ − SRC $2\lambda_{r_1} + \lambda_{r_2} + \lambda_{r_3} \leq 1$ has a violation of 0.1. The minimal set M obtained that yields a lm-$(C, M, 1/2)$-SRC with the same violation is $M = C \cup \{4, 5\} \cup \{7\} \cup \{8\} \cup \{4\}$.

4 Labeling algorithms

In this section we present the algorithms used for pricing ng-routes, for fixing arc-load variables by reduced costs, and for enumerating elementary routes. Those algorithms should take into account the following points: (i) the reduced cost of an arc, defined as (12), depends on its load q; (ii) some variables x_a^q may be already fixed to zero; and (iii) non-robust lm-SRCs may be present.

4.1 Pricing

4.1.1 Forward labeling

The forward dynamic programming labeling algorithm for the pricing problem represents an ng-feasible partial path $P = (0, \ldots, i), i \in V$, as a *label* $L(P) = (\bar{c}(P), v(P) = i, q(P), \Pi(P), S(P), pred(P))$ storing its reduced cost, end vertex,

load, set of vertices forbidden as immediate extensions due to ng-sets, vector of states corresponding to the n_S lm-SRCs with non-zero dual variables in the current Master LP solution, and a pointer to its predecessor label. Each $(i, q) \in V_Q$ defines a *bucket* $F(i, q)$. A label $L(P)$ is stored in bucket $F(v(P), q(P))$. A label $L(P_1)$ dominates a label $L(P_2)$ if every feasible completion of P_2 yields a route with reduced cost not smaller than the feasible route obtained by applying the same completion into P_1. The following four conditions, together, are sufficient to ensure such a domination:

$$\text{(i) } v(P_1) = v(P_2), \quad \text{(ii) } q(P_1) = q(P_2), \quad \text{(iii) } \Pi(P_1) \subseteq \Pi(P_2), \text{ and}$$
$$\text{(iv) } \bar{c}(P_1) \leq \bar{c}(P_2) + \sum_{1 \leq s \leq n_S : S(P_1)[s] > S(P_2)[s]} \sigma_s,$$

where $\sigma_s < 0$ is the dual variable associated to lm-SRC s. Remark that the reduced costs depending on q or the fixing of some x_a^q variables to zero prevent (ii) to be strengthened to $q(P_1) \leq q(P_2)$. This happens because, if $q(P_1) \neq q(P_2)$, the reduced cost of a completion for P_2 may differ from the reduced cost of same completion for P_1. In fact, due to the fixing, a feasible completion for P_2 may be infeasible for P_1. Note also that the second term in the right-hand side of (iv) is an upper bound on what a completion of P_2 can gain over the same completion of P_1, by avoiding the penalizations of revisiting the lm-SRCs s in which $S(P_1)[s] > S(P_2)[s]$. Only non-dominated labels are kept in the buckets. To accelerate the checking for dominated labels, it is convenient to keep labels of the same bucket ordered by reduced cost. The base set, multiplier and memory set of a lm-SRC s are denoted by $C(s)$, $p(s)$, and $M(s)$, respectively. Consider $NG(0)$ as $\{0\}$. Algorithm 3 presents the pseudocode of the forward labeling procedure. In the end of the algorithm, each non-empty bucket

Algorithm 3 Procedure forward labeling

1: $F(i, q) \leftarrow \emptyset, \forall (i, q) \in V_Q$
2: $F(0, 0) \leftarrow \{(0, 0, 0, \emptyset, \mathbf{0}, nil)\}$
3: **for** $q = 0$ to Q **do**
4: **for all** i such that $(i, q) \in V_Q$ **do** ▷ Process buckets F with load q
5: **for all** j such that $(i, j)^q \in A_Q$ and $x_{i,j}^q$ is not fixed to 0 **do**
6: **for all** $L_1 = (\bar{c}_1, i, q, \Pi_1, S_1, \_) \in F(i, q)$ **do**
7: **if** $j \notin \Pi_1$ **then**
8: $\bar{c}_2 \leftarrow \bar{c}_1 + \bar{c}_{ij}^q$, $S_2 \leftarrow S_1$
9: **for** $s = 1$ to n_S **do**
10: **if** $j \notin M(s)$ **then** $S_2[s] \leftarrow 0$
11: **else if** $j \in C(s)$ **then**
12: $S_2[s] \leftarrow S_2[s] + p(s)$
13: **if** $S_2[s] \geq 1$ **then** $\bar{c}_2 \leftarrow \bar{c}_2 - \sigma_s$, $S_2[s] \leftarrow S_2[s] - 1$
14: $L_2 \leftarrow (\bar{c}_2, j, q + d_j, (\Pi_1 \cap NG(j)) \cup \{j\}, S_2,$ pointer to $L_1)$
15: $insertLabel \leftarrow$ **true**
16: **for all** $\mathcal{L} \in F(j, q + d_j)$ **do**
17: **if** L_2 dominates $\mathcal{L}$ **then** delete $\mathcal{L}$
18: **else if** $\mathcal{L}$ dominates L_2 **then** $insertLabel \leftarrow$ **false**, **break**
19: **if** $insertLabel$ **then**
20: $F(j, q + d_j) \leftarrow F(j, q + d_j) \cup \{L_2\}$

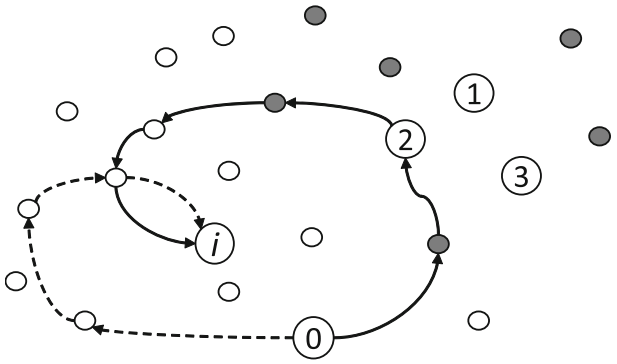

Fig. 2 Example illustrating the performance gain in the pricing when using lm-SRCs

$F(0, q)$, $1 \leq q \leq Q$, will contain only one label, representing the minimum reduced cost route with load q.

Now, it is possible to explain why the lm-SRCs have a reduced impact in the pricing when their memory sets are small. There are $O(nQ)$ buckets in the labeling algorithm. In a rough analysis, the time spent processing each bucket grows quadratically with the average number of non-dominated labels in it and linearly with n_S.

- A first observation is that the state of an lm-SRC s needs only to be explicitly present in labels corresponding to partial paths ending in $M(s)$. For the remaining labels, this state is zero by definition, which means that σ_s will not play any role in the processing of their buckets. In other words, only the vertices in $M(s)$ need to know about the existence of that cut. This allows the acceleration of the algorithm by a factor of $\theta(n/M_{avg})$, where M_{avg} is the average size of the memory sets.
- However, the crucial point for the improved algorithm efficiency is related to the dominance. If there are no SRCs, the maximum number of non-dominated labels in a bucket $F(i, q)$ is bounded by $2^{|NG(i)|-1}$, as follows from dominance conditions (iii) and (iv). If the cardinality of the ng-sets is small (we used 8 in this work), the pricing is guaranteed to be reasonably fast (unless Q is very large). However, if a traditional SRC is added, its dual variable may make condition (iv) *weaker in all buckets*. As other SRCs are separated, this may quickly result in an exponential proliferation of non-dominated labels. In practice, this severely limits the number of SRCs that can be used. In contrast, a lm-SRC s with a small memory has much less impact because *it can only weaken the dominance in the buckets of* $M(s)$. In practice, many more lm-SRCs can be separated before the exponential proliferation of labels is observed. This is exemplified in Fig. 2. Let P_1 be the solid path and P_2 the dashed one, both paths ending in vertex i and having load q. A 3SRC with base set $C = \{1, 2, 3\}$ may prevent $L(P_1)$ from dominating $L(P_2)$, even though no good completion of P_2 visits C. On the other hand, a lm-3SRC over the same C, having the memory set represented in the figure by filled circles, would not interfere with that dominance.

4.1.2 Bidirectional labeling

The labeling algorithm for the pricing can also be performed backwards. In that case, the labels represent ng-feasible partial paths $P = (i, \ldots, 0)$, $i \in V_+$. The initializing labels are put in buckets $B(0, q)$, $1 \leq q \leq Q$, and the algorithm proceeds in a reversed way, until the label corresponding to the route with minimum reduced cost is found in bucket $B(0, 0)$, as shown in Algorithm 4.

Algorithm 4 Procedure backward labeling

$B(i, q) \leftarrow \emptyset, \forall (i, q) \in V_Q$
for $q = Q$ to 1 **do**
  $B(0, q) \leftarrow \{(0, 0, q, \emptyset, \mathbf{0}, nil)\}$ ▷ Process buckets B with load q
  for all i such that $(i, q) \in V_Q$ **do**
    for all j such that $(i, j)^{q-d_j} \in A_Q$ and $x_{i,j}^{q-d_j}$ is not fixed to 0 **do**
      for all $L_1 = (\bar{c}_1, j, q, \Pi_1, S_1, \_) \in B(j, q)$ **do**
        if $i \notin \Pi_1$ **then**
          $\bar{c}_2 \leftarrow \bar{c}_1 + \bar{c}_{ij}^{q-d_j}$, $S_2 \leftarrow S_1$
          for $s = 1$ to n_S **do**
            if $i \notin M(s)$ **then** $S_2[s] \leftarrow 0$
            else if $i \in C(s)$ **then**
              $S_2[s] \leftarrow S_2[s] + p(s)$
              if $S_2[s] \geq 1$ **then** $\bar{c}_2 \leftarrow \bar{c}_2 - \sigma_s$, $S_2[s] \leftarrow S_2[s] - 1$
          $L_2 \leftarrow (\bar{c}_2, i, q - d_j, (\Pi_1 \cap NG(i)) \cup \{i\}, S_2,$ pointer to $L_1)$
          $insertLabel \leftarrow$ **true**
          for all $\mathcal{L} \in B(i, q - d_j)$ **do**
            if L_2 dominates $\mathcal{L}$ **then** delete $\mathcal{L}$
            else if $\mathcal{L}$ dominates L_2 **then** $insertLabel \leftarrow$ **false**, **break**
          if $insertLabel$ **then**
            $B(i, q - d_j) \leftarrow B(i, q - d_j) \cup \{L_2\}$

The forward and backward variants of the labeling are equivalent in terms of computational cost. However, as pointed out in [30], when forward labeling is used, most of the computational effort is spent in buckets with larger values of q, close to Q. This happens by combinatorial reasons, there are many more possible paths converging into a bucket $F(i, q)$ if q is larger. In a similar way, when backward labeling is used, most of the computational effort is spent in buckets with small values of q. Therefore, it is often advantageous to perform bidirectional search: use the forward labeling for filling the buckets $F(i, q)$ with $q \leq Q/2$ and backward labeling for filling the buckets $B(i, q)$ with $q > Q/2$. The minimum reduced cost paths can be obtained by an additional concatenation step. After implementing this bidirectional algorithm, we realized that the number of labels in the backward part was consistently larger (3–10 times more is typical) than in the forward part. This happens because the backward labeling has more starting labels. Therefore, the algorithm performance can be improved by better balancing both parts. This means that the concatenation will occur at a value (dynamically determined) larger than $Q/2$.

Let $nFL(nBL)$ denote the current number of non-dominated forward (backward) labels generated so far. Algorithm 5 presents the pseudocode of the bidirectional

labeling procedure. The algorithm starts by running the forward labeling and backward labeling in an alternated way. If nFL is smaller than nBL, then the buckets F with load qf are processed and qf is incremented. Otherwise, the buckets B with load qb are processed and qb is decremented. The process ends when $qf = qb - 1$. The minimum reduced cost route with load $1 \leq q \leq qf$ is obtained from bucket $F(0, q)$, which contains at most one label. However, routes with load $q > qf$ must be obtained from the concatenation of forward and backward labels, which is a potentially costly operation. We use the fact that the labels are ordered by reduced cost inside the buckets. Therefore, many concatenations that would not yield negative reduced cost routes are quickly discarded. This is possible because the lm-SRCs only penalize the concatenation of labels, since their duals are negative. Procedure *Save* saves pointers to pairs of labels corresponding to the routes with negative reduced cost, for subsequent use in the column generation.

Algorithm 5 Procedure bidirectional labeling

$F(i, q) \leftarrow \emptyset, \forall (i, q) \in V_Q$
$F(0, 0) \leftarrow \{(0, 0, 0, \emptyset, \mathbf{0}, nil)\}, nFL \leftarrow 1$
$B(i, q) \leftarrow \emptyset, \forall (i, q) \in V_Q, nBL \leftarrow 0$
$qf \leftarrow 0, qb \leftarrow Q$
while $qb - qf > 1$ **do**
  if $nFL < nBL$ **then**
    Process buckets F with load qf updating nFL, $qf \leftarrow qf + 1$
  else
    Process buckets B with load qb updating nBL, $qb \leftarrow qb - 1$
for all $(i, j)^q \in A_Q$ such that $x^q_{i,j}$ is not fixed to 0 and $q \leq qf$ and $q + d_j \geq qb$ **do**
  for all $L_1 = (\bar{c}_1, i, q, \Pi_1, S_1, \_) \in F(i, q)$ in increasing reduced cost order **do**
    for all $L_2 = (\bar{c}_2, j, q + d_i, \Pi_2, S_2, \_) \in B(j, q + d_j)$ in increasing reduced cost order **do**
      $\bar{c}_r \leftarrow \bar{c}_1 + \bar{c}^q_{ij} + \bar{c}_2$
      if $\bar{c}_r \geq 0$ **then break**
      if $\Pi_1 \cap \Pi_2 = \emptyset$ **then**
        for $s := 1, \ldots, n_S$ **do**
          if $S_1[s] + S_2[s] \geq 1$ **then**
            $\bar{c}_r \leftarrow \bar{c}_r - \sigma_s$
        if $\bar{c}_r \leq 0$ **then**
          Save(pointer to L_1, pointer to L_2)

4.1.3 Completion bounds

Besides dominance, there is a second mechanism for eliminating labels along the forward and backward labeling algorithms. If it can be proved that a partial path P can not be completed into a route with negative reduced cost, then $L(P)$ can be removed from its bucket. Of course, it is not possible to find the exact cost of the best completion for every P without running the full original algorithms. Instead, one needs to devise *completion bounds*, i.e., lower bounds on the cost of the completions; they should be obtained in a relatively fast way. For example, completion bounds can be obtained by runs of the labeling algorithms that only consider, for each label L, the dimensions of $S(L)$ that correspond to the n'_S active lm-SRCs out of the n_S existing ones. Since

the effect of a lm-SRC in the pricing is penalizing the reduced cost of some routes, this corresponds to a relaxation of the original problem. However, as proposed in [8], better completion bounds can be obtained by incorporating part of the effect of the remaining $n_S - n'_S$ SRCs into the reduced costs as follows:

- For a 3-SRC or a 5,2SRC s, the value $\sigma_s/2$ can be subtracted from the reduced cost of the arcs inside $C(s)$.
- For a 4-SRC s, the value $2\sigma_s/3$ can be subtracted from the reduced cost of the arcs inside $C(s)$.

However, it is not possible to transfer part of the dual variables corresponding to 1-SRCs or to 5,1SRCs into arc reduced costs. In the first case, because there are no arcs inside C. In the second case, because a route that uses just one arc inside the C set of an 5,1SRC may have coefficient zero in the cut.

The completion bounds for the forward labeling are obtained by first running the relaxation defined above of the backward labeling. Let $\widehat{F}(i, q)$ be the minimum reduced cost of a label in a bucket $B'(i, q)$, $(i, q) \in V_Q$, filled by that relaxation. Then, the extension of label $L(P) = (\bar{c}(P), i, q, \_, \_, \_)$ to a customer j in Algorithm 3 is avoided if the following condition holds:

$$\bar{c}(P) + \bar{c}^q_{ij} + \widehat{F}(j, q + d_j) \geq 0. \tag{15}$$

Analogously, the backward Algorithm 4 uses completion bounds obtained by running a relaxed run of the forward labeling. Therefore, Algorithm 5 uses both forward and backward completion bounds.

Table 10 in the appendix shows detailed results over a number of hard instances of five variants of the pricing: 1—forward labeling (Algorithm 3); 2—bidirectional labeling with concatenation at $Q/2$; 3—bidirectional labeling with dynamic determination of the concatenation point (Algorithm 5); 4—the later algorithm with mild use of completion bounds (n'_S is small); 5—the same algorithm with aggressive use of completion bounds (n'_S is larger).

4.1.4 Non-robustness control

Even with all the previously described enhancements, the addition of too many lm-SRCs will still cause an exponential explosion of the number of labels in the pricing algorithms. Worse, it is not possible to predict when the explosion will occur. We have seen examples of runs where several hundreds such cuts are being normally handled by the pricing algorithm and then, at some node deep in the tree, the separation of a few dozen additional lm-SRCs makes this algorithm 100 times or even 1000 times slower. In some cases the BCP crashed due to memory overflow.

The first strategy tried for avoiding such failures was setting a conservative limit on n_S and stopping the separation of lm-SRCs after it is reached. A much better strategy is to handle the lm-SRCs dynamically. In the beginning of the root node, when no lm-SRCs were added, the pricing still has a pseudo-polynomial complexity (assuming that the size of the ng-sets is fixed). Those robust runs of the pricing are used to establish a baseline BL on the number of labels. The algorithm proceeds by

separating rounds of lm-SRCs. The number of labels in the pricing is likely to increase, values up to $5BL$ are always acceptable, larger values may be tolerated if the rate of lower bound improvement is still good. This mechanism determines, for each node, when separation will stop and enumeration/branching will be called. However, if the number of labels in a call of the pricing exceeds $50BL$, the BCP concludes that an exponential explosion is occurring. The pricing is aborted and the node is rolled back to its previous state before the last round of cuts. This means that the lm-SRCs added in that round, even if they are active, are removed.

4.1.5 Heuristic pricing

Even with the use of the accelerating techniques mentioned in this section, the exact pricing algorithm can still be quite time-consuming. Therefore, the column generation can be accelerated by also using faster pricing heuristics. In fact, the exact pricing may be only called when the pricing heuristics can not find routes with negative reduced cost. Effective and simple such heuristics can be obtained by modifying the label setting algorithms. The *bucket pruning* heuristic (for example, see [12]) used in this work consists of storing only a fixed (small) number of labels per bucket. This means that many non-dominated labels, those that are less likely to lead to optimal solutions, may be dropped.

Algorithm 6 Procedure variable fixing

Run forward labeling
Run backward labeling
for all $(i, j)^q \in A_Q$ such that $x^q_{i,j}$ is not fixed to 0 **do**
    $Fix \leftarrow$ **true**
    for all $L_1 = (\bar{c}_1, i, q, \Pi_1, S_1, \_) \in F(i, q)$ in RC order **do**
        for all $L_2 = (\bar{c}_2, j, q + d_j, \Pi_2, S_2, \_) \in B(j, q + d_j)$ in RC order **do**
            $\bar{c}_r \leftarrow \bar{c}_1 + \bar{c}^q_{ij} + \bar{c}_2$
            if $\bar{c}_r > gap$ **then break**
            if $\Pi_1 \cap \Pi_2 = \emptyset$ **then**
                for $s = 1, \ldots, n_S$ **do**
                    if $S_1[s] + S_2[s] \geq 1$ **then**
                        $\bar{c}_r \leftarrow \bar{c}_r - \sigma_s$
                if $\bar{c}_r \leq gap$ **then** $Fix \leftarrow$ **false**, **break**
        if not Fix **then break**
    if Fix **then** Fix variable $x^q_{i,j}$ to 0

4.2 Variable fixing by reduced costs

The labeling algorithms are also employed in a key part of the BCP algorithm: the elimination of variables by reduced costs. A full separated run of both forward and backward labeling should be performed. The minimum reduced cost of a route passing by an arc $(i, j)^q \in A_Q$, denoted by $\bar{C}^q_{ij}$, can be obtained by concatenating the labels in $F(i, q)$ from the forward run with the labels in $B(j, q + d_j)$ from the backward run. If $\bar{C}^q_{ij}$ is larger than the gap of the Lagrangean bound associated to the current dual

solution with respect to the best known integer solution (see [26]), then $x_{i,j}^{q}$ can be fixed to zero. The pseudocode of our variable fixing is presented by Algorithm 6. It is quite similar to that shown in Algorithm 5, the bidirectional labeling procedure. Note that the concatenation here also relies on the fact that labels are ordered by reduced cost inside buckets.

A similar fixing procedure was also proposed in [15], but it is weaker because it only removes an arc $(i, j) \in A$, if a single particular dual solution allows removing $(i, j)^q$ for all values of q. On the other hand, as our BCP already works on the arc-load formulation, individual arcs $(i, j)^q$ can be naturally removed. For instance, it is quite typical that, at a certain point of the BCP, 95 % of the arc-load variables were already fixed to zero, while the fixing on arcs would not achieve 80 %.

4.3 Route enumeration

Baldacci et al. [2] introduced a route enumeration based approach in order to close the duality gap after the root node. An elementary route can only be part of a solution that improves the best known upper bound if its reduced cost is smaller than the gap. The enumeration of elementary routes may be performed by a label setting algorithm, producing a set partitioning problem that is given to a general MIP solver. This may work very well if the size of the set R of enumerated routes is not too large. If $|R| <$ 20,000 the set partitioning is usually solved in less than 1 minute in a modern machine. Larger values of $|R|$ may cause the MIP solver to take too much time, $|R| > 100{,}000$ is often unpractical.

Contardo [8] proposed another strategy in order to better profit from the route enumeration. The enumeration is performed even if the resulting R has a few million routes, which are stored in a pool. The column and cut generation proceeds. However, instead of using a labeling algorithm, the pricing starts to be performed by inspection in the pool. Now, the non-robust cuts have little impact in the pricing complexity. Therefore, they may be separated in a very aggressive way. This usually increases substantially the lower bounds, allowing reductions in the pool size by fixing variables (that now are routes) by reduced costs. For example, he reported that the enumeration of instance M-n151-k12 with gap of 5 units produced a pool with 4M routes. The separation of non-robust cuts then reduced the gap to 1.5 and the final pool only had 13K routes. The resulting set partitioning was easily solved, finishing the instance.

In this work we used the enumeration scheme propose in [8], allowing pools with up to 50M routes. After enumeration, SRCs and also clique cuts (separated using the routines from [33]) are added. Anyway, since the enumeration of so many routes may be very time-consuming, the implementation of the labeling algorithms used for that purpose becomes critical. The forward route enumeration algorithm represents a feasible (elementary) path $P = (0, \ldots, i), i \in V$, as a label $L(P) = (c(P), \bar{c}(P), v(P) = i, q(P), V(P), S(P), h(P), pred(P))$ storing its original cost, reduced cost, end vertex, load, set of visited vertices, vector of states corresponding to the n_S lm-SRCs with non-zero dual variables in the current Master LP solution, a hash value depending on $V(P)$, and a pointer to its predecessor label. A label $L(P_1)$ dominates a label $L(P_2)$ if:

$$\text{(i) } v(P_1) = v(P_2), \quad \text{(ii) } V(P_1) = V(P_2)$$
$$\text{(iii) } c(P_1) \leq c(P_2).$$

It is very important to also use completion bounds to eliminate labels that can not be extended to routes with reduced cost smaller than the current gap, denoted by gap. Let $\widehat{F}$ be the forward completion bounds obtained running the backward labeling procedure with respect to the current values of $\bar{c}$ and $\bar{S}$. The forward route enumeration extends a label $L(P)$ with $v(P) = i$ to a customer j if $j \notin V(P)$ and:

$$\bar{c}(P) + \bar{c}^q_{ij} + \widehat{F}(j, q(P) + d_j) < gap.$$

Remark that all the n_S lm-SRCs are taken into account in the calculation of $\widehat{F}$, the relaxation is allowing ng-routes instead of only elementary routes. To accelerate the label dominance checking, the labels are stored in a hash table H having $|H|$ buckets, where $|H|$ is a power of 2. Each vertex i in V receives two random numbers between 0 and $|H| - 1$, $r_1(i)$ and $r_2(i)$. A label $L(P)$ is stored in bucket calculated by taking the bitwise exclusive-or of the numbers $r_1(i)$, $i \in V(P)$, and $r_2(v(P))$. This ensures that labels that may have a dominance relationship will be in the same bucket. The pseudocode of the forward enumeration procedure is shown in Algorithm 7. The algorithm also keeps a list $\mathcal{Q}$ of unprocessed labels, initially containing only the starting label. In the end of the algorithm, the labels $L(P)$ with $v(P) = 0$ represent all elementary routes with reduced costs not greater than the duality gap.

Algorithm 7 Procedure forward enumeration

Run backward labeling for computing $\widehat{F}$ completion bounds
$H(0) \leftarrow \{L_0 = (0, 0, 0, 0, \emptyset, \mathbf{0}, 0, nil)\}$
$Insert(\mathcal{Q}, L_0)$
while $\mathcal{Q}$ is not empty **do**
  $L_1 = (c_1, \bar{c}_1, i, q, V_1, S_1, h_1, \_) \leftarrow Remove(\mathcal{Q})$
  for all $(i, j)^q \in A_Q$ such that $x^q_{i,j}$ is not fixed to 0 **do**
    if $j \notin V_1$ **then**
      if $\bar{c}_1 + \bar{c}^q_{ij} + \widehat{F}(j, q + d_j) \leq gap$ **then**
        $c_2 \leftarrow c_1 + c_{ij}$, $\bar{c}_2 \leftarrow \bar{c}_1 + \bar{c}^q_{ij}$, $S_2 \leftarrow S_1$, $h_2 \leftarrow h_1 \oplus r_1(j)$
        for $s := 1, \ldots, n_S$ **do**
          if $j \in C(s)$ **then**
            $S_2[s] \leftarrow S_2[s] + p(s)$
            if $S_2[s] \geq 1$ **then**
              $\bar{c}_2 \leftarrow \bar{c}_2 - \sigma_s$, $S_2[s] \leftarrow S_2[s] - 1$
        $L_2 = (c_2, \bar{c}_2, j, q + d_j, V_1 \cup \{j\}, S_2, h_2,$ pointer to $L_1)$
        $insertLabel \leftarrow$ **true**
        $index \leftarrow h_2 \oplus r_2(j)$
        for all $\mathcal{L} \in H(index)$ **do**
          if L_2 dominates $\mathcal{L}$ **then** $Remove(\mathcal{Q}, \mathcal{L})$, $H(index) \leftarrow H(index) \setminus \{\mathcal{L}\}$, **break**
          else if $\mathcal{L}$ dominates L_2 **then** $insertLabel \leftarrow$ **false**, **break**
        if $insertLabel$ **then**
          $H(index) \leftarrow H(index) \cup \{L_2\}$
          $Insert(\mathcal{Q}, L_2)$

A similar enumeration procedure could also be built by extending labels backwards. For this purpose, the list $\mathcal{Q}$ is initialized with starting labels at the depot carrying all possible loads $1 \leq q \leq Q$. The backward enumeration in itself has no advantage. However, as happens with the pricing, it can be used as a component of a significantly faster bidirectional enumerator (as already done in [2] and in [9]). In this case, the forward and backward enumeration algorithms are run up to a point around $Q/2$ and a concatenation phase is used for retrieving the elementary paths with reduced cost smaller than the gap.

5 Strong branching

It is clear that route enumeration is an important element in some of the best performing algorithms for the CVRP, being able of drastically reducing the running times of some instances. Nevertheless, it is disturbing to consider that route enumeration, no matter how cleverly done, is an inherently exponential space procedure (it would remain exponential even if P = NP) that is bound to fail on larger/harder instances. However, one does not need to take a radical stance on completely avoiding branching. The hybrid strategy used in [24] and [27] performs route enumeration after solving each node. If a limit on the number of routes is reached, the enumeration is aborted and the BCP proceeds by traditional branching. Of course, since deeper nodes will have smaller gaps, at some point the enumeration will work.

The branching in our BCP is performed over sets $S \subseteq V_+$, imposing the disjunction $(\sum_{a \in \delta^-(S)} x_a = 1) \vee (\sum_{a \in \delta^-(S)} x_a \geq 2)$. Those branching constraints are robust and can be translated into arc reduced cuts for the pricing in both child nodes. The BCP in [12] adopted a similar branching over sets (already used in [20]). In order to better choose the branching set, that BCP used strong branching. Each set in a collection containing from 5 to 10 candidate sets was heuristically evaluated by applying a small number of column generation iterations in its child nodes. Remark that the exact evaluation of each candidate, performing full column generation (perhaps also cut generation) in both child nodes, would be too expensive.

The recent work by [32] showed that it is possible to obtain major improvements in a BCP performance by performing a more sophisticated and aggressive strong branching:

- The simpler branching over individual arcs was used. The procedure starts by performing a quick evaluation of 30 candidate arcs, producing a ranking. The best ranked candidate is then fully evaluated and becomes the incumbent winner. Then, other candidates with good ranking are better evaluated, but only while they have a reasonable chance of beating the incumbent winner.
- It is possible to collect statistics along the enumeration tree to help on choosing the candidates. The previous evaluations of an arc are good predictors of future evaluations.

His extensive experiments were performed both on CVRP and VRPTW instances. This strong branching was the key algorithmic element that allowed his BCP to solve the hard instance M-n151-k12, with optimum 1015, starting from the rather modest root lower bound of 1001.5. This BCP was the first algorithm to solve all the 56 VRPTW Solomon instances with 100 customers.

Inspired by those good results, we devised a hierarchical strong branching procedure for our BCP:

- The phase zero performs the first selection of candidate sets. In the root node, this is a collection of $\min\{n, 300\}$ sets, chosen by the proximity of $\sum_{a \in \delta^-(S)} \bar{x}_a$ to 1.5 in the current fractional solution. For a non-root node v, the collection has cardinality between 50 and 10, depending on $TS(v)$, the estimative of the size of the subtree rooted in v. Half of the candidates are chosen based on the history of previous calls to the strong branching procedure. In contrast, the choice of the remaining candidates favors fresh sets, that were never evaluated before.
- The phase one performs a rough evaluation of each candidate by solving the current restricted Master LP twice, adding the constraint corresponding to each child node. Column and cut generation are not performed. The resulting improvements in the lower bounds are usually overvaluations of the true improvements if that candidate is selected. The candidates are ranked by the product rule [1] and the best (between 20 and 3) candidates go to phase two. If $TS(v)$ is very small, Phase 2 is skipped and the branching is performed with the best candidate of Phase 1.
- Phase two performs quite precise evaluations. Column and cut generation are performed, the only difference to the standard solving is that only heuristic pricing is used. Actually, if $TS(v)$ is large, even the exact pricing may be called. The results of Phase 2 are not only used to select the candidate to perform the current branching, they are stored in tables (the branching history) for subsequent use in the phase zero.

The whole procedure is guided by the principle that the strong branching effort in a node should depend on the expected subtree size. The logic behind this is the following. If $TS(v)$ is large, even a small improvement in that branching will pay the computational cost of the precise evaluation of several candidates. On the other hand, if $TS(v)$ is small, the branching should be fast, relying on the historical data and on the rough evaluations of phase one. The estimation $TS(v)$, following the model proposed in [18], is calculated from the node gap (the upper bound minus the lower bound of v) and from the values of Δ_l and Δ_r, the estimatives of the average lower bound improvements in the left children (corresponding to constraints $=1$) and in the right children (constraints ≥ 2) of the subtree rooted at v. It is typical that Δ_l is larger than Δ_r, reflecting a quite unbalanced tree. In the first nodes, the evaluations of the candidates performed by strong branching itself are used as proxies for Δ_l and Δ_r.

As mentioned before, our BCP uses a hybrid strategy. Enumeration is tried after solving each node. However, the strong branching can still be performed after a successful enumeration. This happens when the final set of routes R is too large (we use 20,000 as the limit) for the MIP solver. Of course, in both children of an enumerated node the pricing will continue to be done by inspection in the pool.

6 Computational experiments

Our BCP was coded in C++, IBM ILOG CPLEX Optimizer 12.5 was used as the LP solver and MIP solver in the exact method, the CVRPSEP package [19] was used to

separate Rounded Capacity Cuts and Strengthened Combs, and routines from [33] were used to separate generic Clique Cuts after enumeration. The experiments were conducted in a single core of an Intel Xeon CPU E5-2667 v2 3.30 GHz with 264 GB RAM running CentOs Linux. We report results over the standard classes of instances (A, B, E, F, M, and P) used for testing exact methods for the CVRP. Since larger instances came into reach of the proposed BCP, results are also reported over other instances with up to 360 customers proposed in literature. All instances used in this paper are available in the CVRPLIB website (http://vrp.atd-lab.inf.puc-rio.br/).

Table 1 summarizes the performance of the new BCP, comparing with the recent exact algorithms for the CVRP on the standard benchmarks. As usual in the literature where similar tables appear, classes E and M are grouped together. Columns **Opt** indicate the number of instances solved to optimality. Columns **Gap** and **T(s)** are the average percent gap in the root node (before sending the set of enumerated routes to a MIP solver, if the method uses that technique) and average times in seconds over the indicated machines. In order to provide an indication of the relative speed of each machine, the scores found at PassMark site (https://cpubenchmark.net/singleThread.html) are reported in parenthesis. The labels **LLE04**, **FLL+06**, **BCM08**, **BMR11**, **Con12**, **CM14**, and **Rop12** refer to the algorithms proposed in [2,3,8,9,12,20] and [32], respectively. Label **BCP** refers to the proposed BCP. All averages of each method are computed only over the instances solved by it. This explains, for example, why BCP has larger average times in series E-M than other methods that could not solve some of its instances.

The new BCP algorithm has a good performance and could solve all those 81 instances to optimality. On instance M-n200-k16, it showed that the previous best known solution was not optimal. We should remark that instances F-n72-k4, P-n101-k4, and F-n135-k7 are still better solved by a branch-and-cut algorithm, like [20]. As in [12], we could have used a hybrid method that is able to automatically switch to a branch-and-cut after severe problems with column generation convergence are found. However, in order to stick to the BCP paradigm, we prefer to report the results of a version where convergence problems triggers a dual stabilization strategy, similar to the one described in [25]. In fact, without dual stabilization F-n135-k7 can not be solved in reasonable time.

Table 2 presents detailed information on the resolution of a selected set of larger instances. Besides M-n151-k12, M-n200-k16, and M-n200-k17, it includes results on two instances from [6] and seven instances from [14]. In those cases, the number of customers is displayed in parenthesis. For each instance and method, column **IUB** presents the initial upper bound used by the method. Columns **RLB1**, **ER1**, **RLB2**, **ER2**, **RT(s)** are root node information. They are the lower bound obtained before enumeration (**RLB1**), the number of routes enumerated (if the method performs it and if the enumeration succeeds), (**ER1**), the improved root node lower bound after route enumeration, obtained by adding additional non-robust cuts (if Contardo style enumeration is performed) (**RLB2**), the number of remaining enumerated routes after that (**ER2**) and the total root node computing time (**RT(s)**). The final lower bound given by **FLB**, which is in bold when optimal, the number of nodes in the search tree denoted by **Nds** and the total computational time in seconds **TT(s)** complete the table columns. Detailed results over each individual instance and the description of

Table 1 Comparison of recent algorithms on series A, B, E, F, M, and P

Class	NP	LLE04			FLL+06			BCM08			BMR11		
		Opt	Gap	T(s)	Opt	Gap	T(s)	Opt	Gap	T(s)	Opt	Gap	T(s)
A	22	15	2.06	6638	22	0.81	1961	22	0.2	118	22	0.13	30
B	20	19	0.61	8178	20	0.47	4763	20	0.16	417	20	0.06	67
E-M	12	3	2.10	39592	9	1.19	126987	8	0.69	1025	9	0.49	303
F	3	3	0.06	1046	3	0.14	2398				2	0.11	164
P	24	16	2.26	11219	24	0.76	2892	22	0.28	187	24	0.23	85
Total	81	56			78			72			77		
Processor		Celeron 700 MHz (?)			Pent. 4 2.4 GHz (561)			Pent. 4 2.6 GHz (624)			X7350 2.93 GHz (1108)		
Class	NP	Con12			CM14			Rop12			BCP		
		Opt	Gap	T(s)	Opt	Gap	T(s)	Opt	Gap	T(s)	Opt	Gap	T(s)
A	22	22	0.07	59	22	0.09	59	22	0.57	53	22	0.36	2.24
B	20	20	0.05	89	20	0.08	34	20	0.25	208	20	0.14	6.87
E-M	12	10	0.30	2807	10	0.27	1548	10	0.96	44295	12	0.33	2712
F	3	2	0.06	3	3	0.03	27722	3	0.25	2163	3	0.00	3183
P	24	24	0.13	43	24	0.18	240	24	0.69	280	24	0.42	21.3
Total	81	78			79			79			81		
Processor		E5462 2.8 GHz (1204)			E5462 2.8 GHz (1204)			i7-2620M 2.7 GHz (1563)			E5-2667 3.3 GHz (1996)		

Table 2 Detailed results over selected instances

Instance	Algo	IUB	RLB1	RLB2	RT(s)	FLB	Nds	TT(s)
M-n151-k12	BMR11	1015	1004.3	–	380	1004.3	1	380
	Con12	1015	1008.9	1012.5	19041	**1015**	1	19699
	Rop12	1015	1001.5	–		**1015**	5268	417146
	CM14	1015	1011.2	1012.6	9942	**1015**	1	10110
	BCP	1015	1011.2	1013.0	180	**1015**	1	186
M-n200-k16	BMR11		1256.6	–	319	1256.6	1	319
	Con12	1278	1263.0	–	265589	1263.0	1	265589
	Rop12	1278	1253.0	–		1258.2	106	7200
	CM14	1278	1266.9	–	432000	1266.9	1	432000
	BCP	1278	1266.6	–	807	**1274**	75	28620
M-n200-k17	BMR11	1275	1258.7	–	436	1258.7	1	436
	Con12	1275	1265.1	–	34351	1265.1	1	34351
	Rop12	1276	1255.3	–		1261.4	144	7200
	CM14	1275	1269.2	–	432000	1269.2	1	432000
	BCP	1275	1268.8	–	695	**1275**	7	3479
C4 (150)	BCP	1028.42	1025.0	1026.4	565	**1028.42**	3	1117
C5 (199)	BCP	1291.29	1284.62	–	904	**1291.29**	57	22090
G17 (240)	BCP	707.76	705.01	–	345	**707.76**	15	14693
G13 (252)	BCP	857.19	851.93	–	4803	851.93	1	4803
G9 (255)	BCP	579.71	576.86	–	9013	576.86	1	9013
G18 (300)	BCP	995.13	993.22	–	381	**995.13**	23	13769
G14 (320)	BCP	1080.55	1076.10	–	8124	**1080.55**	1175	≈39 days
G10 (323)	BCP	736.26	731.28	–	25689	731.28	1	25689
G19 (360)	BCP	1365.60	1362.76	–	467	**1365.60**	339	162405

additional mechanisms to detect and exploit the symmetry present in the instances G9, G10, G13, G14, G17, G18, and G19 are provided in the appendix.

7 Conclusions

This paper have described a BCP algorithm for the CVRP that was the result of a deliberate effort of testing, improving and combining ideas proposed by several authors. The obtained results were very positive, considerably increasing the size where instances can be expected to be solved to optimality. We conclude by stating more personal views on what general BCP algorithm construction can learn from that CVRP experience:

- The separation of non-robust cuts should be as much integrated with the pricing as possible. More precisely, besides taking classical polyhedral considerations into account, the non-robust cuts should be designed in order to minimize their impact on the specific kind of algorithm used in the pricing. In the BCP presented in this paper, the limited memory SRCs has a quite odd algorithmic definition that only makes sense in the context of the labeling algorithm. For example, suppose an alternative BCP for the CVRP where a MIP model is used to price elementary routes. The limited memory SRCs would not fit in that algorithm, they would probably cause more negative impact in the pricing than ordinary SRCs.
- When designing non-robust cuts, it is desirable to have a parameter that allows a smooth control on cut strength vs impact in the pricing. In the case of limited memory SRCs, the parameter M has that role.
- Even if designed and separated in a careful way, at some point the non-robust cuts can indeed make the pricing intractable, halting the BCP algorithm. Therefore, it is advisable to have suitable escape mechanisms, like the rollback introduced in this work.
- Fixing variables from the original formulation by reduced costs can help a lot. But this fixing can be more effective if the original formulation is chosen in order to match the algorithm using in the pricing. In fact, in the arc-load formulation, the vertices in V_Q have a one-to-one correspondence with the buckets in the labeling algorithm.
- The idea of finishing a BCP by enumerating all columns that may be part of the optimal solution into a pool and perform pricing by inspection after that and eventually sending a much reduced problem to a MIP solver can be quite effective in some instances. However, for consistency reasons, it should be hybridized with traditional branching.
- Strong branching is now a standard technique for improving branch-and-bound or branch-and-cut performance. Our experience confirms the statement by [32] that it can also improve a lot the performance of BCP algorithms.

Appendix 1

We report detailed computational results for a large set of CVRP instances, comparing the proposed BCP with some of the recent exact methods described in Sect. 1. Additional computational results include a comparison of five variants of the pricing presented in Sect. 4.1. Moreover, some techniques used in the BCP to better handle special instances with a high degree of symmetry are presented.

Detailed computational results

The BCP was run over the standard classes of instances (A, B, E, F, M, and P, available at www.branchandcut.org) used for testing exact methods for the CVRP. The name of an instance from this benchmark follows the general format X-nY-kZ, where X denotes the class it belongs to, Y corresponds to the number of vertices and Z refers to the *fixed* number of routes (K) required for any feasible solution. As usual in the literature of exact methods, the cost matrix for these instances is calculated from depot and customers coordinates, following the TSPLIB convention of rounding the euclidean distances to the nearest integer. Moreover, in order to better compare our results with those presented in [3], our BCP uses for each of these instances the same initial upper bound they used.

We also report results over instances traditionally used in the literature on heuristic methods, proposed in [5,6] (instances with initial letter "C") and [14] (instances with initial letter "G"). In those cases, the convention is *not rounding* the Euclidean distances and *not fixing* the number of routes. The initial upper bound used for each of these instances corresponds to the best solution value available in the literature.

Tables 3, 4, 5, 6 and 7 present detailed results for all instances from the datasets A, B, E-M, F and P, respectively. For each instance, column Ins gives the name of the instance, UB presents the initial upper bound used and OPT shows the value of the optimal solution found (values in bold indicate proven optimality for the first time). The following 10 columns are root node information. The first of these columns indicates the lower bound obtained using only robust cuts (RLB) and the next four columns report information immediately before enumeration: the improved lower bound due the addition of non-robust lm-SRCs (NRLB), the number of active lm-SRCs (i.e., those with non-zero dual variables) in the master LP (nS), the percentage of arcs $(i, j)^q \in A_Q$ fixed to zero (Fix) and the accumulated time up to this point (T1). Then, the number of routes enumerated (the word "limit" indicates that the enumeration was unsuccessful), (|R|i), the improved root node lower bound after route enumeration, obtained by adding additional non-robust SRCs and clique cuts (NLB), the number of remaining enumerated routes after that (|R|f), the time spent with route enumeration plus the time to go from lower bound NRLB to lower bound NLB (or only the time spent with enumeration, if it is aborted) (T2) and, finally, the time spent solving the resulting MIP (TMIP) complete the information of the root. Columns |R|i to TMIP contain values only for the instances where a successful enumeration is performed at the root node. Column Nds gives the number

Table 3 Detailed results on CVRP instances of class A

Ins	UB	OPT	RLB	NRLB	nS	Fix	T1	|R|i	NLB	|R|f	T2	TMIP	Nds	TT	BMR11	Con12	Rop12	CM14
A-n37-k5	669	669	667.5	667.5	0	98.6	0.57	147	667.5	95	0.06	0.01	1	0.64	6	8.0	3	4.3
A-n37-k6	949	949	937.1	937.1	0	91.6	0.29	1743	940.1	811	0.08	0.09	1	0.47	8	20.6	17	9.7
A-n38-k5	730	730	723.4	723.4	0	94.2	0.48	947	723.6	551	0.08	0.06	1	0.61	7	13.5	4	7.7
A-n39-k5	822	822	817.8	817.8	0	93.2	0.67	2863	818.9	777	0.10	0.05	1	0.82	7	19.5	5	16.8
A-n39-k6	831	831	824.5	824.5	0	96.5	0.43	723	825.2	524	0.07	0.05	1	0.55	7	12.8	6	6.4
A-n44-k6	937	937	934.9	934.9	0	99.1	0.49	125	937.0	0	0.07		1	0.56	5	11.5	3	10.5
A-n45-k6	944	944	941.2	941.2	0	98.9	0.72	142	944.0	0	0.07		1	0.79	12	30.4	11	12.8
A-n45-k7	1146	1146	1140.5	1140.5	0	94.7	0.64	1524	1142.3	515	0.10	0.04	1	0.77	11	27.7	10	17.6
A-n46-k7	914	914	914.0	914.0	0	100	0.66						1	0.66	8	7.1	1	5.8
A-n48-k7	1073	1073	1071.6	1071.6	0	99.4	0.82	87	1073.0	0	0.06		1	0.88	15	24.3	7	11.5
A-n53-k7	1010	1010	1003.7	1003.7	0	93.7	1.44	8355	1004.6	1956	0.15	0.2	1	1.78	24	30.0	29	22.4
A-n54-k7	1167	1167	1155.5	1155.5	0	83.1	1.80	53392	1165.3	310	1.30	0.05	1	3.15	35	95.4	30	46.7
A-n55-k9	1073	1073	1068.5	1068.5	0	97.3	0.91	713	1069.4	449	0.09	0.04	1	1.04	19	18.3	6	15.0
A-n60-k9	1354	1354	1345.0	1345.0	0	88.5	1.45	23546	1351.9	516	0.97	0.11	1	2.53	46	58.5	84	38.8
A-n61-k9	1034	1034	1023.6	1023.6	0	91.9	1.29	6976	1024.5	4571	0.14	0.64	1	2.07	37	61.0	69	33.0
A-n62-k8	1290	1288	1280.8	1280.8	0	88.6	2.56	125919	1285.4	4697	2.51	2.93	1	8.01	96	104.6	80	95.4
A-n63-k9	1616	1616	1608.6	1608.6	0	91.6	1.81	9647	1611.7	1190	0.18	0.12	1	2.11	40	77.5	40	49.7
A-n63-k10	1315	1314	1302.5	1302.5	0	88.7	1.29	23094	1308.6	2785	0.75	0.81	1	2.84	51	73.7	42	585.7
A-n64-k9	1402	1401	1387.8	1387.8	0	86.7	1.49	120559	1394.5	7698	1.78	3.24	1	6.5	63	224.3	379	83.2
A-n65-k9	1174	1174	1168.2	1168.2	0	95.9	1.88	3338	1169.7	861	0.14	0.08	1	2.1	32	47.3	21	35.5
A-n69-k9	1159	1159	1143.8	1143.8	0	85.4	1.97	103294	1155.4	1022	1.56	0.18	1	3.72	55	89.9	135	62.6
A-n80-k10	1763	1763	1756.5	1756.5	0	95.6	4.03	12703	1757.1	7284	0.21	2.33	1	6.57	82	247.9	186	135.5
Solved														22	22	22	22	22
Average														2.24	30.3	59.3	53.0	59.4
Geometric mean														1.50	20.4	36.6	18.9	25.3

Table 4 Detailed results on CVRP instances of class B

Ins	UB	OPT	RLB	NRLB	nS	Fix	T1	|R|i	NLB	|R|f	T2	TMIP	Nds	TT	BMR11	Con12	Rop12	CM14
B-n38-k6	805	805	805.0	805.0	0	100	0.33						1	0.33	4	4.4	2	3.8
B-n39-k5	549	549	549.0	549.0	0	100	0.73						1	0.73	6	0.2	13	0.1
B-n41-k6	829	829	829.0	829.0	0	100	0.46						1	0.46	4	5.0	4	2.5
B-n43-k6	742	742	737.2	737.2	0	89.0	0.69	6643	737.8	3517	0.13	0.43	1	1.25	23	24.6	9	14.6
B-n44-k7	909	909	909.0	909.0	0	100	0.69						1	0.69	3	6.3	2	0.4
B-n45-k5	751	751	751.0	751.0	0	100	1.15						1	1.15	9	21.0	19	8.3
B-n45-k6	678	678	678.0	678.0	0	100	0.89						1	0.89	10	20.6	11	10.4
B-n50-k7	741	741	741.0	741.0	0	100	0.91						1	0.91	4	3.3	2	0.2
B-n50-k8	1312	1312	1303.5	1303.5	0	77.3	0.95	188964	1308.6	5170	3.2	2.71	1	6.86	147	144.3	51	67.9
B-n51-k7	1032	1032	1026.8	1026.8	0	86.4	1.16	15802	1027.5	3238	0.2	0.34	1	1.69	34	30.0	42	22.6
B-n52-k7	747	747	747.0	747.0	0	100	1.36						1	1.36	8	3.5	12	2.4
B-n56-k7	707	707	705.0	705.0	0	97.8	1.32	339	705.0	158	0.08	0.01	1	1.40	11	26.6	22	41.6
B-n57-k7	1153	1153	1153.0	1153.0	0	100	1.75						1	1.75	25	123.4	32	26.0
B-n57-k9	1598	1598	1596.0	1596.0	0	98.3	1.21	439	1596.2	203	0.08	0.02	1	1.31	19	37.3	14	27.9
B-n63-k10	1496	1496	1487.2	1487.2	0	88.6	1.43	41758	1496.0	0	3.89		1	5.32	55	89.0	27	38.9
B-n64-k9	861	861	861.0	861.0	0	100	2.16						1	2.16	15	44.5	26	10.0
B-n66-k9	1316	1316	1308.9	1308.9	0	83.4	2.31	168189	1314.8	262	2.35	0.04	1	4.69	292	202.5	43	85.8
B-n67-k10	1032	1032	1027.6	1027.6	0	94.5	1.68	4727	1028.2	2700	0.14	0.40	1	2.22	43	60.9	49	27.2
B-n68-k9	1275	1272	1263.8	1266.5	58	81.0	6.70	496047	1269.2	45308	41.34		3	98.03	526	762.9	3671	178.0
B-n78-k10	1221	1221	1217.2	1217.2	0	96.2	3.79	3649	1217.6	2071	0.17	0.22	1	4.19	97	169.6	107	108.5
Solved														20	20	20	20	20
Average														6.87	66.8	89.0	208	33.9
Geometric mean														1.85	20.9	25.1	20.2	10.3

Table 5 Detailed results on CVRP instances of classes E and M

Ins	UB	OPT	RLB	NRLB	nS	Fix	T1	|R|i	NLB	|R|f	T2	TMIP	Nds	TT	BMR11	Con12	Rop12	CM14
E-n51-k5	521	521	519.3	519.3	0	99.1	1.78	330	519.4	231	0.10	0.02	1	1.90	8	13.7	9	11.2
E-n76-k7	682	682	671.1	675.3	56	94.6	12.05	132933	679.1	3890	5.14	1.19	1	18.38	152	378.7	1819	766.4
E-n76-k8	735	735	726.8	726.8	0	89.6	4.50	415444	732.3	2042	5.57	0.35	1	10.43	82	171.8	767	137.6
E-n76-k10	830	830	817.3	817.3	0	85.5	2.26	393254	825.5	4903	4.70	1.70	1	8.66	114	183.9	1666	134.3
E-n76-k14	1021	1021	1006.9	1006.9	0	87.7	0.76	40533	1014.9	3311	0.86	1.21	1	2.83	52	141.3	1429	107.2
E-n101-k8	815	815	804.4	811.2	106	95.9	50.70	169728	814.7	2	16.63	0.01	1	67.34	579	1092	6611	1560
E-n101-k14	1071	1067	1053.6	1059.7	73	88.0	7.13	1140176	1064.2	102464	43.02		5	109.9	453	662.5	4325	587.9
M-n101-k10	820	820	820.0	820.0	0	100	3.52						1	3.52	35	13.2	34	0.1
M-n121-k7	1034	1034	1032.5	1032.5	0	99.0	35.00	25733	1032.9	4002	0.75	0.35	1	36.10	1.249	5713	11140	2065
M-n151-k12	1015	1015	1000.4	1011.2	217	97.6	140.3	125992	1013.0	5784	39.89	5.96	1	186.1		19699	417146	10110
M-n200-k16	1278	1274	1252.6	1266.6	398	86.7	807.2	Limit					75	28620				
M-n200-k17	1275	1275	1255.1	1268.8	404	94.1	694.7	Limit					7	3480				
Solved														12	9	10	10	10
Average over all instances solved by the method														2712	303	2807	44495	1548
Average over the 9 instances solved by all methods														29.8	303	930	3089	597
Geometric mean over the 9 instances solved by all methods														12.7	124	221	880	114

Table 6 Detailed results on CVRP instances of class F

Ins	UB	OPT	RLB	NRLB	nS	Fix	T1	|R|i	NLB	|R|f	T2	TMIP	Nds	TT	BMR11	Con12	Rop12	CM14
F-n45-k4	724	724	724	724.0	0	100	27.64						1	27.64	23	0.7	13	0.40
F-n72-k4	237	237	235.58	235.6	0	99.9	3210.01	1020	237.0	0	1743		1	4953	304	4.0	3349	0.10
F-n135-k7	1162	1162	1160.3	1162.0	61	97.1	4568.97						1	4569			3126	83165
Solved														3	2	2	3	3
Average over all instances solved by the method														3183	164	2.4	2163	27772
Average over the 2 instances solved by all methods														2490	164	2.4	1681	0.2
Geometric mean over the 2 instances solved by all methods														370	84	1.6	208	0.2

Table 7 Detailed results on CVRP instances of class P

Ins	UB	OPT	RLB	NRLB	nS	Fix	T1	|R|i	NLB	|R|f	T2	TMIP	Nds	TT	BMR11	Con12	Rop12	CM14
P-n16-k8	450	450	448.0	448.0	0	93.5	0.01	23	448.0	20	0.05	0	1	0.06	1		0	
P-n19-k2	212	212	212.0	212.0	0	100	0.14						1	0.14	1		1	
P-n20-k2	216	216	213.0	213.0	0	98.1	0.13	80	216.0	0	0.06		1	0.18	1		1	
P-n21-k2	211	211	211.0	211.0	0	100	0.19						1	0.19	1		1	
P-n22-k2	216	216	216.0	216.0	0	100	0.22						1	0.22	1		2	
P-n22-k8	603	603	603.0	603.0	0	100	0.01						1	0.01	1		0	
P-n23-k8	529	529	529.0	529.0	0	100	0.01						1	0.01	1		0	
P-n40-k5	458	458	458.0	458.0	0	100	0.69						1	0.69	2	4.1	3	2.8
P-n45-k5	510	510	507.0	507.0	0	97.5	1.10	1269	507.2	505	0.09	0.04	1	1.24	6	12.1	13	7.8
P-n50-k10	696	696	689.4	689.4	0	96.7	0.27	945	689.5	882	0.08	0.06	1	0.41	9	19.3	6	10.4
P-n50-k7	554	554	551.8	551.8	0	98.7	0.92	537	552.0	327	0.09	0.03	1	1.03	68	41.5	127	21.9
P-n50-k8	631	631	617.2	617.2	0	86.1	0.47	15387	618.1	7533	0.16	1.31	1	1.94	9	19.1	6	13.2
P-n51-k10	741	741	736.2	736.2	0	97.1	0.36	567	736.2	551	0.07	0.03	1	0.47	11	20.0	6	11.9
P-n55-k10	699	694	681.8	681.8	0	80.8	0.47	49958	689.4	8742	1.01	2.48	1	3.96	17	47.5	98	26.7
P-n55-k15	993	989	972.5	972.5	0	89.2	0.20	2410	972.8	2168	0.08	0.31	1	0.59	18	17.4	36	18.6
P-n55-k7	568	568	559.2	559.2	0	92.1	1.21	35379	565.4	783	1.33	0.17	1	2.71	29	38.9	110	26.3
P-n55-k8	588	588	580.6	580.6	0	93.7	0.97	13849	581.0	8724	0.17	1.61	1	2.75	27	85.8	16	77.6
P-n60-k10	756	744	738.9	738.9	0	81.8	0.60	71297	744.0	23209	2.00	1.45	1	4.04	30	27.8	15	20.4
P-n60-k15	1033	968	963.5	963.5	0	15.2	0.29	1169158	968.0	926739	12.07	0.42	1	12.79	73	30.4	4	20.7
P-n65-k10	792	792	787.0	787.0	0	97.6	1.00	2054	788.2	739	0.12	0.04	1	1.15	14	42.2	14	31.2
P-n70-k10	834	827	814.3	814.3	0	75.0	1.44	1085904	825.6	27956	25.14	7	1	33.58	166	132.6	534	76.1
P-n76-k4	593	593	589.3	590.3	39	97.1	23.10	247154	592.3	207	11.69	0.01	1	34.79	118	362.7	497	856.7
P-n76-k5	627	627	617.7	622.1	87	95.0	61.53	136034	625.3	998	10.82	0.25	1	72.60	282	1330.9	3888	859.6
P-n101-k4	681	681	678.6	679.8	101	98.2	312.6	42853	679.8	3868	22.86	0.73	1	336.19	1.155	5793.3	1343	3689.1
Solved														24	24	24	24	24
Average														21.32	85	472	280	339

of nodes in the branch-and-bound tree and `TT` gives the total time in seconds. Additionally, there are four other columns: `BMR11`, `Con12`, `Rop12` and `CM14`. They are, respectively, the time in seconds reported by [3,8,32], and [9]. For each method, the number of solved instances, average time and geometric mean time are also reported.

We remark that the Ref. [32] only presents the aggregated results shown in Table 1. The detailed results for Røpke's method shown in Tables 3, 4, 5, 6 and 7 were kindly provided to us in 2012 by personal communication. In fact, as one of the first tasks of the project that culminated with BCP proposed in this article, we replicated his method and obtained similar results in each instance. Therefore, we can attest that those results are trustworthy. This is an important concern, since they did not appeared yet in a peer reviewed paper.

Table 8 shows detailed results for selected instances proposed by [5] and [6] and Table 9 presents detailed results for instances with up to 360 customers proposed by [14]. Those instances are the most relevant for comparing recent heuristic approaches. For each instance, two new columns indicate the number of customers and the number of routes in the optimal solution found. These tables do not include columns of competing exact methods because no such results are reported in the literature.

Figures 3, 4 and 5 illustrate the optimal solution found for instances M-n200-k16, G14 and G19.

Table 10 shows detailed results over a number of selected instances of five variants of the pricing: 1—forward labeling (Algorithm 3); 2—bidirectional labeling with concatenation at $Q/2$; 3—bidirectional labeling with dynamic determination of the concatenation point (Algorithm 5); 4—the later algorithm with mild use of completion bounds (n'_S is small); 5—the same algorithm with aggressive use of completion bounds (n'_S is larger). For each instance, there are five rows, each giving the results for a pricing variant. The name of the columns are: **Ins**—the name of the instance; **UB**—the initial upper bound; **LB**—the root lower bound obtained; **Fix**—the percentage of arcs $(i, j)^q \in A_Q$ fixed at the end of the root node; **Alg**—the pricing variant; **nS**—the number of lm-SRCs with non-zero duals; **nFL**—the number of forward labels; **FT**—the time to run the forward labeling; **nBL**—the number of backward labels; **BT**—the time to run the backward labeling; **CT**—the time to run the concatenation; **CBT**—the time to calculate completion bounds; **TT** - the total time of the pricing; and **qf**—the concatenation point. The number of customers is indicated in parentheses for the instances proposed in [14]. Note that for variant 1, $\mathbf{qf} = Q$.

The results (columns from **nS** to **qf**) are an average of the rounds of the exact pricing performed in the end of the root node, after the last separation of lm-SRCs. In order to ensure a fair comparison, in this particular experiment, all the pricing variants are run "in parallel" with exactly the same input, the reduced costs. We only use columns found by variant 5, the columns from the other variants are discarded.

The results in Table 10 show that variant 5 is the best pricing algorithm, at least for very hard instances, those with more than 199 customers. Therefore, it was the variant chosen for the proposed BCP.

Table 8 Detailed results on instances from [5] and [6]

Ins	n	Ksol	UB	OPT	RLB	NRLB	nS	Fix	T1	|R|i	NLB	|R|f	T2	TMIP	Nds	TT
C1	50	5	524.61	524.61	522.95	522.95	0	97.8	1.81	1987	523.10	1450	0.12	0.11	1	2.04
C2	75	10	835.26	**835.26**	822.41	822.41	0	84.0	2.54	580895	832.71	2839	9.2	1.89	1	13.63
C3	100	8	826.14	**826.14**	814.92	823.35	147	95.8	93.41	166437	826.08	507	28.97	0.12	1	122.5
C12	100	10	819.56	**819.56**	819.56	819.56	0	99.1	3.73	147	819.56	62	0.06	0.01	1	3.80
C11	120	7	1042.12	**1042.12**	1041.94	1041.94	0	99.6	35.00	702	1042.12	316	0.3	0.03	1	35.33
C4	150	12	1028.42	**1028.42**	1013.27	1025.06	272	96.7	337.1	370221	1026.43	49069	227.9		3	1117
C5	199	16	1291.29	**1291.29**	1270.09	1284.62	430	92.0	904.4	Limit					57	22090
Solved																7
Average																3341
Geometric mean																73.2

Table 9 Detailed results on instances from [14]

Ins	n	Ksol	UB	OPT	RLB	NRLB	nS	Fix	T1	|R|i	NLB	|R|f	T2	TMIP	Nds	TT
G17	240	22	707.76	707.76	700.91	705.01	682	90.3	345.1	Limit					15	14693
G13	252		857.19		845.80	851.93	582	94.0	4803	Limit						
G9	255		579.71		575.67	576.86	424	94.3	9013	Limit						
G18	300	27	995.13	995.13	988.36	993.22	807	95.5	381.2	Limit					23	13769
G14	320	30	1080.55	1080.55	1070.50	1076.10	750	96.3	8124	Limit					1175	3354659
G10	323		736.26		729.86	731.28	587	90.3	25689	Limit						
G19	360	33	1365.60	1365.60	1356.18	1362.76	727	94.3	467.2	Limit					339	162405
Solved																4
Average over the 4 instances solved by BCP																886381
Geometric mean over the 4 instances solved by BCP																102464

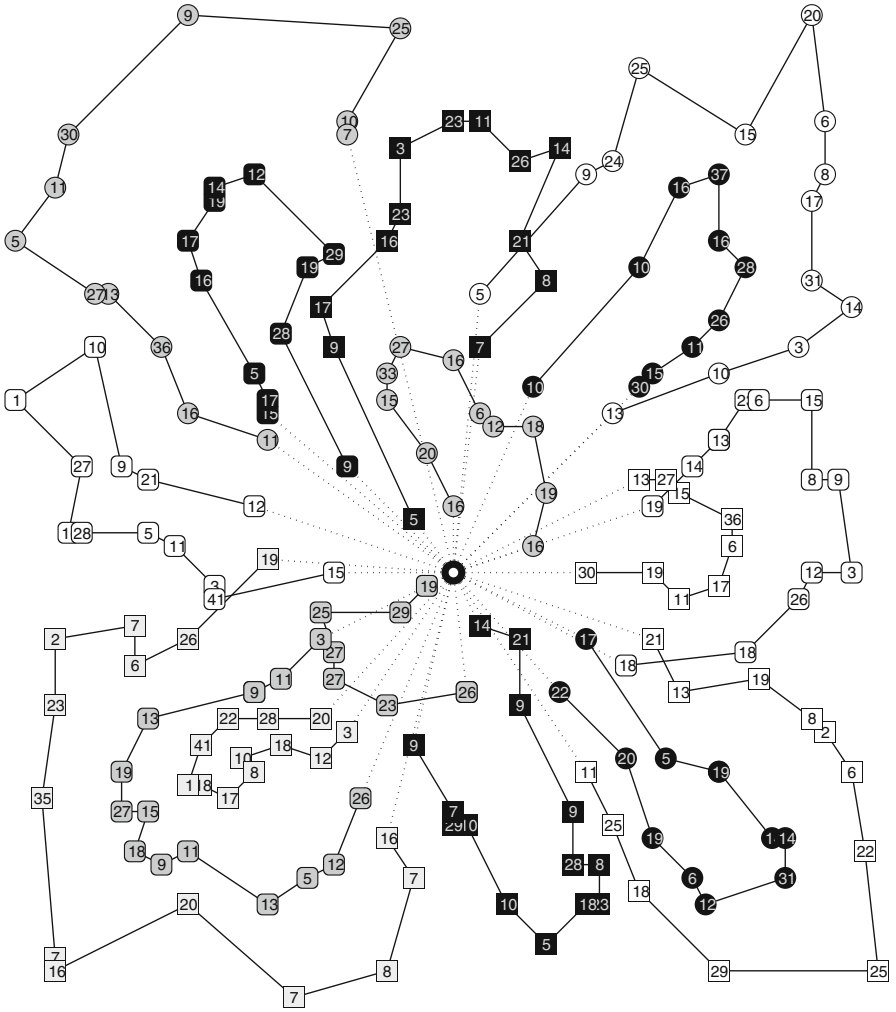

Fig. 3 Optimal solution of M-n200-k16, value 1274

7.1 Handling symmetrical instances

A high level of symmetry is observed in most instances from [14]. This impacts negatively the BCP performance, making the cutting and branching operations less efficient than usual. In fact, whenever a fractional solution is cut, it is likely that a symmetric solution with the same cost will appear in its place. The lower bounds can only move after all symmetric solutions are cut. We implemented some special techniques in our BCP in order to mitigate this negative impact.

The proposed BCP has a procedure that automatically detects, for instances that use non-rounded Euclidean distances calculated from the customer and depot coordinates, angles of rotation and reflection with respect to the depot that result in equivalent instances. For example, the instance G14 (see Fig. 4) remains the same if it is rotated

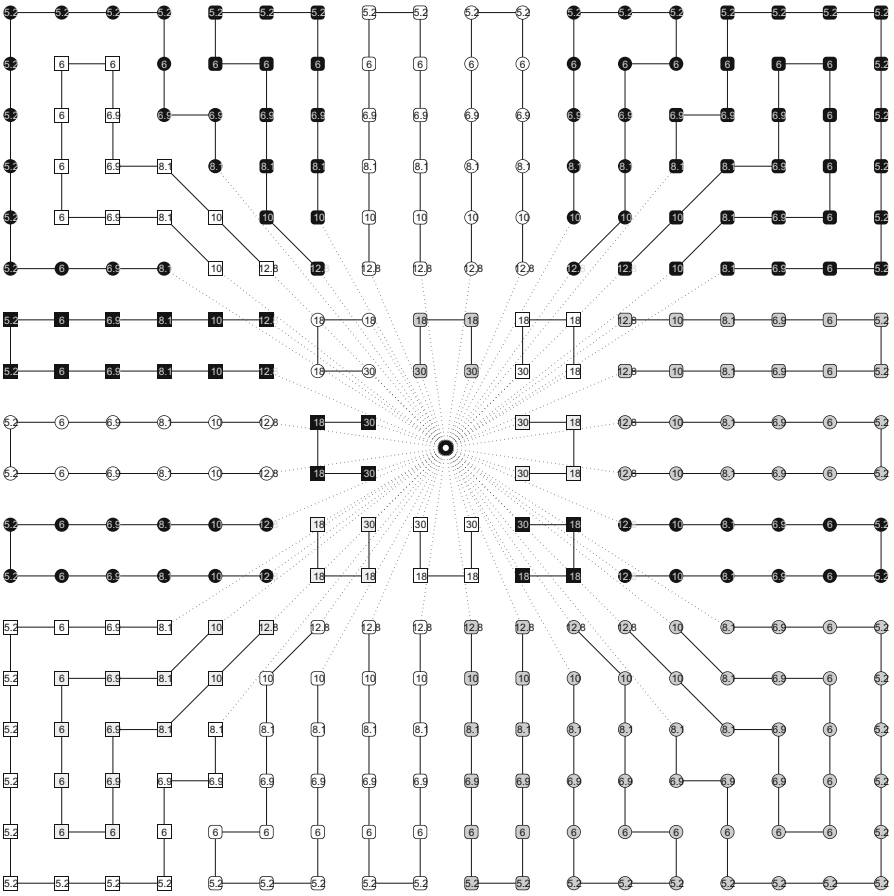

Fig. 4 Optimal solution of G14, value 1080.55

around the depot in an angle of 90 degrees or reflected over a horizontal line passing through the depot. As a result, for every feasible solution for this instance (fractional or integral), there are eight symmetric solutions with the same cost. For the instance G19, shown in Fig. 5, the number of symmetric solutions is 12 because the rotation angle is 60 degrees. More precisely, the procedure uses the rotation and reflection operations mentioned before to partition the set of arcs into symmetry groups. In instances G9–G12, each group has two members, in instances G13–G19, each group has eight members, in instances G17–G20, the groups have size 12. Instances G1–G8 are also very symmetric, but they are not considered in this CVRP work because they have additional distance constraints.

Let $Sym(\phi)$ be the set of all elements of a given instance I that correspond to the element ϕ in one of the equivalent instances that belong to symmetry group of I, an element being a vertex, arc, route or solution. The symmetry detection procedure works as follows. First, it seeks for rotation symmetries. For an arbitrarily chosen customer vertex i, it tests every other vertex j having $d_j = d_i$ and $c_{0j} = c_{0i}$ as candidates to belong to $Sym(i)$. For each candidate, it checks whether or not the corresponding

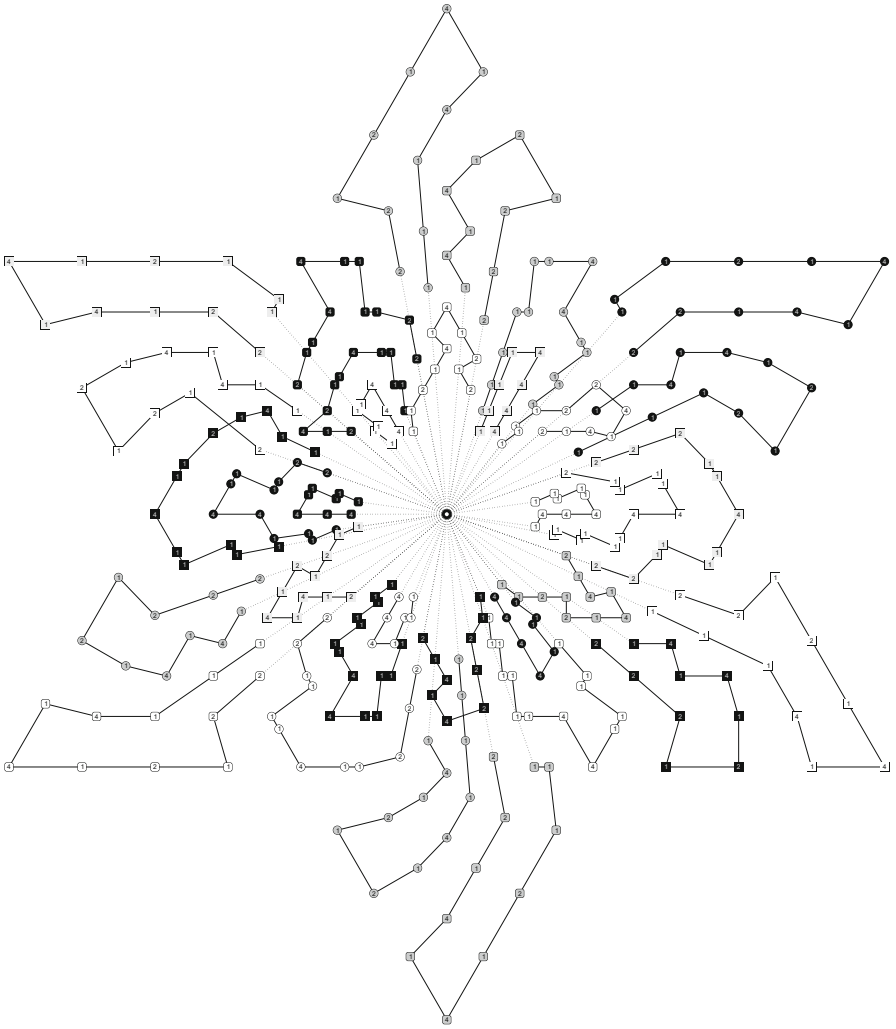

Fig. 5 Optimal solution of G19, value 1365.60

rotation angle with respect to the depot can be applied to every vertex leading to another vertex with the same demand value. In the positive case, the rotation angle is accepted. After all candidates are checked, the smallest accepted rotation angle is used. Then, the procedure seeks for a straight line passing through the depot that defines a reflection symmetry operation. The used algorithm is similar but, since any vertex can be exactly over the reflection line all pairs of vertices are tested as candidates. For each candidate, the corresponding reflection line is applied to all vertices, being accepted if every vertex that do not belong to the line has a corresponding symmetric. In this case the procedure stops upon the first accepted reflection line. Although the time to detect symmetries can be potentially significant, we observed that it is negligible (less than one hundredth of second) for all instances from [14].

Table 10 Results for 5 variants of the pricing

Ins	UB	LB	Fix	Alg	nS	nFL	FT	nBL	BT	CT	CBT	TT	qf
B-n68-k9	1272	1268.43	93.0	1	206	681, 215	162.13					162.13	100
				2		29, 305	1.10	123, 533	9.36	0.76		11.21	50
				3		56, 400	3.10	69, 091	3.60	0.77		7.46	57
				4		56, 886	3.40	66, 073	3.53	0.81	0.06	7.80	57
				5		55, 000	3.13	64, 386	3.54	0.81	4.83	12.31	57
E-n101-k8	815	814.39	98.3	1	190	81, 238	0.33					0.33	200
				2		11, 539	0.03	54, 223	0.19	0.03		0.25	100
				3		24, 802	0.08	30, 123	0.08	0.03		0.19	123
				4		24, 165	0.08	29, 837	0.08	0.03	0.04	0.23	123
				5		24, 577	0.08	28, 792	0.08	0.03	0.14	0.33	123
M-n151-k12	1015	1012.54	98.7	1	349	300, 434	3.12					3.12	200
				2		53, 043	0.28	213, 691	1.95	0.14		2.37	100
				3		110, 990	0.79	111, 542	0.79	0.16		1.73	122
				4		111, 731	0.78	107, 555	0.77	0.16	0.11	1.83	123
				5		107, 585	0.72	104, 803	0.71	0.16	0.63	2.23	122
M-n200-k16	1278	1266.13	86.0	1	362	824, 411	30.13					30.13	200
				2		205, 676	4.31	445, 318	10.60	0.20		15.10	100
				3		290, 721	7.10	322, 890	6.98	0.21		14.29	113
				4		282, 208	2.86	264, 891	2.86	0.21	1.00	6.92	118
				5		164, 252	1.63	293, 159	1.63	0.17	2.85	6.28	96

Table 10 continued

Ins	UB	LB	Fix	Alg	nS	nFL	FT	nBL	BT	CT	CBT	TT	qf
M-n200-k17	1275	1268.53	93.9	1	362	713, 344	15.80					15.80	200
				2		182, 203	2.24	442, 487	7.16	0.16		9.56	100
				3		275, 242	4.15	296, 893	4.15	0.17		8.46	116
				4		265, 776	2.12	240, 002	2.15	0.17	0.64	5.08	121
				5		183, 267	1.38	239, 635	1.36	0.15	2.26	5.16	104
G17 (240)	707.76	705.7	92.3	1	1015	574, 434	52.56					52.56	20
				2		142, 898	8.41	184, 784	12.47	1.61		22.48	10
				3		142, 898	8.58	184, 784	12.75	1.59		22.93	10
				4		175, 048	7.27	137, 401	6.60	1.73	0.14	15.74	11
				5		136, 400	5.41	127, 199	3.96	1.60	2.52	13.48	10
G13 (242)	857.19	851.74	94.1	1	598	8, 945, 606	711.68					711.68	1000
				2		691, 079	24.46	5, 939, 668	397.29	1.52		423.26	500
				3		2, 063, 475	105.74	2, 378, 698	106.14	1.32		213.20	635
				4		2, 018, 630	60.62	1, 974, 895	61.01	1.22	3.69	126.54	651
				5		1, 009, 351	26.79	1, 648, 510	26.84	1.56	21.63	76.83	548
G9 (255)	579.71	576.84	94.2	1	371	4, 721, 819	185.30					185.30	1000
				2		1, 868, 427	58.49	3, 812, 497	142.65	1.10		202.24	500
				3		2, 589, 046	90.78	2, 644, 367	90.84	0.86		182.48	568
				4		2, 556, 700	51.12	2, 172, 705	51.30	0.63	5.10	108.14	589
				5		818, 102	14.46	1, 629, 212	14.40	0.11	24.64	53.60	579
G14 (320)	1080.55	1076.12	96.1	1	712	6, 856, 892	338.01					338.01	1000
				2		1, 062, 545	28.18	5, 346, 066	234.80	0.41		263.39	500
				3		2, 393, 118	87.77	2, 601, 846	87.88	0.34		175.99	624
				4		2, 397, 374	39.57	1, 948, 837	39.74	0.31	5.06	84.67	652
				5		1, 181, 803	17.68	1, 863, 797	17.74	0.41	20.05	55.88	523

The BCP benefits from the symmetry detection in two ways.

1. For every variable x_a^q fixed to zero by reduced cost, all variables $x_{a'}^q$ where $a' \in Sym(a)$ can also be fixed to zero. This is valid because the same argument that was used to fix a can be turned into an argument to fix a', by only switching indices.
2. Consider the simple disjunction $(x_a = 1) \vee (x_a = 0)$ on the branching. The stronger disjunction $(x_a = 1) \vee (\sum_{a \in Sym(a)} x_a = 0)$ is also valid. This can be explained as follows: if all optimal solutions use at least one arc in $Sym(a)$, there is an optimal solution that uses a. It is important to remark that the left node (corresponding to $x_a = 1$) from this branching is not completely symmetrical anymore. This is good, but also means that the symmetry related tricks can not be used in the corresponding subtree. On the other hand, the right node (corresponding to $\sum_{a \in Sym(a)} x_a = 0$) is still symmetrical.
   Now, consider a disjunction $(\sum_{a \in \delta^-(S)} x_a = 1) \vee (\sum_{a \in \delta^-(S)} x_a \geq 2)$ on the branching. Let X be a non-empty proper subset of the arcs in A. We say that the subset $X' \subset A$, $|X'| = |X|$, belongs to $Sym(X)$ if every arc in X can be mapped into an arc in X' by the same combination of rotation and reflection operations. It can be seen that all possible sets $Sym(X)$ have the same cardinality $|Sym|$, which is also the cardinality of the groups of symmetrical arcs. We claim that the right side of disjunction can be strengthened to $(\wedge_{X' \in Sym(\delta^-(S))} \sum_{a \in X'} x_a \geq 2)$. In other words, $|Sym|$ inequalities can be imposed on the right node. This is valid because if in all optimal solutions there is a set $X' \in Sym(\delta^-(S))$ such that $\sum_{a \in X'} x_a = 1$, there is an optimal solution where $\sum_{a \in \delta^-(S)} x_a = 1$. Using this improved disjunction, the symmetry is partially broken in the left child but is maintained in the right child, and thus can be exploited again.

The use of those techniques makes a lot a difference in the BCP performance on highly symmetrical instances. For example, instance G17 takes more than 3 days to be solved without them.

References

1. Achterberg, T.: Constraint integer programming. PhD thesis, Technische Universitat Berlin (2007)
2. Baldacci, R., Christofides, N., Mingozzi, A.: An exact algorithm for the vehicle routing problem based on the set partitioning formulation with additional cuts. Math. Program. **115**(2), 351–385 (2008)
3. Baldacci, R., Mingozzi, A., Roberti, R.: New route relaxation and pricing strategies for the vehicle routing problem. Oper. Res. **59**(5), 1269–1283 (2011)
4. Boland, N., Dethridge, J., Dumitrescu, I.: Accelerated label setting algorithms for the elementary resource constrained shortest path problem. Oper. Res. Lett. **34**(1), 58–68 (2006)
5. Christofides, N., Eilon, S.: An algorithm for the vehicle-dispatching problem. Oper. Res. Q. **20**, 309–318 (1969)
6. Christofides, N., Mingozzi, A., Toth, P.: The vehicle routing problem. In: Christofides, N., Mingozzi, A., Toth, P., Sandi, C. (eds.) Combinatorial Optimization, pp. 315–338. Wiley Interscience, New York (1979)
7. Christofides, N., Mingozzi, A., Toth, P.: Exact algorithms for the vehicle routing problem, based on spanning tree and shortest path relaxations. Math. Program. **20**, 255–282 (1981)
8. Contardo, C.: A new exact algorithm for the multi-depot vehicle routing problem under capacity and route length constraints. Technical report, Archipel-UQAM 5078, Université du Québec à Montréal, Canada (2012)

9. Contardo, C., Martinelli, R.: A new exact algorithm for the multi-depot vehicle routing problem under capacity and route length constraints. Discrete Optim. **12**, 129–146 (2014)
10. Dantzig, G., Ramser, J.: The truck dispatching problem. Manag. Sci. **6**(1), 80–91 (1959)
11. Desrochers, M., Desrosiers, J., Solomon, M.: A new optimization algorithm for the vehicle routing problem with time windows. Oper. Res. **40**, 342–354 (1992)
12. Fukasawa, R., Longo, H., Lysgaard, J., Poggi de Aragão, M., Reis, M., Uchoa, E., Werneck, R.F.: Robust branch-and-cut-and-price for the capacitated vehicle routing problem. Math. Program. **106**(3), 491–511 (2006)
13. Godinho, M.T., Gouveia, L., Magnanti, T., Pesneau, P., Pires, J.: On time-dependent models for unit demand vehicle routing problems. In: International Network Optimization Conference (INOC) (2007)
14. Golden, B., Wasil, E., Kelly, J., Chao, I.: The impact of metaheuristics on solving the vehicle routing problem: algorithms, problem sets, and computational results. In: Fleet Management and Logistics, pp. 33–56. Springer, Berlin (1998)
15. Irnich, S., Desaulniers, G., Desrosiers, J., Hadjar, A.: Path-reduced costs for eliminating arcs in routing and scheduling. INFORMS J. Comput. **22**(2), 297–313 (2010)
16. Irnich, S., Villeneuve, D.: The shortest-path problem with resource constraints and k-cycle elimination for $k \geq 3$. INFORMS J. Comput. **18**(3), 391–406 (2006)
17. Jepsen, M., Petersen, B., Spoorendonk, S., Pisinger, D.: Subset-row inequalities applied to the vehicle-routing problem with time windows. Oper. Res. **56**(2), 497–511 (2008)
18. Kullmann, O.: Fundaments of branching heuristics. In: Biere, A., Heule, M., van Maaren, H., Walsh, T. (eds.) Handbook of Satisfiability, pp. 205–244. IOS Press, Amsterdam (2009)
19. Lysgaard, J.: CVRPSEP: a package of separation routines for the capacitated vehicle routing problem. Aarhus School of Business, Department of Management Science and Logistics (2003)
20. Lysgaard, J., Letchford, A., Eglese, R.: A new branch-and-cut algorithm for the capacitated vehicle routing problem. Math. Program. **100**, 423–445 (2004)
21. Martinelli, R., Pecin, D., Poggi, M.: Efficient elementary and restricted non-elementary route pricing. Eur. J. Oper. Res. **239**, 102–111 (2014)
22. Naddef, D., Rinaldi, G.: Branch-and-cut algorithms for the capacitated VRP, Chap. 3. In: Toth, P., Vigo, D. (eds.) The Vehicle Routing Problem, pp. 53–84. SIAM (2002)
23. Pecin, D., Pessoa, A., Poggi, M., Uchoa, E.: Improved branch-cut-and-price for capacitated vehicle routing. In: Integer Programming and Combinatorial Optimization, pp. 393–403. Springer (2014)
24. Pessoa, A., Poggi de Aragão, M., Uchoa, E.: Robust branch-cut-and-price algorithms for vehicle routing problems. In: Golden, B., Raghavan, S., Wasil, E. (eds.) The Vehicle Routing Problem: Latest Advances and New Challenges, pp. 297–325. Springer, Berlin (2008)
25. Pessoa, A., Sadykov, R., Uchoa, E., Vanderbeck, F.: In-out separation and column generation stabilization by dual price smoothing. In: Symposium on Experimental Algorithms, pp. 354–365. Springer (2013)
26. Pessoa, A., Uchoa, E., Poggi de Aragão, M., Rodrigues, R.: Exact algorithm over an arc-time-indexed formulation for parallel machine scheduling problems. Math. Program. Comput. **2**(3–4), 259–290 (2010)
27. Pessoa, A., Uchoa, E., Poggi de Aragão, M.: A robust branch-cut-and-price algorithm for the heterogeneous fleet vehicle routing problem. Networks **54**(4), 167–177 (2009)
28. Poggi de Aragão, M., Uchoa, E.: Integer program reformulation for robust branch-and-cut-and-price. In: Wolsey, L. (ed.) Ann. Math. Program. Rio, pp. 56–61. Búzios, Brazil (2003)
29. Poggi de Aragão, M., Uchoa, E.: New exact algorithms for the capacitated vehicle routing problem. In: Toth, P., Vigo, D. (eds.) Vehicle Routing: Problems, Methods, and Applications, pp. 59–86. SIAM (2014)
30. Righini, G., Salani, M.: Symmetry helps: bounded bi-directional dynamic programming for the elementary shortest path problem with resource constraints. Discrete Optim. **3**(3), 255–273 (2006)
31. Righini, G., Salani, M.: New dynamic programming algorithms for the resource constrained elementary shortest path problem. Networks **51**(3), 155–170 (2008)
32. Røpke, S.: Branching decisions in branch-and-cut-and-price algorithms for vehicle routing problems. Presentation in Column Generation (2012)
33. Santos, H., Toffolo, T., Gomes, R., Ribas, S.: Integer programming techniques for the nurse rostering problem. Ann. Oper. Res. **239**(1), 225–251 (2014)