European Journal of Operational Research 239 (2014) 102–111

Contents lists available at ScienceDirect

European Journal of Operational Research

journal homepage: www.elsevier.com/locate/ejor

Discrete Optimization

Efficient elementary and restricted non-elementary route pricing

Rafael Martinelli [1], Diego Pecin, Marcus Poggi *

Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro (PUC-Rio), Rua Marquês de São Vicente, 225 – RDC 4º andar, Gávea, Rio de Janeiro/RJ 22451-900, Brazil

ARTICLE INFO

Article history:
Received 7 November 2013
Accepted 5 May 2014
Available online 14 May 2014

Keywords:
Routing problems
Column generation
Non-elementary routes
Elementary routes

ABSTRACT

Column generation is involved in the current most efficient approaches to routing problems. Set partitioning formulations model routing problems by considering all possible routes and selecting a subset that visits all customers. These formulations often produce tight lower bounds and require column generation for their pricing step. The bounds in the resulting branch-and-price are tighter when elementary routes are considered, but this approach leads to a more difficult pricing problem. Balancing the pricing with route relaxations has become crucial for the efficiency of the branch-and-price for routing problems. Recently, the *ng*-routes relaxation was proposed as a compromise between elementary and non-elementary routes. The *ng*-routes are non-elementary routes with the restriction that when following a customer, the route is not allowed to visit another customer that was visited before if they belong to a dynamically computed set. The larger the size of these sets, the closer the *ng*-route is to an elementary route. This work presents an efficient pricing algorithm for *ng*-routes and extends this algorithm for elementary routes. Therefore, we address the Shortest Path Problem with Resource Constraint (SPPRC) and the Elementary Shortest Path Problem with Resource Constraint (ESPPRC). The proposed algorithm combines the Decremental State-Space Relaxation technique (DSSR) with completion bounds. We apply this algorithm for the Generalized Vehicle Routing Problem (GVRP) and for the Capacitated Vehicle Routing Problem (CVRP), demonstrating that it is able to price elementary routes for instances up to 200 customers, a result that doubles the size of the ESPPRC instances solved to date.

1. Introduction

Since the work of Christofides, Mingozzi, and Toth (1981) on the Capacitated Vehicle Routing Problem (CVRP), column generation has become a widely applied technique for exactly solving different routing problems. Currently, it is involved in almost all of the current most efficient approaches to routing problems. These approaches use integer and mixed-integer programming formulations with variables associated with the set of all possible routes. These formulations are set partitioning based, and these constraints impose the selection of the route to serve each customer. We refer to this as SPP formulation. The resolution of its linear relaxation requires the use of column generation techniques. The pricing subproblem to be solved is the Elementary Shortest Path Problem with Resource Constraints (ESPPRC).

The ESPPRC is a shortest path problem on a graph where the customers have an amount of resources that are consumed during a visit. The resource constraints require that the total of the resources consumed by any feasible solution does not exceed the existing limits. There may be edges with negative cost, and these edges may generate negative cycles, but because a feasible solution must be an elementary path, revisiting a customer is strictly forbidden.

The ESPPRC is a difficult to solve $\mathcal{NP}$-hard problem (Dror, 1994). In general, the current best performing algorithms have acceptable processing times when the optimal solution is a path with at most fourteen customers. We refer to Di Puglia Pugliese and Guerriero (2012) for a review of the approaches proposed throughout the last three decades. Most of the main ideas for the ESPPRC resolution can be found in their work and in the proceedings of Feillet, Dejax, Gendreau, and Gueguen (2004), Chabrier (2006), Righini and Salani (2006), Righini and Salani (2008) and Boland, Dethridge, and Dumitrescu (2006).

Instead of solving the ESPPRC, the original work of Christofides et al. (1981) solves its relaxation, the Shortest Path Problem with Resource Constraints (SPPRC). This relaxation does allow revisiting a same customer in a route. The resulting non-elementary routes

* Corresponding author. Tel.: +55 21 3527 1500x4339; fax: +55 21 3527 1530.
E-mail addresses: rafael.martinelli@iceb.ufop.br (R. Martinelli), dpecin@inf.puc-rio.br (D. Pecin), poggi@inf.puc-rio.br (M. Poggi).
[1] Present address: Departamento de Computação, Universidade Federal de Ouro Preto (UFOP), Campus Universitário Morro do Cruzeiro, Ouro Preto, MG 35400-000, Brazil.

http://dx.doi.org/10.1016/j.ejor.2014.05.005
0377-2217/

are often called q-routes. However, the resource constraints are the same as the ESPPRC, and therefore, every time a customer is visited, the relevant resource consumption is counted, and the total consumption must still respect the existing limits. A recent survey about the SPPRC can be found in Di Puglia Pugliese and Guerriero (2013).

This relaxation has some interesting properties. First, as distinct from the original problem, it can be solved in pseudo-polynomial time using a dynamic programming algorithm. In addition, even relaxing the elementarity constraint, the SPP bounds found by its linear relaxation are usually strong, especially when there are few customers per route. Furthermore, to strengthen the bounds of the linear relaxation, Christofides et al. (1981) also demonstrated that size two cycles can be forbidden with almost no extra effort.

However, even with the 2-cycle elimination restriction, the linear relaxation is still weak, a behavior that motivated researchers to seek better cycle elimination devices to turn the non-elementary routes closer to elementary routes, without dealing with the whole complexity of the ESPPRC. The work of Irnich and Villeneuve (2006) devised an algorithm that solves the SPPRC by forbidding cycles of an arbitrary size. This algorithm is significantly more complicated, resulting in a complexity that grows factorially with the size of the cycles being forbidden. On account of this, their method quickly becomes impractical. Eliminating cycles of size four or more is already too time consuming compared with the bound improvement obtained. Such behavior was verified in practice by Fukasawa et al. (2006) for the CVRP.

Recently, Baldacci, Mingozzi, and Roberti (2011) proposed a compromise between routes and q-routes: ng-routes. These ng-routes are restricted non-elementary routes built accordingly to customer sets, ng-sets, which are associated with each customer and act like their "memory." Therefore, when a path reaches a given customer, it "forgets" whether another customer was visited if it does not belong to the ng-set of the current customer. Moreover, the further extension is only allowed to customers that are not "remembered". The ng-sets are usually composed of the closest customers and clearly with a larger size set the problem is harder to solve. This is due to the fact that the ng-routes generated are going to be increasingly closer to elementary routes.

Although the SPPRC with ng-routes can be solved in pseudo-polynomial time for a fixed ng-set size, to the best of our knowledge, there is no work that solves this problem for ng-sets larger than 20. This finding is observed because of the exponential-sized data structure used by Baldacci et al. (2011) to speed up the algorithm.

1.1. Contributions

This work aims at efficiently solving the SPPRC with restricted non-elementary routes and the ESPPRC. The first improvement is obtained by adapting the Decremental State-Space Relaxation (DSSR) technique of Righini and Salani (2008) to the SPPRC with ng-routes. This technique was initially proposed for the ESPPRC, where the elementarity restriction is relaxed and the problem is then solved iteratively, rebuilding the restrictions as needed, until the optimal solution is found. The main difference of our algorithm is instead of relaxing the elementarity of the routes, we relax the restriction imposed by the ng-sets.

Next, we accelerate this approach using completion bounds. Because an iteration of the DSSR is a relaxation for its next iteration, the completion bounds estimate lower bounds for completing the paths being built on a given iteration, and, given an upper bound for the optimal solution (which is usually equal to zero for pricing algorithms), it avoids the extension of paths that may exceed the known upper bound on the next DSSR iteration.

These two techniques were already used together in the work of Pecin (2010). In that work, a column generation procedure that uses the ESPPRC as the pricing subproblem was proposed for the CVRP. Using the algorithm of Fukasawa et al. (2006), instances with up to 100 customers could be solved to optimality pricing only elementary routes.

Finally, we demonstrate how our algorithm for the SPPRC with restricted non-elementary routes can be easily extended to generate only elementary routes. We also highlight the two new elements existing in our approach that allow us to double the size of the ESPPRC instances solved thus far.

The proposed algorithms are then applied to the Generalized Vehicle Routing Problem (GVRP), where the customers to be visited are clusters of vertices. Each cluster has an associated demand and can contain one or more vertices. The demand of each cluster must be fully collected in exactly one vertex of the cluster. This problem is a generalization of the CVRP and the Traveling Salesman Problem (TSP), and, as in the classical CVRP, identical vehicles are given, and routes must start and end at the depot and the capacity of the vehicle must not be exceeded. We report experiments demonstrating that our algorithms are able to solve the SPPRC with ng-set sizes up to 64 and the ESPPRC for hard instances of both the GVRP and CVRP. The CVRP instances are solved through reducing them to GVRP instances. The results of the column generation algorithm also provide a clear idea of the gains in the lower bounds comparing the SPPRC with different ng-set sizes and also with the ESPPRC, as well as the time required for computing them. In addition, several new best lower bounds are identified for the GVRP, especially for large instances.

This paper is organized as follows. Section 2 presents the ESPPRC and explains the required mathematical notation. The ng-route relaxation is described in Section 3. In Section 4, we explain the techniques used to solve the ng-route relaxation. In Section 5, we demonstrate how our algorithm can be used to obtain only elementary routes, and we highlight the main elements that allowed us to build a very efficient method for solving the ESPPRC. Section 6 presents the Generalized Vehicle Routing Problem formally. Section 7 reports the computational results for both the GVRP and CVRP. Finally, Section 8 presents some conclusions.

2. Elementary shortest path problem with resource constraints

Let $G = (V, A)$ be a graph with arc set A and vertex set V, which is composed of the set of customers $\mathcal{C}$ plus a source vertex s and a destination vertex t, and let $\mathcal{R}$ be a set of resources. For each arc $(i,j) \in A$, let c_{ij} be the cost of the arc and w_{ij}^r be the consumption of the edge, for each $r \in \mathcal{R}$. For each pair $i \in \mathcal{C}$ and $r \in \mathcal{R}$ let a_i^r and b_i^r be two non-negative values, such that the total resource consumption along a path from s to i must belong to the interval $[a_i^r, b_i^r]$. The ESPPRC aims to find a minimum cost elementary path from s to t that satisfies all resource constraints.

The resources constraints can model different types of restrictions. For instance, most vehicle routing problems consider that the vehicles have a known capacity, and this capacity cannot be exceeded in a single route. Other problems have time windows, which require the route to visit a customer in a given interval of time. Moreover, one can also view the elementarity constraint as resource constraints, where each customer defines a binary resource and when a route visits a customer, it consumes all of the associated resource.

In this work, we deal only with the capacity constraint, in addition to the obvious elementarity constraint. Thus, the customer set has an associated demand function $d : \mathcal{C} \to \mathbb{Z}$, and there is a global capacity limit Q, which no feasible solution may exceed. Because we apply our algorithm for routing problems, we can consider

the source and the destination vertices as a single vertex called the depot and labeled 0. Therefore, the solution of the problem is a route instead of a path. This is straightforward because most of the applications for the ESPPRC involve a pricing routine embedded in a column generation scheme that solves some type of vehicle routing problem.

3. The *ng*-route relaxation

Recently, the *ng*-route relaxation was introduced in the work of Baldacci et al. (2011) for the CVRP and the CVRP with Time Windows (CVRPTW), and it was later extended to the GVRP by Bartolini, Cordeau, and Laporte (2011), where it was used to solve transformed CARP instances. This new relaxation aims at obtaining a better compromise between efficiently priced non-elementary routes and obtaining good lower bounds.

For each customer $i \in \mathcal{C}$, let $N_i \subseteq \mathcal{C}$ be a subset of customers that have a relationship with i. A possible representation for this relationship can be a neighborhood relationship, i.e., N_i contains the nearest customers of i, including i. These sets are called *ng*-sets, and they contain the customers that customer i is able to "remember." For instance, when a path P is being built, by the time it arrives at customer i, it has a set $\Pi(P)$ that represents its "memory" thus far. If customer i belongs to set $\Pi(P)$, the extension is forbidden. On the other hand, if i does not belong to set $\Pi(P)$, the extension is allowed and set $\Pi(P)$ is updated to "forget" the customers that customer i is not able to "remember," i.e., the customers that do not belong to N_i. It is clear that if a customer is "forgotten," it can be used to form a cycle in future extensions of path P. At this point, we can conclude that the size of the ng-sets is an important factor in the quality of solutions because the larger the ng-sets are, the greater the smallest cycles that can appear in a path. The size of each ng-set N_i is limited by $\Delta(N_i)$, which is a parameter defined *a priori*. Obviously, this size also changes the pricing complexity.

Let $P = (0, i_1, \ldots, i_{p-1}, i_p)$ be a path starting at the depot, visiting a sequence of customers and ending at customer i_p and $\mathcal{C}(P)$ be the set of customers visited by path P. The function $\Pi(P)$ of prohibited extensions (the "memory") of path P can be defined as follows.

$$\Pi(P) = \left\{ i_k \in \mathcal{C}(P) \setminus \{i_p\} : i_k \in \bigcap_{s=k+1}^{p} N_{i_s} \right\} \cup \{i_p\}. \tag{1}$$

Given $d(P) = \sum_{i \in \mathcal{C}(P)} d_i$ as the total demand serviced by path P and $c(P)$ as the total cost of path P, let $\mathcal{L}(P) = (i_p, d(P), \Pi(P), c(P))$ be a label associated with a path P, which ends at customer i_p. We say that a label $\mathcal{L}(P)$ can be extended to a customer i_{p+1} if $i_{p+1} \notin \Pi(P)$ and $d(P) + d_{i_{p+1}} \leqslant Q$. After the extension, the customer i_{p+1} becomes the last customer of a new path $P' = (0, \ldots, i_p, i_{p+1})$ and a new label $\mathcal{L}(P')$ can be obtained from the label $\mathcal{L}(P)$ by the following operations:

$$\mathcal{L}(P') = (i_{p+1}, d(P) + d_{i_{p+1}}, \Pi(P) \cap N_{i_{p+1}} \cup \{i_{p+1}\}, c(P) + c_{i_p i_{p+1}}). \tag{2}$$

These labels are computed using a forward dynamic programming algorithm and, in contrast to the q-route relaxation, it does not result in a pseudo-polynomial complexity. This algorithm is exponential on the size of $\Delta(N_i)$, remaining pseudo-polynomial for fixed $\Delta(N_i)$. Furthermore, its efficiency depends on the use of some techniques to speed up its execution.

To reduce the number of possible paths, a dominance rule is incorporated into the algorithm. Given the labels of two paths $\mathcal{L}(P_1)$ and $\mathcal{L}(P_2)$, path P_1 dominates path P_2 if and only if every possible extension from P_2 can be done from P_1 with a lower or equal total cost. For this to be true, the following three conditions must hold:

(i) $d(P_1) \leqslant d(P_2)$,
(ii) $c(P_1) \leqslant c(P_2)$ and
(iii) $\Pi(P_1) \subseteq \Pi(P_2)$.

4. An efficient *ng*-route relaxation

As discussed earlier, a basic *ng*-route relaxation implementation does not allow the use of large ng-sets, a limitation which weakens the quality of the lower bounds found. To address this issue, we provide an efficient implementation, adapting the Decremental State Space Relaxation for the *ng*-route relaxation. This technique was introduced by Righini and Salani (2008) to solve the ESPPRC. The original version of the algorithm helps reduce the number of labels to be managed during the dynamic programming algorithm that builds elementary paths. First, it relaxes the elementarity of the paths and, at each iteration, identifies which customers are being repeated on the best path found and then prohibits the repetition of these customers in subsequent iterations.

The main difference of our algorithm is that instead of relaxing the elementarity of the paths, the new algorithm relaxes the *ng*-set of each customer, therefore relaxing the *ng*-route restrictions.

4.1. Basic exact dynamic programming algorithm

We start by creating a $(Q+1) \times |\mathcal{C}|$ dynamic programming matrix $\mathcal{M}$, where each entry $\mathcal{M}(d, i)$ is a bucket containing labels representing paths that start at the depot and end at customer i with total demand equal to d. At first, we set $\mathcal{M}(d_i, i)$ with a single label $\mathcal{L}_i = (i, d_i, \{i\}, c_{0i}), \forall i \in \mathcal{C}$, and all other entries with no label. Next, a forward dynamic programming is used to fill the matrix $\mathcal{M}$, running from $d = 1$ up to $d = Q$.

Algorithm 1 presents the pseudocode of our basic dynamic programming procedure. When processing the bucket $\mathcal{M}(d, i)$, the algorithm iterates through all labels $\mathcal{L}(P_j)$ belonging to $\mathcal{M}(d - d_i, j)$, for all customers $j \in \mathcal{C}$, such that $d - d_i > 0$. As the basic *ng*-route relaxation algorithm, the extension from $\mathcal{L}(P_j)$ to i can only be performed if $i \notin \Pi(P_j)$. If this condition holds, a new label, say $\mathcal{L}(P_i)$, is then created and it must be stored in the bucket $\mathcal{M}(d, i)$. Therefore, this is the right time to check for the dominance rule, which can be verified for all the labels in $\mathcal{M}(d', i), \forall d' \leqslant d$. Surprisingly, we have found that the algorithm runs faster if the dominance rule is tested only for labels of the same bucket, i.e., for the labels from inside $\mathcal{M}(d, i)$. This comes from the fact that labels associated with paths using less capacity are unlikely to dominate others using higher capacity.

In addition, Baldacci et al. (2011) described another way to improve the dominance rule. When the paths of the bucket $\mathcal{M}(d, i)$ are being computed, the algorithm uses a dominance list associated with the customer i, which stores the best costs for every possible configuration of $\Pi(P)$ and $d' < d$, to do the dominance. This way, it is faster to check the list than to iterate through the dynamic programming matrix for each $d' < d$. We do not use this technique because it is not scalable, as the size of the dominance list is exponential in the value of $\Delta(N_i)$, reaching its size limit when $\Delta(N_i) \approx 13$.

Algorithm 1 also presupposes the existence of procedure *buildRoutes*, which takes matrix $\mathcal{M}$ as a parameter and extracts the best routes from it.

Algorithm 1. Basic Dynamic Programming *ng*-Route Algorithm

procedure DynamicProgramming($\mathcal{M}, \mathbf{N}$)
  input: matrix $\mathcal{M}$ and *ng*-sets $N_i \subseteq \mathcal{C}, \forall i \in \mathcal{C}$.
  output: the best *ng*-routes with respect to *ng*-sets N_i.
  $\mathcal{M}(d,i) \leftarrow \emptyset, \forall i \in \mathcal{C}, d \in \{0,\ldots,Q\}$
  $\mathcal{M}(d_i,i) \leftarrow \{(i,d_i,\{i\},c_{0i})\}, \forall i \in \mathcal{C}$
  for $d := 1,\ldots,Q$ **do**
    for all $i \in \mathcal{C}$ **do**
      if $d - d_i > 0$ **then**
        for all $j \in \mathcal{C}$ **do**
          for all $\mathcal{L}(P_j) \in \mathcal{M}(d-d_i,j)$ **do**
            if $i \notin \Pi(P_j)$ **then**
              $\mathcal{L}(P_i) \leftarrow (i, d, \Pi(P_j) \cap N_i \cup \{i\}, c(P_j) + c_{ji})$
              insertLabel $\leftarrow$ **true**
              for all $\mathcal{L}(P'_i) \in \mathcal{M}(d,i)$ **do**
                if $\mathcal{L}(P_i)$ dominates $\mathcal{L}(P'_i)$ **then**
                  delete $\mathcal{L}(P'_i)$
                else if $\mathcal{L}(P'_i)$ dominates $\mathcal{L}(P_i)$ **then**
                  insertLabel $\leftarrow$ **false**
                  break
              if *insertLabel* **then**
                $\mathcal{M}(d,i) \leftarrow \mathcal{M}(d,i) \cup \mathcal{L}(P_i)$
  return buildRoutes ($\mathcal{M}$)

4.2. Improved exact dynamic programming algorithm

4.2.1. Decremental state-space relaxation

The adapted DSSR is an iterative algorithm and it works by relaxing the state space of the original *ng*-sets N_i. At each iteration k, the algorithm uses the subsets $\Gamma_i^k \subseteq N_i$ as a replacement for N_i. These subsets Γ_i^k take the role of N_i in the definition of the function Π, described in Eq. (1), and in the creation of new labels, as shown in Eq. (2). Initially, the algorithm sets $\Gamma_i^0, \forall i \in \mathcal{C}$, as an empty set and executes the basic dynamic programming Algorithm 1. As the best routes found by this dynamic programming are not necessarily *ng*-routes w.r.t. the original *ng*-sets N_i, they cannot be considered as the result of the pricing without verifying their feasibility. This test is performed and the Γ_i^k subsets are updated if necessary, as described hereafter. If at the end of iteration k some subset Γ_i^k is updated, the dynamic programming algorithm is executed again with new subsets Γ_i^{k+1}.

Let a cycle of customers be defined as a sub-path $H = (i,\ldots,j)$, where $i = j$, and let $\mathcal{H}(P)$ be the set of all cycles of customers in the path P. To evaluate if the best route R_k^* found in the end of iteration k is an *ng*-route, the algorithm must check if there is no cycle $H \in \mathcal{H}(R_k^*)$ which would not be allowed to be created if the original *ng*-sets N_i were being used. This happens only when the first customer j of cycle H (we also refer to it as the repeated customer of the cycle) is in all $N_l, \forall l \in \mathcal{C}(H)$, i.e., customer j is not "forgotten" by any other customer of the cycle. In this case, we designate such a cycle H as a *forbidden* cycle w.r.t the original *ng*-sets N_i. If any such cycle H is found, we add the repeated customer j to subsets $\Gamma_l^{k+1}, \forall l \in \mathcal{C}(H)$. This prohibits cycle H from appearing in any path obtained in the next iterations. More than that, this prohibits any cycle $H' = (j,\ldots,j)$ in which $\mathcal{C}(H') \subseteq \mathcal{C}(H)$ from appearing in the next iterations. On the other hand, if no such forbidden cycle is found at the end of iteration k, then the best route R_k^* is an *ng*-route and the algorithm stops.

Algorithm 2 reports the DSSR procedure. The input parameter of the algorithm is the original *ng*-sets $N_i, \forall i \in \mathcal{C}$. The procedures with self-explanatory names *selectBestRoute*, *isNGRoute* and *updateNGSets* are responsible, respectively, for extracting the best route of a set of routes, determining if a given route is a valid *ng*-route with respect to *ng*-sets N_i and to update the subsets Γ_i^k to the next iteration. Algorithm 2 also uses the procedure *dynamicProgramming*, which is presented in Algorithm 1, to obtain *ng*-routes with respect to subsets Γ_i^k passed as input parameters.

Algorithm 2. Pure DSSR *ng*-Route Algorithm

procedure DSSR($\mathcal{M}, \mathbf{N}$)
  input: matrix $\mathcal{M}$ and *ng*-sets $N_i \subseteq \mathcal{C}, \forall i \in \mathcal{C}$.
  output: the best *ng*-routes with respect to *ng*-sets N_i.
  $\Gamma_i \leftarrow \emptyset, \forall i \in \mathcal{C}, ng \leftarrow$ **false**, $k \leftarrow 0$
  while not *ng* **do**
    $\mathcal{R} \leftarrow$ dynamicProgramming ($\mathcal{M}, \Gamma$)
    $R_k^* \leftarrow$ selectBestRoute ($\mathcal{R}$)
    if isNGRoute(R_k^*) **then**
      $ng \leftarrow$ **true**
    else
      updateNGSets$(\mathbf{N}, R_k^*)$
    $k \leftarrow k + 1$
  return $\mathcal{R}$
procedure isNGRoute(R)
  for all $H = (v,\ldots,v) \in \mathcal{H}(R)$ **do**
    forbiddenCycle $\leftarrow$ **true**
    for all $l \in \mathcal{C}(H)$ **do**
      if $v \notin N_l$ **then**
        forbiddenCycle $\leftarrow$ **false**
        break
    if *forbiddenCycle* **then**
      return false
  return true
procedure updateNGSets($\mathbf{N}, R$)
  for all $H = (v,\ldots,v) \in \mathcal{H}(R)$ **do**
    forbiddenCycle $\leftarrow$ **true**
    for all $l \in \mathcal{C}(H)$ **do**
      if $v \notin N_l$ **then**
        forbiddenCycle $\leftarrow$ **false**
        break
    if *forbiddenCycle* **then**
      for all $l \in \mathcal{C}(H)$ **do** $\Gamma_l \leftarrow \Gamma_l \cup \{v\}$

It is noteworthy to mention that if the best route found is indeed an *ng*-route, the algorithm can stop and return only this route. However, if the objective is to find a set of feasible solutions, the algorithm can also return this route together with any other route certified to be an *ng*-route. Furthermore, if the best route is not an *ng*-route and one needs to find *any* feasible solution, any route that is indeed an *ng*-route can be returned, even if the best route is not feasible. In this case, we consider it as being a heuristic run of the algorithm, not an exact one. This method is useful to quickly price routes on intermediate iterations of column generation algorithms, where there is no need to generate the optimal solution.

4.2.2. Completion Bounds

To further speed up the algorithm, we calculate completion bounds during the DSSR in a similar manner as performed by Pecin (2010) for the elementary route. At the end of iteration k, the completion bounds are calculated for each customer $i \in \mathcal{C}$ with every capacity $d \in \{0,\ldots,Q\}$, and then used at iteration $k+1$. As mentioned before, the completion bounds are used to estimate a lower bound on the value of a route during its creation, thus discarding any route that would not lead to a negative reduced

cost. Given $T_k^*(d,i)$, the value of the best path which starts at customer i and ends at the depot with total capacity exactly d, the completion bounds $\widehat{T}_k(d,i)$ are calculated as shown in (3) and represent the value of the best path that starts at customer i and ends at the depot with total capacity less than or equal d.

$$\widehat{T}_k(d,i) = \min_{d' \leqslant d}\{T_k^*(d',i)\}. \tag{3}$$

It is important to observe that if the corresponding problem is represented by means of an undirected graph, $T_k^*(d,i)$ can be obtained directly from the dynamic programming matrix. This is true because the value of the best path that starts at the depot and ends at customer i with total capacity exactly d has the same value as $T_k^*(d,i)$, as shown by Baldacci et al. (2011). The reason for this is that if the best forward path contains a cycle, then the best backward path may also contain this cycle and vice versa because at least one costumer belonging to this cycle must "forget" the customer that repeats. On the other hand, if the problem is represented using a directed graph, to obtain these values, the direction of the edges has to be reversed, as also shown by Baldacci et al. (2011). In this case, the last iteration of the DSSR algorithm has to be executed again before the calculation. This occurs because when a route is traversed in the opposite direction on an asymmetric graph, it does not generate the same cost.

After calculating the completion bounds at the end of iteration k, they can be used at iteration $k+1$ to avoid the extension of a given label $\mathcal{L}(P) = (j, d(P), \Pi(P), c(P))$ to a customer i if the following conditions hold:

$$c(P) + c_{ji} + \widehat{T}_k(Q - d(P), i) \geqslant 0. \tag{4}$$

This equation calculates a lower bound on the value of the reduced cost of any route the label $\mathcal{L}(P)$ can generate, because $\widehat{T}_k(Q-d(P),i)$ is a lower bound on the value of the best path, which would close path P after it is extended to customer i. Obviously, if the value of Eq. (4) is greater or equal than zero, the label $\mathcal{L}(P)$ cannot generate any route with a negative reduced cost, and therefore, it can be discarded.

It is interesting to highlight that the completion bounds becomes stronger along the iterations of the DSSR, because iteration k is a relaxation of iteration $k+1$, i.e., given that $\Gamma_i^k \subseteq \Gamma_i^{k+1}, \forall i \in \mathcal{C}$. These bounds can be used for other important parts of a branch-cut-and-price algorithm, such as route enumeration and variable fixing.

Algorithm 3 reports the pseudocode for pricing *ng*-routes with DSSR and completions bounds. The procedures *selectBestRoute*, *isNGRoute* and *updateNGSets* have the same meaning as before and *generateCompletionBounds* is a new procedure that calculates the bounds as shown in (3). The procedure *BoundedDynamicProgramming* is a slight modification of procedure *dynamicProgramming* of Algorithm 1, that targets only inclusion of the completion bounds calculated as shown in (4).

Algorithm 3. DSSR with Completion Bounds ng-Route Algorithm

```
1: procedure DSSRwithBounds(M, N)
2:   input: matrix M and ng-sets N_i ⊆ C, ∀i ∈ C.
3:   output: the best ng-routes with respect to ng-sets N_i.
4:   Γ_i ← ∅, ∀i ∈ C, ng ← false, k ← 0
5:   T̂(d,i) ← −∞, ∀i ∈ C, d ∈ {0,...,Q}
6:   while not ng do
7:     R ← BoundedDynamicProgramming (M, Γ, T̂)
8:     R*_k ← selectBestRoute (R)
9:     if isNGRoute(R*_k) then
10:      ng ← true
11:    else
12:      updateNGSets(N, R*_k)
13:      generateCompletionBounds (M, T̂)
14:    k ← k + 1
15:  return R
16: procedure BoundedDynamicProgramming(M, Γ, T̂)
17:  M(d,i) ← ∅, ∀i ∈ C, d ∈ {0,...,Q}
18:  M(d_i,i) ← {(i, d_i, {i}, c̄_0i)}, ∀i ∈ C
19:  for d := 1,...,Q do
20:    for all i ∈ C do
21:      if d − d_i > 0 then
22:        for all j ∈ C do
23:          for all L(P_j) ∈ M(d − d_i, j) do
24:            if i ∉ Π(P_j) then
25:              if checkCompletionBound (T̂, L(P_j), i) then
26:                L(P_i) ← (i, d, Π(P_j) ∩ N_i ∪ {i}, c̄(P_j) + c̄_ji)
27:                insertLabel ← true
28:                for all L(P'_i) ∈ M(d,i) do
29:                  if L(P_i) dominates L(P'_i) then
30:                    delete L(P'_i)
31:                  else if L(P'_i) dominates L(P_i) then
32:                    insertLabel ← false
33:                    break
34:                if insertLabel then
35:                  M(d,i) ← M(d,i) ∪ L(P_i)
36: return buildRoutes (M)
37: procedure generateCompletionBounds(M, T̂)
38:  T̂(d,i) ← ∞, ∀i ∈ C, d ∈ {0,...,Q}
39:  T̂(0,0) ← 0
40:  for i ∈ C do
41:    for d := 1,...,Q do
42:      T̂(d,i) ← min(M(d,i))
43:      if T̂(d − 1,i) < T̂(d,i) then
44:        T̂(d,i) ← T̂(d − 1,i)
```

4.3. Heuristic pricing

Even with the improvements described in Section 4.2, the exact *ng*-route pricing still requires a long time to be executed. Because of this, a simple but effective heuristic was developed to quickly price a large initial set of routes with negative reduced cost. It was based on the heuristic pricing done for the elementary route pricing by Pecin (2010). The purpose of this heuristic is to reduce the number of calls to the exact *ng*-route pricing. Therefore, the heuristic *ng*-route pricing is used as a hot-start for the exact *ng*-route pricing.

The heuristic closely resembles the *q*-route pricing without eliminating any cycle. The main difference between the pricing algorithms is that when extending one path, the heuristic *ng*-route pricing respects the ng-sets N_i. Its data structure is also a $(Q+1) \times |\mathcal{C}|$ matrix, and each entry consists of just one label. For each customer and each capacity, this label is chosen as the best one with respect to the reduced costs. In addition, as the ng-sets N_i must be respected, each label of the dynamic programming matrix must contain the Π sets for each customer and capacity.

Notice that unlike the exact algorithms, the heuristic algorithms use neither the dominance rules nor the speed-up techniques

(DSSR and completion bounds) described in Section 4. Nevertheless, they are responsible for obtaining approximately 90% of the routes during the column generation. Moreover, it is straightforward to verify that the resulting complexity of the algorithms is $\mathcal{O}(n^2Q)$.

5. Achieving elementarity

To achieve elementarity, we change the definition of the Γ_i^k sets used in the DSSR algorithm. Instead of being a subset of the N_i sets, which are defined *a priori* and have a fixed size $\Delta(N_i)$, the algorithm uses $\Gamma_i^k \subseteq \mathcal{C}$. In this case, it will allow the insertion of any customer of the instance. In addition, the algorithm is quite the same the one described in Section 4. The only difference is in the DSSR feasibility test, described as follows.

The algorithm starts with empties $\Gamma_l^0, \forall l \in \mathcal{C}$. At the end of each iteration k, it identifies all cycles on the best solution, and the repeated customer of each cycle $H \in \mathcal{H}(R_k^*)$ is inserted on subsets $\Gamma_l^{k+1}, \forall l \in \mathcal{C}(H)$. Thus, if cycle $H = (i, \ldots, j), i = j$, is identified at the end of iteration k (that is, if cycle H belongs to $\mathcal{H}(R_k^*)$), the next DSSR iterations will not generate any path with a cycle $H' = (i, \ldots, j)$, in which $\mathcal{C}(H') \subseteq \mathcal{C}(H)$. Note, however, that it is still possible to obtain a path that visits customer j more than once at iteration $k+1$, as this customer is not present in all subsets Γ_l^{k+1}. The algorithm stops when the best route does not contain any cycle, i.e. it is an elementary route, or its reduced cost is non-negative.

The way the algorithm increases the state space along the DSSR iterations resembles the method used by Righini and Salani (2008), however the differences control better the growth of the number of labels, turning the algorithm capable of dealing with larger instances. The DSSR of Righini and Salani (2008) is performed prohibiting the customers which repeat on the best route R_k^* from repeating again in subsequent iterations until an elementary route is found. This is equivalent to inserting each repeated customer of each cycle $H \in \mathcal{H}(R_k^*)$ in all subsets Γ_i^{k+1}, rather than just considering this inclusion in subsets $\Gamma_l^{k+1}, \forall l \in \mathcal{C}(H)$.

We noted in computational experiments that this more aggressive manner of increase of the state space is quite dangerous because the whole algorithm fails if the number of labels to be treated in the dynamic programming becomes critical. In contrast, the size of the largest subset Γ_i^k hardly exceeded 20 in our algorithm, even for instances with 200 customers.

Another important difference from our approach and that of Righini and Salani (2008) is the way we use the DSSR to calculate completion bounds at each iteration to accelerate the subsequent DSSR iterations. Computational results reveal that the use of completion bounds allow us to solve the column generation for CVRP instances that would not be solved in an acceptable time if they were not used.

Furthermore, because the algorithm presented here aims to find only elementary routes, instead of using the heuristic pricing presented in Section 4.3, we implemented a heuristic pricing like the one described in Pecin (2010).

6. Application to the generalized vehicle routing problem

The Generalized Vehicle Routing Problem (GVRP) can be defined as follows. Let $G = (V, E)$ be a graph with vertex set V and edge set E. There is a special vertex 0 called the depot. The vertices are partitioned into disjoint sets, called clusters, $C = \{C_0, C_1, \ldots, C_t\}$, where $C_0 = \{0\}$ contains only the depot. Given the cluster index set $M = \{0, 1, \ldots, t\}$, let $\mu(i) \in M$ be defined, for each vertex $i \in V$, as the index of the cluster which contains i. There exists a demand function $d : M \to \mathbb{Z}^+$ associated with all clusters, in which the depot has demand $d_0 = 0$. These demands are to be serviced by a set $\mathcal{K}$ of identical vehicles with capacity Q, located at the depot. The edge set $E = \{\{i,j\} | i,j \in V, \mu(i) \neq \mu(j)\}$ contains the edges between all pairs of vertices from different clusters. Associated with these edges, there exists a traversal cost function $c : E \to \mathbb{Z}_0^+$. Let $\mathcal{R}$ be the set of all possible closed routes starting and ending at the depot. The objective of the GVRP is to select a subset of k routes from $\mathcal{R}$ that: (i) minimizes the total traversal cost (ii) ensures that the demand from every cluster is serviced by a single vehicle on a single vertex from each cluster and (iii) ensures that the total demand serviced by each route does not exceed the vehicle capacity Q.

The GVRP is a generalization of the Capacitated Vehicle Routing Problem (CVRP) and the Generalized Traveling Salesman Problem (GTSP). When all the clusters contain only one vertex, it is simply the CVRP. Similarly, when there is only one vehicle, it is simply the GTSP. It is clear that when both conditions are true, it is simply the Traveling Salesman Problem (TSP). In the view of this, it is easy to see that any solution for the GVRP can be directly used to solve these problems.

This problem is strongly $\mathcal{NP}$-hard and has gained attention in the literature in recent years. To the best of our knowledge, the first published work to deal with this problem is Ghiani and Improta (2000), where a transformation to the Capacitated Arc Routing Problem (CARP) is presented to use the existing algorithms for the CARP. One instance was proposed and solved using this approach.

Since then, few works have been published on the GVRP. Recently, the work of Bektaş, Erdoğan, and Røpke (2011) has proposed four formulations for the GVRP. After extensive experiments with these formulations, a branch-and-cut algorithm was devised using one of them, an undirected formulation with an exponential number of constraints. The reader may consult this paper for further details on these formulations.

7. Experimental results

For the computational experiments, all algorithms were implemented in C++ using Microsoft Visual C++ 2010 Express and IBM ILOG CPLEX Optimizer 12.5 for solving the formulations. The experiments were conducted on an Intel Core i7-3960X 3.30 GHz with 64 GB RAM running Linux Ubuntu Server 12.04 LTS and are divided into two parts. First, we compare the three exact pricing algorithms described in this paper, that is, the basic dynamic programming, the pure DSSR and DSSR with completion bounds. This is enough to conclude that the latter is the best algorithm, and this evaluation is conducted by running each algorithm inside a column generation schema for some classical CVRP instances. All three algorithms were tested using different values of $\Delta(N_i)$, allowing us to analyze the scalability of each one when the state space relaxation is increased. Second, using our best algorithm, we price elementary and restricted non-elementary routes for both GVRP and CVRP instances. To improve the lower bounds and also demonstrate that our algorithm still works well when *robust* cuts (see this terminology in work of Poggi de Aragão & Uchoa (2003)) are added in the SPP formulation, we included capacity inequalities and strengthened comb, both described in detail in the work of Lysgaard, Letchford, and Eglese (2004). We separated and added these cuts in a similar manner as done by Fukasawa et al. (2006) for the CVRP.

For all tests, the column generation starts by calling the heuristic pricing at each iteration. The heuristic pricing returns the best 20 routes with negative reduced costs. If the heuristic is no longer capable of finding routes with negative reduced cost, the column generation algorithm calls the exact pricing. If the latter succeeds

Table 1
Results for column generation with NG = 8 and NG = 16 for selected CVRP instances.

Ins	OPT	NG = 8				NG = 16			
		LB	T1	T2	T3	LB	T1	T2	T3
A-n62-k8	1288	1250.24	1.0	2.0	1.0	1254.83	4.8	4.7	1.2
A-n63-k10	1314	1286.58	0.5	1.7	0.5	1286.81	3.2	8.9	0.8
A-n64-k9	1401	1368.24	0.8	1.7	0.8	1374.49	3.6	5.4	1.2
A-n69-k9	1159	1129.97	1.0	2.1	0.9	1131.33	6.3	8.1	1.1
A-n80-k10	1763	1729.81	1.7	3.6	1.7	1731.45	5.9	8.4	1.9
B-n50-k8	1312	1266.45	0.5	0.9	0.4	1266.63	4.3	2.5	0.5
B-n68-k9	1275	1198.20	1.0	2.5	0.7	1203.78	36.6	21.2	1.5
B-n78-k10	1221	1166.73	1.7	4.6	1.5	1167.46	47.2	19.3	1.7
E-n51-k5	521	517.14	0.7	1.2	0.5	517.14	5.0	4.4	0.7
E-n76-k7	682	664.20	2.9	5.7	2.6	664.82	10.2	19.2	2.8
E-n76-k8	735	717.82	1.8	3.9	1.6	718.74	10.6	12.2	2.2
E-n76-k10	830	811.77	1.1	2.5	1.0	811.87	2.3	4.2	1.0
E-n76-k14	1021	1001.85	0.5	1.6	0.5	1002.77	0.8	2.3	0.5
E-n101-k8	815	789.37	7.3	14.3	6.9	790.71	32.9	40.8	7.9
E-n101-k14	1067	1047.20	1.9	4.8	1.9	1048.45	7.1	9.0	2.4
F-n135-k7	1162	1134.79	2055.2	–	3818.5	1135.90	–	–	3925.1
M-n121-k7	1034	1026.37	27.0	69.1	28.3	1029.21	–	–	382.1
M-n151-k12	1015	995.73	19.8	41.8	22.1	996.64	68.1	100.3	26.4
M-n200-k16	1274	1250.23	38.5	102.6	61.3	1250.87	134.3	255.9	61.2
M-n200-k17	1275	1252.52	38.9	94.2	53.8	1252.87	129.5	193.6	58.3
P-n50-k8	631	614.64	0.2	0.6	0.2	615.55	0.5	1.4	0.2
P-n70-k10	827	809.29	0.5	1.6	0.5	810.61	2.4	4.0	0.7

Table 2
Results for column generation with NG = 32 and NG = 64 for selected CVRP instances.

Ins	OPT	NG = 32				NG = 64			
		LB	T1	T2	T3	LB	T1	T2	T3
A-n62-k8	1288	1254.83	4177.1	27.8	1.5	1254.83	–	1888.0	1.8
A-n63-k10	1314	1286.83	180.9	142.4	0.8	1286.83	–	–	0.9
A-n64-k9	1401	1376.90	177.9	48.4	1.0	1376.90	–	1653.9	1.6
A-n69-k9	1159	1131.34	2002.6	361.6	1.1	1131.34	–	–	1.8
A-n80-k10	1763	1731.58	720.9	36.2	2.2	1731.58	–	6768.7	3.6
B-n50-k8	1312	1266.64	1521.4	8.9	0.6	1266.64	–	411.2	0.6
B-n68-k9	1275	1204.00	–	606.0	1.5	1204.00	–	–	2.2
B-n78-k10	1221	1167.52	–	281.1	1.8	1167.52	–	1564.7	2.9
E-n51-k5	521	517.14	1988.1	100.5	0.8	517.14	–	–	0.8
E-n76-k7	682	665.59	–	494.6	4.1	665.59	–	–	4.6
E-n76-k8	735	718.78	2795.3	239.6	2.5	718.78	–	–	2.7
E-n76-k10	830	812.48	715.6	26.6	1.3	812.48	–	142.4	1.5
E-n76-k14	1021	1002.77	16.8	3.8	0.6	1002.77	–	7.2	0.7
E-n101-k8	815	790.99	–	2462.0	9.1	790.99	–	–	10.9
E-n101-k14	1067	1050.42	726.0	35.0	2.5	1050.42	–	381.7	2.8
F-n135-k7	1162	1136.73	–	–	–	–	–	–	–
M-n121-k7	1034	1029.75	–	–	–	1029.79	–	–	1815.0
M-n151-k12	1015	997.28	–	1076.9	31.3	997.43	–	–	37.5
M-n200-k16	1274	1251.36	–	2449.2	62.9	1252.05	–	–	99.4
M-n200-k17	1275	1253.43	–	2950.6	67.1	1254.01	–	–	87.8
P-n50-k8	631	615.55	24.6	4.1	0.2	615.55	–	17.0	0.3
P-n70-k10	827	810.94	163.2	11.9	0.7	810.94	–	31.7	1.0

in obtaining at least one route with negative reduced cost, the column generation procedure restarts by calling the heuristic pricing. Otherwise, the column generation stops and the current value is returned as a lower bound. This procedure is stopped prematurely if the time limit of two hours is exceeded.

7.1. Problem instances

For the GVRP, we applied our algorithms to the instance datasets recently generated by Bektaş et al. (2011). These instance datasets are derived from the CVRP instance datasets A, B, P and M. The transformation is performed using a method similar to that of Fischetti, Salazar-González, and Toth (1997), which transforms TSP instances into GTSP instances. The number of clusters is $t = \lceil n/\theta \rceil$, where θ is a parameter defined *a priori*. For each original CVRP instance dataset, two new instance datasets were created, using $\theta = 2$ and $\theta = 3$, resulting in 158 GVRP instances. All lower and upper bounds shown in the tables were taken from the work of Bektaş et al. (2011).

The name of the GVRP instances follows the general convention of the CVRP instance, although slightly modified to include additional parameters used. The general format is *X*-n*Y*-k*Z*-CΩ-VΦ, where *X* corresponds to the type of the instance, *Y* refers to the number of vertices, *Z* corresponds to the number of vehicles in the original CVRP instance, Ω is the number of clusters and Φ is the number of vehicles in the GVRP instance. For all problem instances, we calculate the cost matrix using Euclidean distance rounded to the nearest integer value.

For the CVRP, we used just a representative set extracted from the classical instance datasets A, B, E, P and M. All optimal values

Table 3
Results for column generation with cut separation for selected CVRP instances.

Ins	OPT	NG = 8		NG = 16		NG = 32		NG = 64	
		LB	Time	LB	Time	LB	Time	LB	Time
A-n62-k8	1288	1280.75	2.3	1281.99	2.8	1282.08	2.7	1282.09	3.3
A-n63-k10	1314	1302.42	1.0	1303.85	1.1	1303.89	1.2	1303.97	1.7
A-n64-k9	1401	1387.80	1.7	1389.40	1.5	1389.97	1.7	1390.02	2.2
A-n69-k9	1159	1143.80	1.3	1145.22	1.3	1145.38	1.8	1145.42	2.4
A-n80-k10	1763	1756.46	3.4	1756.97	3.9	1756.90	3.9	1756.80	5.4
B-n50-k8	1312	1303.48	0.7	1303.62	0.9	1303.76	0.9	1303.77	1.0
B-n68-k9	1275	1263.75	2.9	1263.92	3.9	1263.92	3.3	1263.93	4.1
B-n78-k10	1221	1217.06	3.7	1217.38	3.1	1217.32	4.1	1217.62	5.7
E-n51-k5	521	519.26	1.1	519.45	1.2	519.52	1.4	519.49	1.5
E-n76-k7	682	671.10	3.9	671.32	4.4	671.94	6.0	671.94	6.2
E-n76-k8	735	726.98	4.0	726.99	3.4	727.18	4.0	727.37	4.4
E-n76-k10	830	817.34	1.7	817.42	1.8	818.01	1.8	818.01	2.3
E-n76-k14	1021	1006.94	0.7	1007.50	0.7	1007.50	0.7	1007.54	0.9
E-n101-k8	815	804.51	16.2	805.00	17.0	805.08	17.6	805.14	21.3
E-n101-k14	1067	1053.70	2.7	1054.09	2.8	1055.01	3.1	1055.01	3.2
F-n135-k7	1162	–	–	1160.57	6854.7	–	–	–	–
M-n121-k7	1034	1032.48	45.8	1032.57	565.1	1032.85	2873.2	–	–
M-n151-k12	1015	1000.39	30.8	1001.50	33.9	1002.20	38.9	1002.42	49.0
M-n200-k16	1274	1252.66	61.4	1253.04	67.7	1253.31	81.4	1254.16	91.4
M-n200-k17	1275	1255.27	68.1	1255.22	64.3	1255.63	70.6	1256.21	91.3
P-n50-k8	631	617.16	0.3	618.02	0.3	618.02	0.3	618.04	0.4
P-n70-k10	827	814.33	1.0	815.29	1.1	815.38	1.3	815.39	1.5

Table 4
Results for column generation with elementary routes for selected CVRP instances.

Ins	OPT	Elem		Elem + Cuts	
		LB	Time	LB	Time
A-n62-k8	1288	1254.83	1.4	1282.09	2.9
A-n63-k10	1314	1286.83	0.8	1303.87	1.3
A-n64-k9	1401	1376.90	1.3	1390.12	1.8
A-n69-k9	1159	1131.34	1.6	1145.24	1.8
A-n80-k10	1763	1731.58	3.3	1756.80	5.0
B-n50-k8	1312	1266.64	0.5	1303.74	0.7
B-n68-k9	1275	1204.00	1.5	1263.93	3.4
B-n78-k10	1221	1167.52	2.2	1217.25	4.9
E-n51-k5	521	517.14	0.7	519.49	1.2
E-n76-k7	682	665.59	4.6	671.94	5.9
E-n76-k8	735	718.78	2.4	727.20	4.2
E-n76-k10	830	812.48	1.2	818.01	2.0
E-n76-k14	1021	1002.77	0.6	1007.50	0.8
E-n101-k8	815	790.99	11.0	805.11	22.7
E-n101-k14	1067	1050.42	3.6	1055.01	4.2
F-n135-k7	1162	–	–	–	–
M-n121-k7	1034	1029.79	1301.8	–	–
M-n151-k12	1015	997.43	52.0	1002.58	57.5
M-n200-k16	1274	1252.05	140.9	1254.16	149.7
M-n200-k17	1275	1254.01	140.8	1255.94	147.5
P-n50-k8	631	615.55	0.2	618.04	0.3
P-n70-k10	827	810.94	0.8	815.29	1.2

shown in the tables were extracted from the work of Fukasawa et al. (2006), except the optimal value for instance M-n151-k12, which was first proved by Contardo (2012) and the optimal values for instances M-n200-k17 and M-n200-k16, which were first proved by the recent algorithm proposed by Pecin, Pessoa, Poggi, and Uchoa (2014).

7.2. Performance evaluation

This section evaluates the performance of our three *ng*-route pricing algorithms: the simple dynamic programming (Algorithm 1), the pure state space relaxation (Algorithm 2) and finally the algorithm that combines the techniques of state space relaxation with completion bounds (Algorithm 3). These algorithms were tested inside a column generation procedure on CVRP instances. The results are shown in Tables 1 and 2. Columns Ins and OPT show the name and the optimum value of each instance. Following these columns, the results for different values of $\Delta(N_i)$ are shown. For each X, where $\Delta(N_i) = X$, NG = X consists of four columns, LB, T1, T2 and T3, which show the lower bound and the total time required to compute it for, respectively, Algorithms 1–3.

The results from Tables 1 and 2 show that for small ng-set sizes (NG = 8), the best approach comes from the simple Algorithm 1. This is not a surprise because the maximum number of non-dominated labels per bucket is limited to 2^X, and therefore, the total number of labels handled by the algorithm does not explode. In this case, a unique running of the dynamic programming considering the entire state space is in general better than running it in the relaxed state space several times. For an average ng-set size (NG = 16), the pure DSSR still does not improve the times of Algorithm 1, but the combination of DSSR and completions bounds

Table 5
Summary of column generation results for GVRP instances.

Set	Num	θ	NG = 8		NG = 16		NG = 32		NG = 64		Elem	
			Gap (%)	Time	Gap (%)	Time	Gap (%)	Time	Gap (%)	Time	Gap (%)	Time
A	27	2	0.48	0.2	0.40	0.2	0.40	0.2	0.40	0.2	0.40	0.2
B	23	2	0.15	0.3	0.13	0.3	0.12	0.3	0.13	0.3	0.13	0.2
M	4	2	1.16	18.1	1.12	21.0	1.09	27.1	1.08	34.3	1.09	24.4
P	24	2	0.43	2.3	0.38	3.4	0.38	4.9	0.38	92.7	0.38	4.5
A	27	3	0.37	0.2	0.25	0.2	0.24	0.1	0.24	0.2	0.24	0.1
B	23	3	0.23	0.2	0.22	0.2	0.22	0.2	0.22	0.2	0.22	0.2
M	4	3	0.65	10.3	0.53	14.7	0.50	23.4	0.50	18.0	0.50	15.3
P	24	3	0.44	1.2	0.40	3.1	0.40	3.9	0.40	12.1	0.40	4.6

Table 6
Results for column generation with cut separation for selected GVRP instances.

Ins	Best bounds		NG = 8		NG = 16		NG = 32		NG = 64		Elem	
	LB	UB	LB	Time	LB	Time	LB	Time	LB	Time	LB	Time
A-n63-k9-C21-V3	625.6	642	629.7	0.2	636.3	0.2	636.3	0.2	636.3	0.2	636.3	0.1
A-n80-k10-C27-V4	679.4	710	706.5	0.5	708.8	0.5	708.8	0.4	708.8	0.5	708.8	0.4
A-n80-k10-C40-V5	957.4	997	982.5	1.1	982.7	1.2	982.8	1.3	983.4	1.1	983.4	0.9
M-n121-k7-C61-V4	707.7	719	710.4	26.0	710.7	37.8	710.9	58.2	710.9	83.5	710.9	40.7
M-n151-k12-C51-V4	465.6	483	482.4	10.4	**483.0**	11.3	**483.0**	13.2	**483.0**	11.2	**483.0**	12.7
M-n151-k12-C76-V6	629.9	659	649.3	13.2	649.6	16.4	649.9	13.4	650.2	14.7	650.0	13.8
M-n200-k16-C67-V6	563.1	605	592.5	15.0	593.6	15.5	594.2	19.2	594.2	19.1	594.2	13.8
M-n200-k16-C100-V8	744.9	791	777.7	29.0	778.4	25.7	778.7	32.0	778.8	33.6	778.8	32.5
P-n50-k8-C25-V4	378.4	392	385.7	0.1	385.8	0.2	385.8	0.2	385.8	0.2	385.8	0.1
P-n55-k15-C28-V8	545.3	555	**555.0**	0.1	**555.0**	0.1	**555.0**	0.0	**555.0**	0.1	**555.0**	0.0
P-n60-k10-C30-V5	433.0	443	435.6	0.3	435.4	0.3	435.3	0.3	435.3	0.3	435.3	0.2
P-n60-k15-C20-V5	380.0	382	380.3	0.1	**381.5**	0.1	**381.5**	0.1	**381.5**	0.1	**381.5**	0.1
P-n60-k15-C30-V8	553.9	565	**564.7**	0.1	**564.8**	0.1	**564.8**	0.1	**564.8**	0.1	**564.8**	0.1

provides a great improvement, allowing the running of all instances in a small amount of time (except for instances F-n135-k7 and M-n121-k7). For large ng-set sizes (NG$\geqslant$32), the pure DSSR significantly outperforms the basic dynamic programming algorithm, but it is still a poor algorithm for most instances. However, the DSSR with completions bound drastically improves the times.

We can also note that the bounds for NG = 32 are almost as good as the elementary for most instances. This is evidence that the routes found by the pricing with large ng-sets are almost elementary when the average size of the routes is up to 12 or 13 customers, in tipically less time than the time required if the elementary constraint is imposed to the routes. But this is not necessarily a rule. It is noteworthy to mention that the greater is the ng-set size used, the better is the completion bounds obtained. In some cases, the improved completion bounds resulting from the use of a large ng-set may compensate the additional complexity imposed. This is indicated by the time for running the instance M-n121-k7, which is significantly greater for NG = 32 than NG = 64.

On the other hand, our algorithms spend a lot of time trying to solve the column generation for instances F-n135-k7 and M-n121-k7, especially for large ng-set sizes. This is mainly due to the average size of the routes that are part of an optimum solution for these instances, which is greater than 17, causing the number of labels to be treated by the dynamic programming algorithm to be prohibitive. In particular, instance F-n135-k7 has a vehicle capacity of 2210, a value at least 10 times greater than the capacity of any other instance considered in the tests.

7.3. Other results

Now, we show the other results obtained with our best pricing algorithm, Algorithm 3. Table 3 presents the results of the column generation for the CVRP considering the separation of capacity and strengthened comb cuts. As performed previously, the results are shown for different values of $\Delta(N_i)$. Note that, considering the randomness factor included on the cut separation algorithms, larger $\Delta(N_i)$ do not necessarily implies in larger lower bounds, as one can check for example on instance A-n80-k10. Table 4 shows the results for the elementary routes for the CVRP. The column named Elem shows the bounds and times of the column generation without cuts and the column Elem + Cuts presents the results with cuts. Finally, Table 5 shows a summary of the results for the GVRP instances, all with separation of capacity and strengthened comb cuts. Columns Set, Num and θ show the name of the sets, the number of instances in each set and the θ used in the transformation, respectively. Columns NG = X show the average gap and average time of the sets for the column generation with the *ng*-route pricing for $\Delta(N_i) = X$ (similarly, the results for the elementary routes are presented for each set). These gaps are obtained by comparing our values with the upper bounds available in the work of Bektaş et al. (2011). We select some instances where our algorithms found better solutions than those described by Bektaş et al. (2011). The results for these instances are shown in Table 6. The column Ins shows the name of each instance, and columns LB and UB show the results from Bektaş et al. (2011). Values in bold are those which we also proved to be the optimal solution with our column generation algorithm.

8. Conclusions

The strength of the *ng*-route pricing is the ability of adjusting the size of the ng-sets in order to calculate a lower bound in a reasonable time, which is close as much as possible to the elementary route bound, and this property justifies an efficient implementation of the method. In this paper, we presented an efficient *ng*-route pricing algorithm for ng-set sizes up to sixty-four, a number at least three times greater than we know so far. Furthermore, we showed how our restricted non-elementary route pricing algorithm can be easily extended in order to price only elementary routes. We highlighted the two elements that allowed us to price elementary routes even for CVRP instances with 200 customers, a result which doubled the size of the ESPPRC instances solved so far. The first element is the way we adapt the Decremental State Space Relaxation (DSSR) technique of Righini and Salani (2008) for the *ng*-routes context, thus improving their way of increasing the state space along the DSSR iterations. The second is the combination of the DSSR technique with completion bounds, which are calculated in each iteration of the DSSR for the purpose of accelerating the next iteration.

The final algorithm was tested for pricing elementary and restricted non-elementary routes to a set partitioning formulation for both GVRP and CVRP. For the first one, we could improve the lower bounds for up to 13 instances, since we also separated and added capacity and strengthened comb cuts.

Acknowledgments

The contribution by Rafael Martinelli, Diego Pecin and Marcus Poggi has been partially supported by the Conselho Nacional de Desenvolvimento Científico e Tecnológico (CNPq), processes numbers 140849/2008-4, 141538/2010-4 and 309337/2009-7. This support is gratefully acknowledged.

References

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

Bartolini, E., Cordeau, J.-F., & Laporte, G. (2011). Improved lower bounds and exact algorithm for the capacitated arc routing problem. Tech. Rep. CIRRELT-2011-33, Interuniversity Research Centre on Enterprise Networks, Logistics and Transportation.

Bektaş, T., Erdoğan, G., & Røpke, S. (2011). Formulations and branch-and-cut algorithms for the generalized vehicle routing problem. *Transportation Science, 45*(3), 299–316.

Boland, N., Dethridge, J., & Dumitrescu, I. (2006). Accelerated label setting algorithms for the elementary resource constrained shortest path problem. *Operations Research Letters, 34*(1), 58–68.

Chabrier, A. (2006). Vehicle routing problem with elementary shortest path based column generation. *Computers & Operations Research, 33*(10), 2972–2990.

Christofides, N., Mingozzi, A., & Toth, P. (1981). Exact algorithms for the vehicle routing problem, based on spanning tree and shortest path relaxations. *Mathematical Programming, 20*, 255–282.

Contardo, C. (2012). A new exact algorithm for the multi-depot vehicle routing problem under capacity and route length constraints. Tech. rep., Archipel-UQAM 5078, Université du Québec à Montréal, Canada.

Di Puglia Pugliese, L., & Guerriero, F. (2012). A computational study of solution approaches for the resource constrained elementary shortest path problem. *Annals of Operations Research, 201*(1), 131–157.

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

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

Feillet, D., Dejax, P., Gendreau, M., & Gueguen, C. (2004). An exact algorithm for the elementary shortest path problem with resource constraints: Application to some vehicle routing problems. *Networks, 44*(3), 216–229.

Fischetti, M., Salazar-González, J. J., & Toth, P. (1997). A branch-and-cut algorithm for the symmetric generalized traveling salesman problem. *Operations Research, 45*(3), 378–394.

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

Ghiani, G., & Improta, G. (2000). An efficient transformation of the generalized vehicle routing problem. *European Journal of Operational Research, 122*(1), 11–17.

Irnich, S., & Villeneuve, D. (2006). The shortest-path problem with resource constraints and k-cycle elimination for $k \geqslant 3$. *INFORMS Journal on Computing, 18*(3), 391–406.

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

Pecin, D. G. (2010). Uso de rotas elementares no CVRP. Master's thesis, Instituto de Informática, Universidade Federal de Goiás (in portuguese).

Pecin, D., Pessoa, A., Poggi, M., & Uchoa, E. (2014). Improved branch-and-cut-and-price for capacitated vehicle routing. In *Integer programming and combinatorial optimization: Vol. 8494. Springer lecture notes in computer science* (pp. 393–403).

Poggi de Aragão, M. & Uchoa, E. (2003). Integer program reformulation for robust branch-and-cut-and-price algorithms. In *Proceedings of the conference mathematical program in Rio: A conference in honour of Nelson Maculan* (pp. 56–61).

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

Righini, G., & Salani, M. (2008). New dynamic programming algorithms for the resource constrained elementary shortest path problem. *Networks, 51*(3), 155–170.