A rollout algorithm for the resource constrained elementary shortest path problem

Francesca Guerriero, Luigi Di Puglia Pugliese and Giusy Macrina

Department of Mechanical, Energy and Management Engineering, University of Calabria, Rende, Italy

ABSTRACT

This paper presents a metaheuristic approach for the resource constrained elementary shortest path problem (RCESPP). RCESPP arises as pricing problem, when the vehicle routing problem is solved by branch-and-price algorithms. The availability of efficient metaheuristic and optimal solution approaches has contributed to the success of solution procedures based on column-generation. We focus on rollout strategies integrated with local search strategies. The scientific literature considers metaheuristics based on a tabu search procedure in order to price out columns. A comparative analysis between the proposed rollout approaches and the tabu search is conducted and the effectiveness of our proposed algorithms is tested. A comparison with exact solution approaches is also carried out in order to assess the behaviour of the implemented solution strategies in terms of both efficiency and solution quality.

1. Introduction

The Resource Constrained Shortest Path Problem (RCSPP) aims at finding a minimum-cost path from a given source node to a given destination node in a digraph, satisfying a set of constraints, defined over a set of resources.

Desrochers introduced the RCSPP in his Ph.D dissertation [15] as a subproblem of a bus driver scheduling problem. The RCSPP can be used to represent several real-life problems, arising in scheduling, telecommunications and transportation networks, thus, it has attracted the attention of many researchers over the last decade. Surveys of scientific contributions related to the RCSPP and its variants can be found in [16–18,26,27].

Its elementary version, referred to as the Resource Constrained Elementary Shortest Path Problem (RCESPP), deals with the specific situation in which the nodes belonging to the optimal path must appear at most once.

It is well known that the development of solution approaches for the RCESPP has contributed to the success of column-generation methods for addressing several versions of the vehicle routing problem (VRP). When the VRP is solved via a branch-and-price approach, the corresponding master problem is formulated as a set-partitioning model and
the corresponding pricing problem can be mathematically represented as a \( \text{RCESPP} \). Indeed, the variables derived from the dual problem of the set-partitioning formulation are viewed as the prizes associated with the nodes. The cost of the path is the sum of the cost for traversing the arcs minus the prizes collected at the nodes. Since the graph is assumed to be cyclic, it is necessary to introduce constraints that guarantee the elementarity of the optimal path.

In the column-generation framework, metaheuristics and/or heuristics are used to obtain negative reduced cost paths quickly. Optimal solution approaches are applied when the heuristic procedures fail to price out columns. Since the \( \text{RCESPP} \) is NP-hard in the strong sense (see [19]), a low number of calls to the optimal approach is beneficial for the entire resolution process. For this reason, the availability of effective heuristics could reduce the overall computational effort. However, performing an optimal algorithm is mandatory in order to prove the existence of negative reduced cost paths. Various optimal solution strategies have been proposed by the scientific literature for the \( \text{RCESPP} \) (see e.g. [4,9,20,30,35]).

Several papers, related to branch-and-price algorithm (see e.g. [8,12,32,33]), suggest to apply dynamic programming-based heuristics when the metaheuristic fails to price out columns and before applying an optimal solution strategy. The dynamic programming-based heuristics make use of modified optimal solution algorithms, that explore a restricted solution space. This allows to reduce the computational effort, but the determination of an optimal solution is not guaranteed.

In general, the pricing procedure is divided into three main levels. In the first level, a metaheuristic is applied. If a path with negative cost is not determined, the second level is executed where a dynamic programming-based heuristic is performed. The third level is invoked in order to solve to optimality the \( \text{RCESPP} \).

This paper focuses on the first level. The aim is to define an effective metaheuristic able to price out as high as possible columns.

Heuristic procedures to address the \( \text{RCESPP} \) have been proposed in [13,14,31,36]. In particular, Desaulniers et al. [14] applied a tabu search method for solving the \( \text{RCESPP} \) arising as subproblem in a branch-and-price-and-cut procedure for the \( \text{VRPTW} \). Archetti et al. [1] extended the approach presented in [14] to the split and delivery \( \text{VRPTW} \).

Dell’Amico et al. [13] investigated the \( \text{VRP} \) with simultaneous distribution and collection, by developing a column-generation based approach, that uses fast deterministic and randomized greedy algorithms to quickly generate columns with negative reduced cost.

Liberatore et al. [31] studied the \( \text{RCESPP} \) with time windows and implemented three different pricing algorithms with increasing complexity and accuracy: a greedy constructive algorithm, a local search algorithm and an approximate dynamic programming algorithm.

Salani and Vacca [36] considered the \( \text{VRP} \) with discrete split deliveries and time window constraints. The problem is solved via a column-generation approach. The authors stated the related pricing problem as an instance of the \( \text{RCESPP} \), and defined a simple greedy heuristic to obtain valid upper bounds on both the cost and the number of vehicles.

Pirkwieser and Raidt [34] presented a column-generation approach to address the periodic \( \text{VRP} \) with time windows. They formulated the pricing problem as a \( \text{RCESPP} \) with
time windows and restrictions on path duration and developed a metaheuristic, which can be viewed as a greedy randomized adaptive search procedure.

Several procedures have been proposed in the scientific literature to determine lower bounds on the optimal solution by eliminating cycles of at most a given length \( z \). In particular, dynamic programming approaches aimed at eliminating cycles of length two have been presented in \([11,25]\), whereas the case of \( z \geq 3 \) has been considered only in \([28]\).

Very recently, Baldacci et al. \([2]\) proposed a new relaxation of the \( RCESPP \) (i.e. the ng-route relaxation), that allows to obtain very good quality lower bounds.

In this work, we design rollout-based approaches (\( RH_s \)) to solve the \( RCESPP \). \( RH_s \) have been introduced originally in \([6]\) and \([7]\) and can be used to solve NP-hard combinatorial optimization problems. The basic idea is to use the cost obtained by applying a heuristic method (i.e. base heuristic) for discriminating among several search options at each step.

Even if our proposed approaches can be used merely to solve the \( RCESPP \), our main concern is to propose a new idea to efficiently generate columns in the context of the column-generation framework. When the \( RCESPP \) is solved as subproblem in column-generation, it is important to rapidly obtain a considerable number of negative reduced cost columns. Thus, the definition of a rollout procedure has been motivated by two main reasons.

Firstly, these algorithms are very appealing from the practical point of view. Indeed, several experimental results, on both deterministic and stochastic problems, in sequential and in parallel computing systems (see \([21–23]\)), have shown that \( RH_s \) significantly improve the performances of the base heuristic in terms of solution quality. Moreover, they are more robust than other metaheuristic techniques (e.g. Tabu Search and Simulated Annealing, as shown in \([7]\) because no input parameters needed to be tuned.

Secondly, a set of feasible paths is determined at each iteration. Thus, it is possible to store feasible solutions with negative reduced cost to populate the pool of paths in a column-generation procedure. Since a set of complete paths is determined at each iteration, a rollout-based approach is able to explore a large-size space.

The remainder of this paper is organized as follows: in Section 2 we introduce and mathematically formulate the \( RCSPP \) and the \( RCESPP \). In Section 3 we give a description of the proposed solution approaches. Since we compare the defined rollout algorithms for the \( RCESPP \) with a tabu search approach, for the sake of completeness, in Section 4 we describe the tabu search. In Section 5 we present the computational results, where the performance of the proposed solution strategies are evaluated and compared with the tabu search algorithm. In Section 6 we summarize the final remarks and conclusions. A detailed accounting of the computational results is reported in the Appendix of the supplemental material.

### 2. Problem definition and notations

The \( RCSPP \) is formulated on a directed graph \( G(\mathcal{N}, \mathcal{A}) \) where \( \mathcal{N} \) is the set of \( n \) nodes and \( \mathcal{A} \) is the set of \( m \leq |\mathcal{N}| \times (|\mathcal{N}| - 1) \) arcs.

Let \( c_{ij} \) and \( R_{ij} \) be the cost and an \( r \)-dimensional vector associated with each arc \((i,j) \in \mathcal{A} \), respectively. The \( r = |R_{ij}| \) scalars \( w_{h_{ij}}, h = 1, \ldots, r \), represent the consumption of the resource \( h \) along the arc \((i,j) \). Given two distinct nodes \( u \) and \( v \), a path \( \pi_{uv} \) from \( u \) to \( v \).
is a sequence of nodes and arcs \( \pi_{uv} = \{u = i_1, (i_1, i_2), i_2, \ldots, (i_{q-1}, i_q), i_q = v\} \). Let \( c(\pi_{uv}) \) be the cost of the path \( \pi_{uv} \) defined as the sum of the costs associated with the arcs of the path \( \pi_{uv} \). A resource extension function (REF), denoted as \( w^h(\pi_{uv}) \), is associated with each resource \( h = 1, \ldots, r \) and represents the quantity of resource \( h \) consumed along the path \( \pi_{uv} \).

Depending on the nature of the resource \( h \), \( w^h(\pi_{uv}) \) can assume different forms, i.e. additive, multiplicative, bottleneck-type (see [26] for more details on REF). Given a source node \( s \) and a destination node \( t \), the aim of the RCSPP is to find a path \( \pi_{st}^* \) from \( s \) to \( t \) such that \( c(\pi_{st}^*) \leq c(\pi_{st}) \) for each path \( \pi_{st} \) in \( \mathcal{G}(\mathcal{N}, \mathcal{A}) \), where \( \pi_{st}^* \) and \( \pi_{st} \) satisfy feasibility criterion associated with each resource \( h = 1, \ldots, r \).

A resource window \([a^h_i, b^h_i]\), with \( a^h_i \leq b^h_i \), \( h = 1, \ldots, r \) is associated with each node \( i \in \mathcal{N} \). A path \( \pi_{st} \) from node \( s \) to node \( t \) is feasible if and only if \( a^h_i \leq w^h(\pi_{st}) \leq b^h_i \). Depending on the values of \( a^h_i \) and \( b^h_i \), at least two types of constraints can be modelled: 1) time windows constraints and 2) capacity constraints. For the former, the resource \( h \) represents time and \( a^h_i, b^h_i \in \mathbb{R}_+, \forall i \in \mathcal{N} \). We distinguish between two kinds of time window constraints: hard and soft constraints. However, in this work only hard constraints are considered. Thus, if the node \( i \) is visited before \( a^h_i \), then waiting is allowed at no cost, in addition, all arrival times after \( b^h_i \) are declared infeasible. For the capacity constraints, \( a^h_i = 0 \) and \( b^h_i = W^h, \forall i \in \mathcal{N}, \) where \( W^h \) is the vehicle capacity.

The RCSPP is the elementary version of RCSPP and its aim is to find the elementary minimum-cost path \( \pi_{st}^* \) from node \( s \) to node \( t \). A path is elementary when it does not contain repeated nodes. Since the cost \( c_{ij}, \forall (i,j) \in \mathcal{A} \), is not constrained in sign and the graph is not assumed to be acyclic, it is necessary to impose that the final solution must be elementary, that is a node cannot appear more than once in the final path.

We highlight that the RCSPP arises as subproblem, named pricing problem, when the VRP is solved via column-generation method; thus the VRP is mathematically reformulated as a set-partitioning problem as follows:

\[
\min \sum_{\pi_{st} \in \tilde{\Pi}} \sum_{(i,j) \in \mathcal{A}} c_{ij}\mu_{ij}(\pi_{st}) x(\pi_{st}) \tag{1}
\]

\[
\text{s.t.} \quad \sum_{\pi_{st} \in \tilde{\Pi}} \sum_{(i,j) \in \mathcal{A}} \mu_{ij}(\pi_{st}) x(\pi_{st}) = 1, \forall i \in \mathcal{N} \tag{2}
\]

\[
x(\pi_{st}) \in \{0, 1\}, \forall \pi_{st} \in \tilde{\Pi} \tag{3}
\]

where \( \tilde{\Pi} \) is the set of all feasible paths from node \( s \) to node \( t \), \( \mu_{ij}(\pi_{st}), \forall (i,j) \in \mathcal{A} \) are binary constants that assume a value of 1 if and only if arc \((i,j)\) belongs to the path \( \pi_{st} \); the binary variables \( x(\pi_{st}) \) are used to indicate whether the path \( \pi_{st} \) is selected or not.

Solving the set-partitioning formulation requires the determination of the set \( \tilde{\Pi} \). However, it is well known that enumerating all paths is impractical. Thus, a relaxation of the model (1)–(3), obtained by considering a subset of the set \( \tilde{\Pi} \) is considered. In this framework, a column-generation approach is used with the aim of finding promising paths not yet included in the subset. Thus a subproblem (i.e. the pricing problem) have to be solved.
in order to price columns (paths) of the set-partitioning problem. In the context of the VRP, the pricing problem represents an instance of the RCESPP.

Let $\lambda, i = 1, \ldots, |N|$ be the dual variables associated with the constraints (2), the reduced cost $\hat{c}(\pi_{st})$ of the path $\pi_{st}$ is:

$$\hat{c}(\pi_{st}) = \sum_{(i,j) \in A} c_{ij}\mu_{ij}(\pi_{st}) - \sum_{(i,j) \in A} \lambda_{i}\mu_{ij}(\pi_{st}) = \sum_{(i,j) \in A} (c_{ij} - \lambda_{i})\mu_{ij}(\pi_{st}).$$

The pricing problem aims to find a feasible minimum reduced cost path, and it can be formulated by the following RCESPP mathematical model:

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

s.t.

$$\sum_{\{j: (i,j) \in A\}} x_{ij} - \sum_{\{j: (j,i) \in A\}} x_{ji} = \begin{cases} 1, & \text{if } i = s \\ -1, & \text{if } i = t \\ 0, & \text{otherwise} \end{cases} \quad (5)$$

$$\sum_{i \in \tilde{N}} \sum_{j \in \tilde{N}} x_{ij} - |\tilde{N}| \leq -1, \forall \tilde{N} \subseteq N \quad (6)$$

$$x \in X \quad (7)$$

$$x_{ij} \in \{0, 1\}, \forall (i,j) \in A, \quad (8)$$

where $\hat{c}_{ij} = c_{ij} - \lambda_{i}$ and $X$ in Equation (7) defines the solution space induced by the resource constraints. In other words, $X$ reflects the requirement imposed to the routes in the VRP in terms of resource consumption. The constraints (6) are the subcycle elimination constraints, where $\tilde{N} \subseteq N$ is a subset of nodes. We highlight that the proposed rollout algorithms work for any definition of set $X$. However, in the computational analysis, we focus our attention to capacity and time window constraints. For the sake of comprehension, we report the definition of set $X$ for the two types of aforementioned resource constraints.

Assuming $h = 1$ is the resource associated with either the pick-up or the delivery operations and $h = 2$ is the resource time, set $X$ is defined as follows:

$$\sum_{(i,j) \in A} w_{ij}^{1}x_{ij} \leq W, \quad (9)$$

$$x_{ij}(w_{j}^{2} - w_{i}^{2} - w_{ij}^{2}) = 0, \forall (i,j) \in A, \quad (10)$$

$$a_{i} \leq w_{i}^{2} \leq b_{i}, \forall i \in N, \quad (11)$$

where $w_{i}^{2}$ is the arrival time to node $i \in N$ from source node $s$, i.e., $w_{i}^{2} = w^{2}(\pi_{si})$. 
3. Rollout approach for the RCESSP

In this section, we present the proposed RH defined to address the RCESSP. A RH can be viewed as a construction procedure, that determines a solution of the problem under investigation, by starting with some feasible partial solution, which is a feasible path starting at node \( s \) and ending at node \( i \in \mathcal{N} \setminus \{ t \} \), (denoted as \( \Pi \)) and by enlarging the current solution iteratively.

A base heuristic, denoted as \( \mathcal{H} \), is used to evaluate the different search options at each step. It is assumed that, given a partial solution \( \Pi \), a complete feasible solution (i.e. a path starting at node \( s \) and ending at node \( t \)) is determined by applying \( \mathcal{H} \) and the corresponding cost is denoted by \( c(\Pi) \).

Let \( \Omega(\Pi) \) be the neighbourhood of the partial solution \( \Pi \) (i.e. the set of all feasible solutions that can be obtained from \( \Pi \) applying an elementary perturbation or move).

At the generic iteration \( k \), a partial solution \( \Pi^{(k)} \) is available, thus, the solution neighbourhood \( \Omega(\Pi^{(k)}) \) of \( \Pi^{(k)} \) can be determined. \( \Omega(\Pi^{(k)}) \) contains all feasible solutions that can be obtained from \( \Pi^{(k)} \) by adding an additional element (i.e. a node). We refer to \( \Pi_i^{k} \) as the complete feasible solution forced to contain node \( i \), generated starting from the partial solution \( \Pi^{(k)} \). Let \( \Pi_0 \) be an initial partial feasible solution, the general scheme of \( \mathcal{RH} \) is depicted in Algorithm 1 [21].

**Algorithm 1. \( \mathcal{RH} \) general scheme.**

- **Step 0.** Set \( k = 0 \); \( \Pi^{(k)} = \Pi_0 \)
- **Step 1.** If \( \Pi^{(k)} \) is a complete solution: **exit**
  - Otherwise, undertake Steps 1a and 1b below.
  - **Step 1a.** If \( \Omega(\Pi^{(k)}) = \emptyset \): **exit**
    - Otherwise, select from the neighbourhood \( \Omega(\Pi^{(k)}) \) of \( \Pi^{(k)} \) the solution \( \Pi_i^k \) for which the heuristic cost is minimized, that is:
      \[
      \Pi_i^k \in \Omega(\Pi^{(k)}): c(\Pi_i^k) \leq c(\Pi), \forall \Pi \in \Omega(\Pi^{(k)})
      \]
  - **Step 1b.** Set \( k = k + 1 \), \( \Pi^{(k)} = \Pi^{(k-1)} \cup \{i\} \). Go to **Step 1**.

It is worth observing that if a feasible solution \( \pi_{st} \) does not exist, then the neighbourhood \( \Omega(\Pi^{(0)}) \) is empty and Algorithm 1 terminates.

Starting from Algorithm 1, a \( \mathcal{RH} \) for solving the RCESSP can be derived, as described in the following. At each iteration \( k \) of \( \mathcal{RH} \), \( \Pi^{(k)} \) corresponds to the partial path \( (s, (s, i_1), i_1, (i_1, i_2), i_2, \ldots, i_{k-1}, (i_{k-1}, i_k), i_k) \) and for \( k = 0 \), \( \Pi_0 = (s) \). The solution neighbourhood \( \Omega(\Pi^{(k)}) \) represents all the feasible solutions (i.e. feasible paths) obtained by adding to the partial path \( \Pi^{(k)} \) one more node \( i_{k+1} \). Obviously, \( i_{k+1} \) can be added to the current partial path \( \Pi^{(k)} \) if and only if it does not already belong to \( \Pi^{(k)} \) and the newly created path satisfies the resource constraints. Furthermore, given a partial path \( \Pi^{(k)} = (s, (s, i_1), i_1, (i_1, i_2), i_2, \ldots, i_{k-1}, (i_{k-1}, i_k), i_k) \), we use the base heuristic \( \mathcal{H} \) to generate a complete feasible path \( \Pi_{i_{k+1}}^{k} = (s, (s, i_1), i_1, (i_1, i_2), i_2, \ldots, i_k, (i_k, i_{k+1}), i_{k+1}, \ldots, t) \), starting from \( \Pi^{(k)} \), with cost \( c(\Pi_{i_{k+1}}^{(k)}) \) forcing the complete feasible solution to contain node \( i_{k+1} \). The node \( i_{k+1} \) that results in a complete feasible solution generated by \( \mathcal{H} \) with
the smallest cost is selected and added to $\Pi^{(k)}$, that is, $\Pi^{(k+1)} = \Pi^{(k)} \cup \{i_{k+1}\}$, such that:

$$
i_{k+1} = \arg \min_{\Pi^k_i \in \Omega(\Pi^{(k)})} c\left(s, (s, i_1), i_1, (i_1, i_2), i_2, \ldots, i_k, (i_k, \tilde{i}), \tilde{i}, \ldots, t\right), \quad (12)
$$

where $\Pi^k_i = (s, (s, i_1), i_1, (i_1, i_2), i_2, \ldots, i_k, (i_k, \tilde{i}), \tilde{i}, \ldots, t)$ is the complete and feasible path generated by $H$ starting from $\Pi^{(k)}$ containing node $\tilde{i}$.

It is worth observing that, in order to add a node to the current solution, the base heuristic is executed $O(n)$ times, whereas each complete feasible solution (i.e., an elementary paths) contains at most $n$ nodes. Consequently, the computational complexity of $RH$ for $RCESPP$ is $O(n^2\theta)$, where $O(\theta)$ represents the computational complexity of the base heuristic $H$.

In the proposed $RH$, we have considered as base heuristic $H$ a hybrid approach defined by combining a construction heuristic, referred to as $H_c$, with a local search method, denoted by $H_{ls}$.

$H_c$ is a simple greedy construction algorithm (nearest feasible neighbour heuristic), that starts from the origin node $s$ and adds new nodes, by taking into account the elementary and resource constraints, until the destination node $t$ is reached. Given a sequence of nodes ending to node $i$, $H_c$ chooses the next node $j$ to be added minimizing the cost $c_{ij}$ and satisfying all constraints, $\forall (i, j) \in A$.

$H_{ls}$ is an improvement heuristic that, starting from a feasible path from node $s$ to node $t$ determined by $H_c$, attempts to improve the solution by a sequence of arc interchanges by using ‘Or-Opt’ algorithm. The ‘Or-Opt’ attempts to improve a current path, by moving a chain of two or three consecutive nodes in a different position. Then, the preceding and the succeeding nodes in the earlier path, are linked by an arc. This kind of local search is widely applied when solving the $VRP$. For more details on ‘Or-Opt’, the reader is referred to [10].

4. Tabu search approach for the $RCESPP$

The Tabu Search ($TS$) method, used for comparison, is inspired by the approach presented in [1]. The authors in their work addressed the $VRPTW$. In particular, they implemented a $TS$ algorithm in order to solve the pricing subproblem when the $VRPTW$ is solved by using a branch-and-cut algorithm.

Let $\pi$ be the route which starts from the source node $s$, visits a set of nodes $N_\pi$ and ends at node $t$ and $N_r = N \setminus N_\pi$. Let $K_{\text{max}}$ be the maximum number of iterations imposed for the $TS$. Let us indicate with $TL_{\text{insert}}$ and $TL_{\text{remove}}$ the tabu search lists. The $TS$ procedure relies on the following two operators:

(1) **Remove**: the main idea is to determine the most expensive node (in terms of cost) belonging to the route. For each vertex $i \in N_\pi \setminus TL_{\text{remove}}$, the operator evaluates $\delta(i)$, representing the decrement in the objective function value, if node $i$ is removed from the current solution, and removes the node $i_{\text{best}}$, for which the greatest decrement is obtained, that is, $i_{\text{best}} = \arg \max_{i \in N_\pi \setminus TL_{\text{remove}}} \{\delta(i)\}$.

(2) **Insert**: the aim is to find the best possible insertion position for all the nodes in $N_r \setminus TL_{\text{insert}}$. The operator tries to insert each node $j \in N_r \setminus TL_{\text{insert}}$ into the current route.
If all the constraints are respected, the cost $\gamma(j)$, that is the value that the objective function assumes if the node $j$ is added to the route, is calculated. The chosen node $j_{\text{best}}$ is then added to the route, where $j_{\text{best}} = \arg\min_{j \in N_r \setminus TL_{\text{insert}}} \{\gamma(j)\}$. In addition, an Or-Opt procedure is applied to try to improve the solution obtained after the node $j_{\text{best}}$ is inserted.

When a node $i$ is removed from the route, it is added to the tabu list $TL_{\text{insert}}$, and it is forbidden to insert this node to the route for a certain number of iterations defined as $TL_{\text{max}}$. Similarly, if a node $j$ is added to the route, it is also added to the tabu list $TL_{\text{remove}}$, thus, removing this node from the route is a forbidden move for $TL_{\text{max}}$ iterations. A greedy algorithm is used in order to provide an initial solution for the $TS$.

The pseudo code of the $TS$ is reported in Algorithm 2.

Algorithm 2. $TS$

$k = 0; TL_{\text{insert}} \leftarrow \emptyset; TL_{\text{remove}} \leftarrow \emptyset; \pi \leftarrow \text{initial solution}$

while $k < K_{\text{max}}$ do

Perform the moves on $\pi$:
- Remove ($i$), $\forall i \in N_\pi \setminus TL_{\text{remove}}$,
- Insert ($j$), $\forall j \in N_r \setminus TL_{\text{insert}}$,

obtaining the new solution $y$.

if $f(y) < f(\pi)$ then

$\pi^* := y$

end if

Update Lists:
- $TL_{\text{insert}} \leftarrow TL_{\text{insert}} \cup (i_{\text{best}})$, 
- $TL_{\text{remove}} \leftarrow TL_{\text{remove}} \cup (j_{\text{best}})$,
- $N_r \leftarrow N_r \setminus j_{\text{best}}$,
- $N_\pi \leftarrow N_\pi \setminus i_{\text{best}}$,
- $\pi := y$.

end while

return $\pi^*$

5. Computational experiments

This section presents the results of the experiments carried out in order to assess the performance of the proposed algorithms.

Starting from a naive version of the rollout algorithm, named ($N\mathcal{R}H$), where the base-heuristic is composed of only the construction heuristic $H$, we defined several versions by combining different local search-based heuristic. We compared the proposed solution approaches with the $TS$, and two optimal solution strategies, i.e. the general state space augmenting algorithm ($GSS\mathcal{A}A$, for short) [9], and the branch-and-cut ($BAC$, for short) proposed in [29]. We coded the proposed solution approaches, the $TS$ and the $GSS\mathcal{A}A$ in Java language. We carried out the computational experiments by using an Intel PC Core\textsuperscript{TMM}i5 - 3337U CPU @ 1.80GHz, 4.00 GB RAM, under Microsoft Windows 8 operating system.
The computational study is divided into two parts. In the first part (see Section 5.2), we carried out a comparison with the TS procedure. To this aim, we consider a version of the proposed rollout algorithm, that is, $RH_1$ where the Or-Opt algorithm \cite{10} is applied starting from the complete solution generated by the $NRH$. The aim of this study is to highlight the usefulness of $RH_1$ when it is embedded in a column-generation procedure. It is well known that the efficiency of the branch-and-price is strongly related to the ability of the heuristic to price out the columns. The number of columns generated has a direct influence in the number of iterations. Scientific contributions highlight that for computationally demanding difficult problems, such as VRP, the higher the number of negative columns generated, the more efficient the branch-and-price algorithm (see e.g. \cite{3}). Part two deals with the evaluation of the results obtained by testing the proposed rollout algorithm and its variants (see Section 5.4). The aim of this part is to give an idea on the quality of the solution in terms of optimality gap. Indeed, beside the number of pricing columns, high quality solution could accelerate the search process in the branch-and-price algorithm.

5.1. Test problems

We carried out computational experiments on two sets of benchmark instances (referred to as S1 and S2).

The instances of set S1 are those used in \cite{16}. They are obtained by the original Solomon’s networks of the class clustered (C), random (R) and random clustered (RC). The three classes differ each others for the positions of the vertexes in the field. To obtain negative cost, we associate a price with each vertex chosen randomly from the range [0, 20]. The price values are selected exactly as done in \cite{35}. From each original instance, one $RCESPP$ instance with 50 nodes has been derived, setting the vehicle capacity to 40.

The test class S2 refers to the networks used in \cite{29}. They are ‘pricing problems’ generated by solving VRP with Capacity Constraints (CVRP) via column-generation algorithms. The dual values have been subtracted, thus some costs are negative. According to the authors, these networks have been divided into six different sets: A, B, E, M, P and G, each of them have a different range of number of nodes (i.e. $A$ [54,80], $B$ [45,78], $E$ [76,101], $G$ 262, $M$ [101,200], $P$ [50,101]).

5.2. Comparative analysis

In this section we present a comparative analysis of the results of the $RH_1$ algorithm with the results of the TS.

At first, we conducted a series of tests in order to choose values of the TS’s parameters. After, we focus on the comparison of the proposed approaches.

Table 1 describes the three parameters used in our work. In particular, the table includes the maximum number of tabu search iterations ($K_{\text{max}}$) and the size of tabu lists ($TL_{\text{max}}$), which depends on a random number $\rho$. This number is related to the size of the instances, in other words it depends on the number of the nodes in the network and it is selected in the interval $[1, |N|]$. We tested all the instances, using $K_{\text{max}}$ taken from \{10, 20\}. For each value of $K_{\text{max}}$, we use six different values of $TL_{\text{max}}$. We chose these parameters in a way that allows a good exploration of the solution space. Tables A2 and A3 in the Appendix of the
Table 1. Parameter values.

| Parameter | Description                        | Value                          |
|-----------|------------------------------------|--------------------------------|
| $K_{\text{max}}$ | Maximum number of Tabu search iterations | {10, 20}                      |
| $\rho$   | Random number related to the size of the instance | $5 + 0.5 \times \rho$          |
| $T_{\text{L}_{\text{max}}}$ | Length of Tabu list | $5 + 0.5 \times |N| + 0.3 \times \rho$         |

supplemental material, give a detailed accounting on the computational results obtained by varying the values of the relevant parameters.

In particular, when the value of $T_{\text{L}_{\text{max}}}$ depends on $\rho$, we executed the algorithm five times on each instance for each different value of $K_{\text{max}}$ and $T_{\text{L}_{\text{max}}}$. The average on the objective function values is reported under the columns $\text{avg cost}$, in the Tables A2 and A3. When $T_{\text{L}_{\text{max}}}$ has a fixed value, the objective function is given under the columns $\text{cost}$.

From Tables A2 and A3 it is possible to observe that the algorithm obtains the best value of $\text{avg cost}$ when $K_{\text{max}} = 20$ and $T_{\text{L}_{\text{max}}} = 5 + 0.5 \times \rho$. Thus, for our tests we set the parameters accordingly to these values.

To analyse and compare the performance of the algorithms, we report in Table 2 the results attained for the classes $C$, $R$ and $RC$ belonging to the set $S1$, and in Table 3 those obtained for the set $S2$. In row $\text{AVG}$ we report the average results for each class of problems, that is, $C$, $R$ and $RC$ for set $S1$ and $A$, $B$, $E$, $M$, and $P$ for set $S2$. In row $\text{AVG1}$ we highlight the average results over all instances, for each set.

Each instance is solved nine times by the $TS$ where the parameter $\rho$ is chosen randomly in the interval $[1, |N|]$.

Each table contains three columns for each strategy: $\text{time}$ is the time in milliseconds spent by $RH_1$ to solve each instance, $\text{cost}$ is the value of the objective function and $\text{nc}$ is the number of the negative columns generated by $RH_1$. $\text{AVG time}$ and $\text{AVG cost}$ are the average on time in milliseconds and the average on objective function values obtained when the instance is solved by $TS$, respectively, while $\text{AVG nc}$ is the average on the number of negative columns generated by $TS$.

According to Table 2, $RH_1$ is competitive in terms of cost and number of negative columns (see row $\text{AVG1}$). Even if by using $TS$ there is, on average, an overall reduction in computational time of about 22%, $RH_1$ is able to find about the 24% more of negative columns compared to $TS$. We also note that for the 89.66% of the instances, the value of the best solution obtained with $RH_1$ is less than or equal to that obtained by $TS$. It is worth observing that for the instances generated from the class $RC$, both $RH_1$ and $TS$ do not find any path with negative cost. However, $RH_1$ behaves the best in terms of solution quality. Indeed, the average cost of the solutions determined by $TS$ is 1.35 times higher than the average cost of the solutions found by $RH_1$. Considering the computational overhead, the two approaches show similar performance and $RH_1$ behaves slightly better than $TS$. The average computational effort of $RH_1$ is 7.50 milliseconds against 7.64 milliseconds required by $TS$.

Table 3 presents the average results for the instances belonging to the set $S2$. The average performance in terms of cost and number of negative columns of $RH_1$ is clearly better.
Table 2. Results obtained by solving the instances C, R and RC, belonging to S1.

| Test  | RH₁ | TS | AVG | AVG | AVG |
|-------|-----|----|-----|-----|-----|
|        | Time | cost | nc  | AVG time | AVG cost | AVG nc |
| c101_50 | 150.00 | −9.00 | 5.00 | 88.44 | −4.00 | 1.00 |
| c102_50 | 86.00 | −13.00 | 16.00 | 43.44 | −15.00 | 21.00 |
| c103_50 | 31.00 | −15.00 | 19.00 | 47.11 | −16.00 | 25.00 |
| c104_50 | 42.00 | −14.00 | 24.00 | 50.00 | −16.00 | 29.00 |
| c105_50 | 30.00 | −9.00 | 6.00 | 10.22 | −4.00 | 1.00 |
| c106_50 | 18.00 | −9.00 | 5.00 | 7.56 | −4.00 | 1.00 |
| c107_50 | 16.00 | −9.00 | 7.00 | 15.89 | −4.00 | 1.00 |
| c108_50 | 48.00 | −10.00 | 10.00 | 12.22 | −4.00 | 1.00 |
| c109_50 | 59.00 | −15.00 | 19.00 | 17.33 | −4.00 | 1.00 |
| AVG    | 53.33 | −11.44 | 12.33 | 32.47 | −7.89 | 9.00 |
| r101_50 | 12.00 | −6.00 | 8.00 | 18.89 | −8.00 | 4.00 |
| r102_50 | 27.00 | −8.00 | 10.00 | 20.67 | −14.00 | 9.00 |
| r103_50 | 31.00 | −14.00 | 12.00 | 15.78 | −14.00 | 12.00 |
| r104_50 | 37.00 | −14.00 | 13.00 | 4.00 | −14.00 | 2.00 |
| r105_50 | 10.00 | −14.00 | 10.00 | 19.22 | −14.00 | 14.00 |
| r106_50 | 21.00 | −14.00 | 11.00 | 15.33 | −14.00 | 9.00 |
| r107_50 | 24.00 | −14.00 | 12.00 | 8.44 | −14.00 | 12.00 |
| r108_50 | 26.00 | −14.00 | 13.00 | 4.78 | −14.00 | 2.00 |
| r109_50 | 19.00 | −14.00 | 10.00 | 14.78 | −14.00 | 8.00 |
| r110_50 | 23.00 | −14.00 | 13.00 | 25.00 | −14.00 | 17.00 |
| r111_50 | 29.00 | −14.00 | 13.00 | 143.67 | −14.00 | 19.00 |
| AVG    | 24.25 | −12.83 | 11.50 | 24.69 | −12.67 | 10.00 |
| rc101_50 | 4.00 | 39.00 | 0.00 | 3.44 | 59.00 | 0.00 |
| rc102_50 | 2.00 | 48.00 | 0.00 | 5.11 | 53.00 | 0.00 |
| rc103_50 | 6.00 | 36.00 | 0.00 | 3.67 | 53.00 | 0.00 |
| rc104_50 | 15.00 | 33.00 | 0.00 | 12.44 | 53.00 | 0.00 |
| rc105_50 | 7.00 | 48.00 | 0.00 | 9.11 | 53.00 | 0.00 |
| rc106_50 | 9.00 | 39.00 | 0.00 | 5.44 | 53.00 | 0.00 |
| rc107_50 | 8.00 | 36.00 | 0.00 | 17.89 | 53.00 | 0.00 |
| rc108_50 | 9.00 | 39.00 | 0.00 | 4.00 | 53.00 | 0.00 |
| AVG    | 7.50 | 39.75 | 0.00 | 7.64 | 53.75 | 0.00 |
| AVG1   | 28.66 | 2.10 | 8.59 | 22.40 | 7.14 | 6.93 |

than that obtained with TS. RH₁ outperforms TS in terms of effectiveness. Indeed, RH₁ finds the best solution for all the instances of the set. Despite TS is faster in finding the solution, the quality of solutions is always widely worse (see row AVG in Table 3). Furthermore, it is worth to observe that RH₁ quickly finds more negative columns than TS. Indeed, RH₁ finds one negative column in 0.38 milliseconds, on average, while TS spends 0.96 milliseconds, the increase is about 152%. In summary, RH₁ finds a large amount of negative columns in a quite short time for both the classes S1 and S2. In addition, the solutions found are not only superior in number, but they are more effective compared to those obtained by using TS.

5.3. Rollout approach: implemented solution strategies

With the purpose to improve the effectiveness of the proposed approach, we considered several different versions of RH, that relies on the use of local search method. We used three different ‘l-Opt algorithms’ in this work, that is 2-Opt, 3-Opt and Or-Opt [10]. In the first version, described in Section 3 and named RH₁, the Or-Opt is applied starting from the complete solution determined by the NRH. In the second one, that is, the RH₂ all the
considered local search methods (in the order Or-Opt, 2-Opt, 3-Opt) are applied starting from the complete solution determined by the $\mathcal{N}^*\mathcal{RH}$. In the third version, referred to as $\mathcal{R}H_3$, the local search is used to improve the complete solution determined by $\mathcal{H}_c$, for evaluating all nodes belonging to the neighbourhood, at each iteration. Whereas in the fourth one, denoted by $\mathcal{R}H_4$, the local search is used to improve the complete solution obtained
by $H_c$ every $p$ iterations, for each neighbourhood. In the last version, named $\mathcal{RH}_5$, $H_c$ is applied for a subset of the neighbours of the current partial solution at each iteration.

In addition, starting from the basic $\mathcal{RH}$, we considered two enhanced versions. The former is a modified version of the ‘Fortified Rollout Algorithm’ $\mathcal{F}(\mathcal{RH})$ [5]. The main idea is to maintain, at each iteration $k$, in addition to the partial solution $\Pi_{k+1}^{(k)}$, a complete path $\Pi_{stk}^{k+1}$ from node $s$ to node $t$, which represents the best feasible trajectory obtained so far, whose cost is denoted by $c(\Pi_{stk}^{k+1})$.

If the cost of the best complete solution $c(\Pi_{k+1}^{(k+1)})$, $\Pi_{k+1}^{(k+1)} \in \Omega(\Pi^{(k)})$, obtained by applying the base heuristic $H_c$ to the partial solution $\Pi_{k+1}^{(k)}$, is better than $c(\Pi_{stk}^{k+1})$, the node $i_{k+1}$ is added to $\Pi^{(k)}$ and $\Pi_{stk}^{k+1}$ is updated accordingly. In the contrary case, the partial solution $\Pi_{k+1}^{(k+1)}$ is set equal to the first $k+1$ nodes of the complete solution $\Pi_{stk}^{k+1}$ and the procedure is iterated until a complete solution is found.

We obtained the second version by incorporating a two step look-ahead strategy into the rollout framework (see [5]). The aim of this variant, denoted by $2SL$, is to predict the profitability of choosing a node, by estimating its effect on its descendant nodes. In particular, at the $k$-th iteration, the partial path $\Pi_{k+1}^{(k)}$ is generated starting from a sequence of two given nodes and the effect on the cost, in the case that these two nodes are added to the path $\Pi_{stk}^{(k)}$, is evaluated.

In the testing phase, we implemented and tested the following strategies:

1. $\mathcal{NRH}$: the naive version of $\mathcal{RH}$ that used $H_c$ as base heuristic.
2. $\mathcal{RH}_1$: the Or-Opt algorithm is applied starting from the complete solution generated by $\mathcal{NRH}$.
3. $\mathcal{RH}_2$: all the local search methods are applied sequentially (in the order Or-Opt, 2-Opt, 3-Opt) starting from the complete solution generated by $\mathcal{NRH}$.
4. $\mathcal{RH}_3$: hybrid version of $\mathcal{RH}$, in which the Or-Opt algorithm is used to improve the complete solution generated by $H_c$ at each iteration.
5. $\mathcal{F}(\mathcal{RH}_3)$: fortified version of $\mathcal{RH}_3$; the Or-Opt is used every $p$ iterations in order to improve the complete solution generated by $H_c$. In the computational test we consider three different values for $p$, that is, $p \in \{1, 2, 3\}$.
6. $\mathcal{F}(\mathcal{RH}_3) + (\mathcal{RH}_2)$: fortified version of $\mathcal{RH}_3$, in addition the local search methods are applied sequentially (in order Or-Op, 2-Opt, 3-Opt) starting from the complete solution determined by $\mathcal{F}(\mathcal{RH}_3)$.
7. $2SL(\mathcal{F}(\mathcal{RH}_3))$: $\mathcal{F}(\mathcal{RH}_3)$ version that implements the two step look-ahead strategy.
8. $2SL(\mathcal{F}(\mathcal{RH}_3) + (\mathcal{RH}_2))$: the Or-Opt algorithm is applied starting from the complete solution determined by $2SL(\mathcal{F}(\mathcal{RH}_3))$.
9. $2SL(\mathcal{F}(\mathcal{RH}_4))$: fortified hybrid version of $\mathcal{RH}$ that implements the two step look-ahead strategy and uses the Or-Opt algorithm only every $p$ iterations, randomly chosen in the interval $[0, |N|]$, in order to improve the complete solution generated by $H_c$.
10. $2SL(\mathcal{F}(\mathcal{RH}_5))$: fortified hybrid version of $\mathcal{RH}$ that implements the two step look-ahead strategy. The $\mathcal{RH}_5$ differs from the $\mathcal{RH}_3$ in the Or-Opt algorithm application. Indeed, the local search algorithm is applied to improve a subset, randomly chosen, of the complete solution generated by $H_c$ at each iteration.
(11) $2\mathcal{S}\mathcal{L}(\mathcal{F}(\mathcal{R}\mathcal{H}_5) + (\mathcal{R}\mathcal{H}_2))$ the Or-Opt algorithm is applied to the complete solution determined by $2\mathcal{S}\mathcal{L}(\mathcal{F}(\mathcal{R}\mathcal{H}_5))$.

### 5.4. Rollout-based algorithms results

We conducted experiments to evaluate the performances, in terms of both computational efficiency and solution quality, of the aforementioned strategies for solving the \textit{RCESPP}. It is worth to observe that we do not expect that our proposed algorithms have superior performances compared to the exact efficient algorithms proposed in the literature [9,29]. Actually, we want to propose a new potential approach that can be embedded in the column-generation, not only by generating a large amount of negative columns in a short computational time, but also by providing good quality solutions.

We compared the solutions found by the rollout approaches, for the instances belonging to the set $S1$, with those determined by using the \textit{GSSAA}. For detailed results, the reader is referred to Appendix of the supplemental material. Table A4 presents the time spent, in milliseconds, to solve each instance by the algorithms and the relative value of objective function. Table 4 summarizes the average results obtained with the \textit{GSSAA}, compared with the different rollout strategies. According to Table 4, the rollout algorithms are highly competitive in terms of computational time. Focusing on set $R$, we are able to find 11 optimal solutions out of 12 instances by using the majority of the algorithms within a significantly reduced time. According to Table 4, our algorithms outperform the \textit{GSSAA} in terms of average on time also for the set $C$ and $RC$. Table 5 presents the percentage gap in cost (‘gap’) and the ratio between the computational overhead of the heuristic approaches and the execution time required by the \textit{GSSAA} (‘speed up’). The results show that all the rollout-based approaches are competitive, particularly, in terms of computational times. However, the $2\mathcal{S}\mathcal{L}(\mathcal{F}(\mathcal{R}\mathcal{H}_3) + (\mathcal{R}\mathcal{H}_2))$ and $2\mathcal{S}\mathcal{L}(\mathcal{F}(\mathcal{R}\mathcal{H}_5))$ versions are highly competitive in terms of both computational effort and cost.

To evaluate the performances of the proposed algorithms, we carried out a second series of tests, by considering the instances belonging to the set $S2$. Then, we compared the solutions found by the rollout approaches with those determined by using the exact approach proposed in [29], that is, \textit{BAC}. For the sake of comprehension, the computational results

| Strategy                     | C AVG time | C AVG cost | R AVG time | R AVG cost | RC AVG time | RC AVG cost |
|------------------------------|------------|------------|------------|------------|-------------|-------------|
| \textit{GSSAA}              | 15,981,707.00 | −17.00     | 19,544.33  | −13.00     | 476,766.25  | 33.50       |
| $N\mathcal{H}\mathcal{R}$   | 17.22      | −10.89     | 9.58       | −12.83     | 5.50        | 40.00       |
| $\mathcal{R}\mathcal{H}_1$ | 30.78      | −11.44     | 9.58       | −12.83     | 5.38        | 39.75       |
| $\mathcal{R}\mathcal{H}_2$ | 24.78      | −11.44     | 14.08      | −12.83     | 16.25       | 39.75       |
| $\mathcal{R}\mathcal{H}_3$ | 23.33      | −10.67     | 21.67      | −12.83     | 21.63       | 39.88       |
| $\mathcal{F}(\mathcal{R}\mathcal{H}_3)$ | 39.67 | −11.11 | 26.25 | −12.83 | 29.50 | 40.00 |
| $\mathcal{F}(\mathcal{R}\mathcal{H}_3) + (\mathcal{R}\mathcal{H}_2)$ | 39.11 | −11.11 | 22.58 | −12.83 | 31.25 | 39.75 |
| $2\mathcal{S}\mathcal{L}(\mathcal{F}(\mathcal{R}\mathcal{H}_5))$ | 145.78 | −13.22 | 96.50 | −12.83 | 87.50 | 36.00 |
| $2\mathcal{S}\mathcal{L}(\mathcal{F}(\mathcal{R}\mathcal{H}_5) + (\mathcal{R}\mathcal{H}_2))$ | 193.56 | −13.67 | 114.00 | −12.83 | 119.38 | 36.00 |
| $2\mathcal{S}\mathcal{L}(\mathcal{F}(\mathcal{R}\mathcal{H}_5))$ | 139.44 | −9.56 | 35.17 | −9.75 | 25.63 | 41.00 |
| $2\mathcal{S}\mathcal{L}(\mathcal{F}(\mathcal{R}\mathcal{H}_5) + (\mathcal{R}\mathcal{H}_2))$ | 131.78 | −13.67 | 93.75 | −12.83 | 39.13 | 36.00 |
Table 5. Solution of the instances belonging to S1 by GSS, AA and rollout strategies.

| Strategy          | C  |          | R  |          | RC |          |
|-------------------|----|----------|----|----------|----|----------|
|                   | Gap (%) | Speed up | Gap (%) | Speed up | Gap (%) | Speed up |
| $N'HR$            | 36  | 2039.41  | 1   | 86,684.77 | 19 | 86,684.77 |
| $R'H_1$           | 33  | 2039.41  | 1   | 88,700.70 | 19 | 88,700.70 |
| $R'H_2$           | 33  | 1387.76  | 1   | 29,339.46 | 19 | 29,339.46 |
| $R'H_3$           | 37  | 902.05   | 1   | 22,046.99 | 19 | 22,046.99 |
| $F(R'H_3)$        | 35  | 744.55   | 1   | 16,161.57 | 19 | 16,161.57 |
| $F((R'H_3) + (R'H_3))$ | 35 | 865.43   | 1   | 15,256.52 | 19 | 15,256.52 |
| $2SLL(F(R'H_3))$  | 22  | 202.53   | 1   | 5448.76   | 7  | 5448.76   |
| $2SLL(F(R'H_3) + (R'H_3))$ | 20 | 171.44   | 1   | 3993.85   | 7  | 3993.85   |
| $2SLL(F(R'H_3))$  | 44  | 555.76   | 25  | 18,605.51 | 22 | 18,605.51 |
| $2SLL(F(R'H_3))$  | 22  | 259.15   | 1   | 14,178.92 | 7  | 14,178.92 |
| $2SLL(F(R'H_3) + (R'H_3))$ | 20 | 208.47   | 1   | 12,185.72 | 7  | 12,185.72 |

Table 6. Average of the solution values obtained by solving the instance belonging to S2 by B, AC and rollout strategies.

| Strategy          | A  |          | B  |          | E  |          |
|-------------------|----|----------|----|----------|----|----------|
|                   | AVG time | AVG cost | AVG time | AVG cost | AVG time | AVG cost |
| $B, AC$           | 10,340.00 | -37,171.80 | 12,616.25 | -144,130.00 | 14,581.67 | -9218.83 |
| $N'HR$            | 154.20 | -20,257.10 | 64.88  | -18,909.63 | 165.17 | -3594.17 |
| $R'H_1$           | 155.80 | -22,642.50 | 66.25  | -19,909.63 | 184.67 | -3760.83 |
| $R'H_2$           | 161.80 | -22,642.50 | 71.75  | -19,909.63 | 186.67 | -3927.50 |
| $R'H_3$           | 254.60 | -23,345.60 | 68.38  | -114,733.50 | 69.67 | -4549.50 |
| $F(R'H_3)$        | 75.30  | -24,749.00 | 82.88  | -82,164.63 | 203.17 | -4543.00 |
| $F((R'H_3) + (R'H_3))$ | 69 | -24,749.00 | 72.50  | -82,164.63 | 190.33 | -4543.00 |
| $2SLL(F(R'H_3))$  | 409.90 | -28,203.30 | 436.00 | -80,775.00 | 1264.83 | -7083.67 |
| $2SLL(F(R'H_3))$  | 410.30 | -28,303.30 | 436.25 | -93,525.00 | 1265.17 | -7083.67 |
| $2SLL(F(R'H_3))$  | 630.10 | -31,462.10 | 583.88 | -129,903.63 | 1025.33 | -7415.17 |
| $2SLL(F(R'H_3))$  | 617.20 | -29,162.00 | 649.00 | -86,126.50 | 1434.83 | -7083.67 |
| $2SLL(F(R'H_3) + (R'H_3))$ | 617 | -29,862.00 | 649.00 | -86,876.50 | 1434.83 | -7083.67 |

Table 6. Average of the solution values obtained by solving the instance belonging to S2 by B, AC and rollout strategies.

| Strategy          | G  |          | M  |          | P  |          |
|-------------------|----|----------|----|----------|----|----------|
|                   | AVG time | AVG cost | AVG time | AVG cost | AVG time | AVG cost |
| $B, AC$           | 53,000.00 | -1,426,535.00 | 27,308.00 | -118,544.60 | 2541.33 | -27,215.40 |
| $N'HR$            | 6202.00 | -676,288.00 | 998.40  | -59,501.20 | 146.87  | -15,666.80 |
| $R'H_1$           | 6207.00 | -683,288.00 | 1021.80 | -61,301.20 | 150.67  | -15,800.13 |
| $R'H_2$           | 6207.00 | -683,288.00 | 977.00  | -61,701.20 | 157.60  | -15,933.47 |
| $R'H_3$           | 9424.00 | -945,496.00 | 1195.60 | -75,457.80 | 96.27   | -18,311.27 |
| $F(R'H_3)$        | 14,492.00 | -1,066,600.00 | 913.00  | -76,439.40 | 172.27  | -18,311.27 |
| $F((R'H_3) + (R'H_3))$ | 13,342.00 | -1,099,600.00 | 985.60  | -80,039.40 | 168.13  | -18,311.27 |
| $2SLL(F(R'H_3))$  | 687,045.00 | -992,482.00 | 41,916.00 | -78,930.80 | 1042.27 | -21,752.73 |
| $2SLL(F(R'H_3) + (R'H_3))$ | 687,048.00 | -1,021,482.00 | 41,917.80 | -80,330.80 | 1042.53 | -22,152.73 |
| $2SLL(F(R'H_4))$  | 666,592.00 | -992,482.00 | 50,209.80 | -86,796.20 | 2169.67 | -23,793.27 |
| $2SLL(F(R'H_3))$  | 601,018.00 | -726,110.00 | 46,691.80 | -87,048.60 | 4167.80 | -22,703.53 |
| $2SLL(F(R'H_3) + (R'H_3))$ | 601,018.00 | -741,110.00 | 46,691.80 | -88,248.60 | 4167.80 | -23,036.87 |

presented in [29] have been reported in Table A1 of the Appendix of the supplemental material and used for comparison. Tables A5 and A6, in the Appendix of the supplemental material, resume the details of the results collected by solving the instances belonging to the set S2. Table 6, presents the comparison results for the instances belonging to set S2.

Focusing our attention to the classes $A$, $B$ and $E$, the results show that the use of more sophisticated algorithms leads an increase of the computational times. However, the most
Table 7. Time in milliseconds spent to find the optimal solution for the instances $P$ belonging to $S_2$.

| Test      | $2SL(F(RH_4))$ | $BAC$   | Speed up |
|-----------|-----------------|---------|----------|
| P-n50-k7-92 | 9.00            | 2420.00 | 268.89   |
| P-n51-k10-30 | 2.00            | 2180.00 | 1090.00  |
| P-n55-k7-116 | 544.00          | 580.00  | 1.07     |
| P-n60-k10-24 | 145.00          | 1950.00 | 13.45    |
| P-n60-k15-8  | 45.00           | 6650.00 | 147.78   |

Sophisticated algorithms yield good results in terms of solution quality. The best performing strategy is $2SL(F(RH_4))$ version. Indeed, $2SL(F(RH_4))$ is 16.41, 21.61 and 14.22 times faster than $BAC$ with an optimality gap of 15%, 10% and 20% for the instances of the classes $A$, $B$ and $E$, respectively.

Similar to the behaviour observed for the classes $A$, $B$ and $E$, the results collected for the classes $G$, $M$ and $P$ (see Table 6) highlight an increase in the computational time when more sophisticated versions of the algorithm are used. The look-ahead strategy is the most time consuming approach for this set of instances. However, only for a limited number of tests, about the 37%, the look-ahead strategies are not competitive in terms of time spent to find a solution. It is worth to observe that, we are able to find the optimal solution with $2SL(F(RH_4))$ strategy for five instances belonging to the class $P$ in a very limited amount of time (see Table 7).

5.5. Final remarks

We carried out the experimental phase with the aim of both comparing the proposed rollout approach with the state-of-the-art and evaluating the performance of the different versions of the proposed solution strategy.

Referring to the comparison with the state-of-the-art, the collected computational results highlight the potential of the proposed rollout algorithm to solve the $RCESPP$ in an efficient and effectiveness way. Indeed, the rollout approach explores a larger portion of solution space than $TS$, determining a higher number of paths, with negative reduced cost, characterized by an objective function value better than that obtained by $TS$. This is a promising result in view of an implementation of the rollout algorithm for the $RCESPP$ embedded in a column-generation approach for solving the set-partitioning formulation of the $VRP$. In addition, even though the proposed rollout algorithm is slightly slower than $TS$ for some instances, it does not require the tuning of the parameters that is mandatory for the $TS$. This is an important enhancement since the rollout has the potential to be applied with success to solve any kind of instances. This is not true for the $TS$ unless an in depth tuning of the parameters is performed.

Considering the comparison among the several defined versions of the rollout approach, the computational results highlight that no version dominates the others. They have similar performance and, as expected, the more sophisticated ones provide better solutions in term of objective function value but they are more expensive in terms of computational cost. In addition, the version $2SL(F(RH_4))$ is able to provide the optimal solution for some instances by paying a higher computational overhead. Thus, there exists a trade-off between quality of the solution and computational cost. In this respect, we can not conclude
that one version is better than the others, rather we leave the reader to choose the version that best fit his/her application based on the in depth analysis carried out in Section 5.4.

6. Conclusions

In this work, we have presented new approaches to address the \textit{RCESPP}. The proposed solution strategies are based on a rollout metaheuristic\cite{7,24} combined with local search procedures.

The developed approach is tested on benchmark instances. We carried out a comparative analysis with the state-of-the-art tabu search algorithm proposed in [1].

The computational results highlight the effectiveness of the proposed strategy. Indeed, on average, it is able to determine a better upper bound than that provided by the tabu search. In addition, the execution time slightly increases with respect the computational overhead of the tabu search.

The search process of the proposed rollout algorithm allows to determine a number of solutions with negative cost higher than that found by the tabu search. These results suggest a better behaviour of the proposed approach than the tabu search for pricing columns in a column-generation approach.

It would be interesting to embed the rollout algorithm in a column-generation approach. In addition, the rollout framework can be parallelized improving the performance in term of efficiency, speeding up the search process. These will be the subjects of future investigation.

Disclosure statement

No potential conflict of interest was reported by the authors.

ORCID

Francesca Guerriero \href{http://orcid.org/0000-0002-3887-1317}{http://orcid.org/0000-0002-3887-1317}
Luigi Di Puglia Pugliese \href{http://orcid.org/0000-0002-6895-1457}{http://orcid.org/0000-0002-6895-1457}

References

[1] C. Archetti, M. Bouchard, and G. Desaulniers, \textit{Enhanced branch and price and cut for vehicle routing with split deliveries and time windows}, Transp. Sci. 45(3) (2011), pp. 285–298.
[2] R. Baldacci, A. Mingozzi, and R. Roberti, \textit{New route relaxation and pricing strategies for the vehicle routing problem}, Oper. Res. 59(5) (2011), pp. 1269–1283.
[3] C. Barnhart, E.L. Johnson, G.L. Nemhauser, M.W.P. Savelsbergh, and P.H. Vance, \textit{Branch-and-price: Column generation for solving huge integer programs}, Oper. Res. 46(6) (1998), pp. 316–329.
[4] J.E. Beasley and N. Christofides, \textit{An algorithm for the resource constrained shortest path problem}, Networks 19 (1989), pp. 379–394.
[5] D.P. Bertsekas, \textit{Dynamic programming and suboptimal control: A survey from \{ADP\} to mpc*}, Eur. J. Control 11(4–5) (2005), pp. 310–334.
[6] D.P. Bertsekas and J.N. Tsitsiklis, \textit{Neuro-Dynamic Programming}, Athena Scientific, Belmont, MA, 1996.
[7] D.P. Bertsekas, J.N. Tsitsiklis, and C. Wu, \textit{Rollout algorithms for combinatorial optimization problems}, J. Heuristics 3 (1997), pp. 245–262.
[8] C. Bode and S. Irnich, *In-depth analysis of pricing problem relaxations for the capacitated arc-routing problem*, Transp. Sci. 49(2) (2015), pp. 369–383.

[9] N. Boland, J. Dethridge, and I. Dumitrescu, *Accelerated label setting algorithms for the elementary resource constrained shortest path problem*, Oper. Res. Lett. 34(1) (2006), pp. 58–68.

[10] O. Bräysy and M. Gendreau, *Vehicle routing problem with time windows, part I: Route construction and local search algorithms*, Transp. Sci. 39(1) (2005), pp. 104–118.

[11] N. Christofides, A. Mingozzi, and P. Toth, *Exact algorithms for the vehicle routing problem based on spanning tree and shortest path relaxations*, Math. Program. 20 (1981), pp. 255–282.

[12] C. Contardo, G. Desaulniers, and F. Lessard, *Reaching the elementary lower bound in the vehicle routing problem with time windows*, Networks 65(1) (2015), pp. 88–99.

[13] M. Dell’Amico, G. Righini, and M. Salani, *Branch-and-price approach to the vehicle routing problem with simultaneous distribution and collection*, Transp. Sci. 40(2) (2005), pp. 235–247A.

[14] G. Desaulniers, F. Lessard, and A. Hadjar, *Tabu search, generalized k-path inequalities, and partial elementarity for the vehicle routing problem with time windows*, Transp. Sci. 42(3) (2006), pp. 387–404.

[15] M. Desrochers, *La Fabrication d’horaires de travail pour les conducteurs d’autobus par une méthode de génération de colonnes*. PhD thesis, Centre de recherche sur les Transports, Canada, 1996.

[16] L. Di Puglia Pugliese and F. Guerriero, *A computational study of solution approaches for the resource constrained elementary shortest path problem*, Ann. Oper. Res. 201(1) (2012), pp. 131–157.

[17] L. Di Puglia Pugliese and F. Guerriero, *A reference point approach for the resource constrained shortest path problems*, Transp. Sci. 47(2) (2013), pp. 247–265.

[18] L. Di Puglia Pugliese and F. Guerriero, *A survey on resource constrained shortest path problems: Exact solution approaches*, Networks 62(3) (2013), pp. 183–200.

[19] M. Dror, *Note on the complexity of the shortest path models for column generation in vrptw*, Oper. Res. 42 (1994), pp. 977–978.

[20] D. Feillet, P. Dejax, M. Gendreau, and C. Guéguen, *An exact algorithm for the elementary shortest path problem with resource constraints: Application to some vehicle routing problems*, Networks 43(3) (2004), pp. 216–229.

[21] F. Guerriero, *Hybrid rollout approaches for the job shop scheduling problem*, J. Optim. Theory Appl. 139(2) (2008), pp. 419–438.

[22] F. Guerriero and M. Mancini, *A cooperative parallel rollout algorithm for the sequential ordering problem*, Parallel Comput. 29(5) (2003), pp. 663–677.

[23] F. Guerriero and M. Mancini, *Parallelization strategies for rollout algorithms*, Comput. Optim. Appl. 31(2) (2005), pp. 221–244.

[24] F. Guerriero, M. Mancini, and R. Musmanno, *New rollout algorithms for combinatorial optimization problems*, Optim. Methods Softw. 17(4) (2002), pp. 627–654.

[25] D.J. Houck, J.C. Picard, M. Queyranne, and R.R. Vemuganti, *The travelling salesman problem as a constrained shortest path problem: Theory and computational experience*, Oper. Res. 17 (1980), pp. 93–109.

[26] S. Irnich, *Resource extension functions: Properties, inversion and generalization to segments*, OR Spectrum 30(1) (2008), pp. 113–148.

[27] S. Irnich and G. Desaulniers, *Shortest path problems with resource constraints*, Networks 2 (2005), pp. 33–65.

[28] S. Irnich and D. Villeneuve, *The shortest-path problem with resource constraints and k-cycle elimination for k ≥ 3*, INFORMS J. Comput. 18(3) (2006), pp. 391–406.

[29] M.K. Jepsen, B. Petersen, and S. Spoorendonk, *A branch-and-cut algorithm for the elementary shortest path problem with a capacity constraint*. Technical Report 08/01 ISSN: 0107-8283, Dept. of Computer Science, University of Copenhagen, 2008.

[30] N. Kohl, *Exact methods for time constrained routing and related scheduling problems*. PhD thesis, Institute of Mathematical Modelling, Technical University of Denmark, DK-2800 Lyngby, 1995. Dissertation no. 16.
[31] F. Liberatore, G. Righini, and M. Salani, A column generation algorithm for the vehicle routing problem with soft time windows, 4OR 9(1) (2011), pp. 49–82.

[32] D. Pecin, C. Contardo, G. Desaulniers, and E. Uchoa, New enhancements for the exact solution of the vehicle routing problem with time windows, INFORMS J. Comput. 29(3) (2017), pp. 489–502.

[33] D. Pecin, A. Pessoa, M. Poggi, and E. Uchoa, Improved branch-cut-and-price for capacitated vehicle routing, Math. Program. Comput. 9(1) (2016), pp. 1–40.

[34] S. Pirkwieser and G.R. Raidl, Column generation approach for the periodic vehicle routing problem with time windows, Proceeding of the International Network Optimization Conference, 2009.

[35] G. Righini and M. Salani, Symmetry helps: Bounded bidirectional dynamic-programming for the elementary shortest path problem with resource constraints, Discrete Optim. 3(3) (2006), pp. 255–273.

[36] M. Salani and I. Vacca, Branch and price for the vehicle routing problem with discrete split deliveries and time windows, Eur. J. Oper. Res. 213 (2011), pp. 213–477.