TRANSPORTATION SCIENCE
Articles in Advance, pp. 1–10
ISSN 0041-1655 (print) | ISSN 1526-5447 (online)

http://dx.doi.org/10.1287/trsc.2014.0582
© 2015 INFORMS

An Exact Algorithm for the Elementary Shortest Path Problem with Resource Constraints

Leonardo Lozano, Daniel Duque, Andrés L. Medaglia
Centro para la Optimización y Probabilidad Aplicada (COPA),
Departamento de Ingeniería Industrial, Universidad de los Andes, Bogotá, Colombia
{leo-loza@uniandes.edu.co, d.duque25@uniandes.edu.co, amedagli@uniandes.edu.co}

The elementary shortest path problem with resource constraints (ESPPRC) is an NP-hard problem that often arises in the context of column generation for vehicle routing problems. We propose an exact solution method that relies on implicit enumeration with a novel bounding scheme that dramatically narrows the search space. We embedded our algorithm within a column generation to solve the linear relaxation (root node) of the vehicle routing problem with time windows (VRPTW) and found that the proposed algorithm performs well when compared against state-of-the-art algorithms for the ESPPRC on the well-known Solomon's test bed for the VRPTW.

Keywords: vehicle routing; shortest path problem; column generation; vehicle routing problem with time windows
History: Received: January 2013; revision received: July 2013; accepted: September 2014. Published online in *Articles in Advance*.

1. Introduction

The elementary shortest path problem with resource constraints (ESPPRC) (Feillet et al. 2004, Irnich and Desaulniers 2005) arises as the backbone of branch-and-price procedures for several variants of the vehicle routing problem (VRP). One of the most widely known VRP variants closely tied to the ESPPRC is the vehicle routing problem with time windows (VRPTW) (Toth and Vigo 2002). The VRPTW consists of finding a set of routes of minimal total cost such that each customer in $\mathcal{N} = \{v_1, \ldots, v_i, \ldots, v_n\}$ is served (exactly once) within its time window $[a_i, b_i]$. The vehicles in the VRPTW begin and end their routes at the depot and are part of an unlimited homogeneous capacitated fleet. Under a column generation framework, it is common practice to use a set covering formulation, where Ω is the set of all feasible routes for the VRPTW and α_{ik} is a binary indicator that takes the value of 1 if route $k \in \Omega$ visits customer $v_i \in \mathcal{N}$ and takes the value of 0 otherwise. Let c_k denote the cost of route $k \in \Omega$ and let x_k be a binary variable that takes the value of 1 if route $k \in \Omega$ is used and 0 otherwise. The VRPTW can be formulated as follows:

$$\text{minimize} \quad \sum_{k\in\Omega} c_k x_k \tag{1}$$

$$\text{subject to} \quad \sum_{k\in\Omega} \alpha_{ik} x_k \geq 1 \quad \forall\, v_i \in \mathcal{N}, \tag{2}$$

$$x_k \in \{0, 1\} \quad \forall\, k \in \Omega. \tag{3}$$

The objective function (1) minimizes the total cost, the set covering constraints (2) ensure that each customer is visited by a route, and constraints (3) ensure that the decision variables are binary. Under a column generation scheme (Desrosiers and Lübbecke 2005), a restricted master problem solves the linear relaxation of the model considering a small subset of its variables. The ESPPRC arises as the subproblem of finding feasible routes (columns) with negative reduced cost that are iteratively added to the restricted master problem.

For the ESPPRC subproblem, consider a directed graph $\mathcal{G} = (\mathcal{N} \cup \{v_s, v_e\}, \mathcal{A})$, where v_s and v_e represent the depot and $\mathcal{A} = \{(v_i, v_j) \mid v_i \in \mathcal{N} \cup \{v_s\}, v_j \in \mathcal{N} \cup \{v_e\}, v_i \neq v_j\}$ denotes the set of arcs. Each node $v_i \in \mathcal{N}$ represents a customer with demand q_i, service time s_i, and a dual multiplier λ_i, associated with the corresponding set covering constraint (2) in the master problem. Each arc $(v_i, v_j) \in \mathcal{A}$ has a distance d_{ij}, a time t_{ij} that includes the service time s_i of node v_i, and a reduced cost contribution $r_{ij} = d_{ij} - \lambda_i$. We assume that the arc time and distance satisfy the triangle inequality, thus the set of arcs can be reduced using the time windows (Desrochers, Desrosiers, and Solomon 1992). Note that a path from start node $v_s \in \mathcal{N}$ to end node $v_e \in \mathcal{N}$ corresponds to a column (variable) in the restricted master problem, representing route $k \in \Omega$ with reduced cost $r_k = \sum_{(v_i, v_j)\in k} r_{ij}$. The ESPPRC consists of finding an elementary path $\mathcal{P}$ (i.e., ordered sequence of nonrepeating nodes) from start node $v_s \in \mathcal{N}$ to end node $v_e \in \mathcal{N}$ that minimizes the reduced cost subject to a vehicle capacity constraint and the time-window constraints over the nodes. The capacity constraint establishes an upper limit Q to the sum of the demands q_i collected by a vehicle along the path. Time-window constraints are given by intervals $[a_i, b_i]$ that restrict the beginning of the service for the customer located at node $v_i \in \mathcal{N}$. If a vehicle traversing the path visits node $v_i \in \mathcal{N}$ before the early time window a_i, it must wait until a_i to begin service; but, it cannot serve node $v_i \in \mathcal{N}$ if it arrives after time b_i.

Accelerating the solution of the ESPPRC subproblem is a critical aspect for improving the performance of column generation-based algorithms in several routing applications such as the VRPTW. Hence, this paper expands the body of knowledge on the ESPPRC by presenting a new exact method for the ESPPRC that extends the ideas proposed by Lozano and Medaglia (2013) for the constrained shortest path problem (Joksch 1966) using a novel bounding scheme. We embedded our algorithm into a column generation procedure for the VRPTW that performed remarkably well when solving the linear relaxation of the problem at the root node. When compared against the state-of-the-art algorithm by Baldacci, Mingozzi, and Roberti (2011a), our algorithm is on average faster in 20 of the 29 instances of the ESPPRC; and our column generation finds tighter bounds in less computational time for 38 of the 48 reported instances from the Solomon's test bed for the VRPTW.

The remainder of this paper is organized as follows. Section 2 presents a literature review for the ESPPRC. Section 3 outlines the algorithm's intuition. Section 4 describes in detail the proposed pruning strategies. Section 5 presents the computational experiments. Finally, §6 concludes the paper and outlines future extensions.

2. Literature Review

Since the ESPPRC is NP-hard in the strong sense (Dror 1994), the seminal work by Desrochers, Desrosiers, and Solomon (1992) solved a relaxed version that allows cycles with a dynamic programming (DP) approach; however, such relaxation leads to weak lower bounds and large branch-and-bound trees (Irnich and Villeneuve 2006). Feillet et al. (2004) proposed the first exact approach for the ESPPRC, a labeling algorithm that extends the one proposed by Desrochers, Desrosiers, and Solomon (1992). The algorithm includes a customer resource that indicates if a given customer can be visited or not by extending the current partial path. After embedding their algorithm in a column generation procedure, Feillet et al. (2004) showed that using the ESPPRC improves the lower bounds obtained at each node of the branch-and-price tree. Rousseau et al. (2004) solved the VRPTW using a branch-and-price approach that handled the ESPPRC subproblem with constraint programming (CP). Although the CP component proved to be flexible, their approach was somewhat slow in comparison with traditional branch-and-price strategies. Following a very similar procedure to the one proposed by Feillet et al. (2004), Chabrier (2006) embedded the ESPPRC into a branch-and-price scheme for the VRPTW and found optimal solutions to 17 previously open instances of the Solomon's test bed. Later, Feillet, Gendreau, and Rousseau (2007) proposed additional refinements to reduce the computing time of the branch-and-price procedure with the ESPPRC subproblem scheme. More recently, Righini and Salani (2008) proposed a bidirectional labeling algorithm for the ESPPRC that relies on a state-space relaxation. This algorithm considerably outperforms the DP algorithms by Feillet et al. (2004) and Chabrier (2006). A very similar DP algorithm was proposed by Boland, Dethridge, and Dumitrescu (2006) achieving remarkably good performance on randomly generated instances. Lately, Jepsen et al. (2008) and Petersen, Pisinger, and Spoorendonk (2008) introduced subset-row inequalities that significantly improved the lower bounds obtained at each node of the branch-and-price tree, but increased the complexity of the pricing problem. Desaulniers, Lessard, and Hadjar (2008) used these inequalities and combined the exact algorithm by Righini and Salani (2008) with a tabu search for the subproblem; they obtained optimal solutions for five open instances (at that time) of the Solomon test bed. Aside from branch-and-price methods, Baldacci et al. (2010) proposed a dual ascent procedure, that combined with a column-and-cut generation algorithm, outperformed all exact methods published so far and provided optimal solutions to several open VRP instances. Baldacci, Mingozzi, and Roberti (2011a) extended the exact algorithm of Righini and Salani (2008) by including bounding functions based on state-space relaxation. They introduced the concept of *ng*-route relaxation that is used to calculate completion bounds for partial paths, thus accelerating the DP algorithm by means of label fathoming. For a comprehensive review on resource constrained shortest paths we refer the reader to the survey presented by Di Puglia Pugliese and Guerriero (2013) and for recent advances in algorithms for the VRPTW we refer the reader to Baldacci, Mingozzi, and Roberti (2012).

3. Pulse Algorithm: Overview and Intuition

We name our algorithm after the simple intuition that lies behind. To explain this intuition we appeal to the analogy of a pulse propagating through the graph. Formally, *pulse propagation* refers to the recursive exploration of partial paths that are extended until they reach the end node or are discarded by a pruning strategy. For the ESPPRC, the *pulse algorithm* comprises two stages: (1) a bounding stage that finds lower bounds on the cost given an amount of resource consumed, and (2) a recursive exploration stage that finds an optimal solution based on an implicit enumeration of the solution space. The exploration is triggered by sending a pulse from the start node $v_s \in \mathcal{N}$. The pulse tries to propagate throughout the outgoing arcs of each visited node, storing at each node the partial path $\mathcal{P}$, the cumulative reduced cost $r(\mathcal{P})$, the cumulative capacity consumption $q(\mathcal{P})$, and the cumulative time consumption $t(\mathcal{P})$. At each node different pruning strategies try to stop the pulse propagation, aggressively pruning the search space. Every pulse that reaches the final node $v_e \in \mathcal{N}$ contains all of the information of a feasible path $\mathcal{P}$ from v_s to v_e. Algorithm 1 presents an overview of the pulse algorithm. Lines 1 to 4 set the initial values for the partial path $\mathcal{P}$, the cumulative reduced cost $r(\mathcal{P})$, and the cumulative resources consumption $q(\mathcal{P})$ and $t(\mathcal{P})$. Line 5 runs the bounding procedure that finds lower bounds for every node in $\mathcal{N}$ (see §4.2). Finally, line 6 invokes the recursive procedure `pulse` starting the propagation from node $v_s \in \mathcal{N}$ and line 7 returns an optimal path $\mathcal{P}^*$ found in the recursion.

递归的

Algorithm 1 (Pulse algorithm)

```
Input: $\mathcal{G}$, directed graph; $v_s$, start node; $v_e$, end node;
       $\Delta$, bound step size; $[\underline{t}, \bar{t}]$, bounding time limits.
Output: $\mathcal{P}^*$, optimal path.
1: $\mathcal{P} \leftarrow \{\}$
2: $r(\mathcal{P}) \leftarrow 0$
3: $q(\mathcal{P}) \leftarrow 0$
4: $t(\mathcal{P}) \leftarrow 0$
5: bound($\mathcal{G}, \Delta, [\underline{t}, \bar{t}]$)                        ▷ see §4.2
6: pulse($v_s, r(\mathcal{P}), q(\mathcal{P}), t(\mathcal{P}), \mathcal{P}$)        ▷ see Algorithm 2
7: return $\mathcal{P}^*$
```

Algorithm 2 shows the recursive procedure `pulse`, where $\Gamma^+(v_i) = \{v_j \in \mathcal{N} \cup \{v_e\} \mid (v_i, v_j) \in \mathcal{A}\}$ is the set of head nodes of the outgoing arcs from node v_i. Lines 1 to 3 use the pruning strategies, namely *infeasibility*, *bounds*, and *rollback*, in order to prune the incoming pulse. The infeasibility pruning strategy ensures the elementarity of the paths. If the pulse is not pruned, line 4 adds the current node to the partial path being explored and line 5 updates the vehicle capacity. Lines 6 to 10 propagate the pulse by invoking the `pulse` procedure over all nodes $v_j \in \Gamma^+(v_i)$. Every time the `pulse` procedure is invoked on the final node v_e, the information for the best-known path $\mathcal{P}^*$ is updated (if needed), the pulse stops its propagation, and the algorithm backtracks to propagate other pulses recursively.

Algorithm 2 (Pulse procedure)

```
Input: $v_i$, current node; $r(\mathcal{P})$, path reduced cost;
       $q(\mathcal{P})$, path load; $t(\mathcal{P})$, path time; $\mathcal{P}$,
       current path.
Output: void
 1: if isFeasible($v_i, q(\mathcal{P}), t(\mathcal{P})$) = true then
                                                        ▷ see §4.1
 2:   if checkBounds($v_i, t(\mathcal{P}), r(\mathcal{P})$) = false then
                                                        ▷ see §4.2
 3:     if rollback($v_i, t(\mathcal{P}), r(\mathcal{P}), \mathcal{P}$) = false then
                                                        ▷ see §4.3
 4:       $\mathcal{P}' \leftarrow \mathcal{P} \cup \{v_i\}$
 5:       $q(\mathcal{P}') \leftarrow q(\mathcal{P}) + q_i$
 6:       for $v_j \in \Gamma^+(v_i)$ do
 7:         $r(\mathcal{P}') \leftarrow r(\mathcal{P}) + r_{ij}$
 8:         $t(\mathcal{P}') \leftarrow \max\{a_j, t(\mathcal{P}) + t_{ij}\}$
 9:         pulse ($v_j, r(\mathcal{P}'), q(\mathcal{P}'), t(\mathcal{P}'), \mathcal{P}'$)   递归
10:       end for
11:     end if
12:   end if
13: end if
```

From this overview of the proposed algorithm we would like to emphasize the following main differences between the pulse algorithm and traditional DP algorithms:

- *Dominance rules*. The pulse algorithm does not rely on dominance rules that compare partial paths (represented by labels). Although this lack of dominance rules may lead to a significant increase in the number of explored paths, it is also true that the pulse does not have to handle an often long list of ordered labels.
- *Depth-first versus breadth-first search*. As the pulse algorithm is recursive, it explores the graph in a depth-first search manner, whereas most labeling algorithms follow a lexicographic breadth-first search. Accordingly, one would expect the pulse algorithm to find feasible solutions faster than DP algorithms. The pulse algorithm can also be seen as a branch and bound, where each node corresponds to an elementary partial path.
- *Pruning strategies*. Although infeasibility pruning is identical for partial paths/labels in pulse/labeling algorithms, the pulse algorithm heavily relies on additional pruning strategies that are not always included in labeling algorithms.

• *Bidirectional search.* Because of its recursive nature, the pulse algorithm cannot apply bidirectional search techniques that are rather useful for labeling algorithms (Righini and Salani 2008).

It is worth highlighting that every time that a pulse is pruned, a whole entire region of the solution space is discarded and not just a single solution. Thus, the algorithm's performance is strongly linked to the strength of the pruning strategies and their ability to 修剪 prune pulses at early stages of the exploration.

4. Pruning Strategies

We use three pruning strategies for the ESPPRC: the first one uses structural constraints (i.e., time-window, capacity, and cycle constraints) to prune infeasible solutions; the second one uses primal (i.e., best solution found) and lower bounds to prune suboptimal solutions; and the third one uses a look-back mechanism to discard suboptimal partial paths.

4.1. Infeasibility Pruning

Whenever a partial path reaches a node $v_i \in \mathcal{N}$, the algorithm checks if visiting the node creates a cycle, exceeds the capacity constraint, or violates the time windows. If any of these events happen, the partial path can be safely pruned because it is infeasible. Discarding infeasible partial paths is a common practice used in labeling algorithms (Desrochers, Desrosiers, and Solomon 1992, Boland, Dethridge, and Dumitrescu 2006).

To check for cycles, we use an indicator function that identifies the nodes already visited. Using this function, the algorithm checks in constant time if node v_i has already been visited by the path or not. Similarly, if $q(\mathcal{P}) > Q$ the vehicle's capacity has been exceeded by the demand of the customers visited along the partial path $\mathcal{P}$, thus the partial path is infeasible and pruned. Regarding the time windows, if $t(\mathcal{P}) > b_i$, we prune the partial path $\mathcal{P}$ because the node v_i is visited after the latest time; but, if $t(\mathcal{P}) < a_i$, then the vehicle must wait until time a_i to start service at node v_i and $t(\mathcal{P}) \leftarrow a_i$.

4.2. Bounds Pruning

Several authors have used bounding functions to fathom suboptimal partial paths. Baldacci, Mingozzi, and Roberti (2011a) propose the use of an *ng*-route relaxation that ignores the vehicle capacity constraints and imposes the elementarity condition only to a subset of nodes. Baldacci, Mingozzi, and Roberti (2011b) introduces additional bounding functions based on different relaxations.

Following the same spirit of bounding functions, we use a 最初的 primal bound $\bar{r}$ that is constantly updated with the value of the best solution found at any time of the exploration and propose a bounding scheme that finds conditional lower bounds $\underline{r}(v_i, t(\mathcal{P}))$ for every node $v_i \in \mathcal{N}$ and for discrete values of resource consumption $t(\mathcal{P})$. These bounds store the minimum reduced cost that can be achieved by any partial path $\mathcal{P}$ that reaches node $v_i \in \mathcal{N}$ with a given amount of consumed resource $t(\mathcal{P})$.

1. v_i与路径的关系是什么？
2. 需要求多少个值（v_i t(路径)组合）

The proposed bounding scheme works as follows. Let $\bar{t}$ be the upper time window at the depot and let Δ be a nonnegative time step. We start by solving an ESPPRC for every node $v_i \in \mathcal{N}$ given a time consumption of $t(\mathcal{P}) = \bar{t} - \Delta$. Given that there are only Δ units of time available, the resulting problems (one for each node) are overly constrained with few feasible solutions, so the `pulse` procedure can quickly sort through them solving the problems to optimality. Every optimal solution found is a lower bound on the minimum reduced cost that can be achieved by any partial path that reaches node v_i given a time consumption $t(\mathcal{P}) \geq \bar{t} - \Delta$. After finding these bounds, we then solve an ESPPRC for every node $v_i \in \mathcal{N}$ given a time consumption of $t(\mathcal{P}) = \bar{t} - 2\Delta$. Although the resulting problems are now less constrained, note that we already have computed lower bounds on the cost for paths with time consumption between $[\bar{t} - \Delta, \bar{t}]$. By using the information computed previously, we continue in a backward mode repeating the same procedure until reaching a given time limit $\underline{t}$. This procedure will result in a lower bound matrix denoted by $\mathbf{B} = [\underline{r}(v_i, \tau)]$ that contains the lower bounds calculated for every node and every discrete time step between $\underline{t}$ and $\bar{t}$. Note that as the resource available increases, the problems solved in the bounding scheme become less constrained and more difficult to solve, but also the amount of known bounds increases, augmenting the chances to prune partial paths with the bounds pruning strategy. Algorithm 3 shows the pseudo-code of the bounding scheme. Line 1 initializes the time consumption τ. Line 2 begins the bounding scheme and checks for the stopping condition. Line 3 updates the time consumption, subtracting the time step Δ at each iteration. Lines 4 to 9 solve an ESPPRC for every node given the time consumption using the pulse procedure. Lines 10 to 14 store the optimal value found for every node at position (v_i, τ) of the lower bound matrix $\mathbf{B}$, if a particular problem is infeasible (i.e., $\mathcal{P}^* = \{\}$) the lower bound is set to positive infinity. Note that these lower bounds solely focus on the time resource consumption and consider a (relaxed) initial capacity consumption of zero for the partial path $\mathcal{P}$ that reaches node $v_i \in \mathcal{N}$.

Algorithm 3 (Bounding scheme)

```
Input: $\mathcal{G}$, directed graph; $\Delta$, step size; $[\underline{t}, \bar{t}]$,
        bounding time limits.
Output: $\mathbf{B} = [\underline{r}(v_i, \tau)]$, lower bound matrix.
 1: $\tau \leftarrow \bar{t}$
 2: while $\tau > \underline{t}$ do
 3:     $\tau \leftarrow \tau - \Delta$
 4:     for $v_i \in \mathcal{N}$ do
 5:         $\mathcal{P} \leftarrow \{\}$
 6:         $r(\mathcal{P}) \leftarrow 0$
 7:         $q(\mathcal{P}) \leftarrow 0$
 8:         $t(\mathcal{P}) \leftarrow \tau$
 9:         pulse$(v_i, r(\mathcal{P}), q(\mathcal{P}), t(\mathcal{P}), \mathcal{P})$
10:         if $\mathcal{P}^* = \{\}$ then
11:             $\underline{r}(v_i, \tau) \leftarrow \infty$
12:         else
13:             $\underline{r}(v_i, \tau) \leftarrow r(\mathcal{P}^*)$
14:         end if
15:     end for
16: end while
17: return B
```

Note that under this bounding scheme, to calculate a lower bound for a given partial path $\mathcal{P}$ that visits a node $v_i \in \mathcal{N}$, the function `checkBounds` must use from the bound matrix **B** the lower closest value to $t(\mathcal{P})$ available. For example, consider a problem with $\bar{t} = 100$ and a time step defined as $\Delta := 10$. If a partial path arrives at a given node $v_i \in \mathcal{N}$ with 85 units of time consumed, then the corresponding lower bound will be the one found for node v_i and 80 units of time consumed. Note that the best reduced cost that can be achieved from a partial path with 80 units of time consumed will be a lower bound for the best reduced cost that can be achieved from a partial path with 85 units of time consumed. The function `checkBounds` will prune a partial path if $r(\mathcal{P}) + \underline{r}(v_i, t(\mathcal{P})) \geq \bar{r}$.

4.3. Rollback Pruning

One of the main drawbacks of depth-first search is that poor decisions made at early stages of the exploration can lead to the exploration of unpromising regions of the search space. It might take a while for the algorithm to backtrack and correct that poor initial choice. Aiming to avoid this behavior, this pruning strategy looks for some reassurance of the last choice made, or if necessary, it rolls back the change. Consider a partial path $\mathcal{P}_{si}$ from v_s to v_i that is extended to node v_k and then reaches node v_j. Once the partial path $\mathcal{P}_{sj} = \mathcal{P}_{si} \cup \{v_k\} \cup \{v_j\}$ reaches node v_j, the rollback pruning strategy reevaluates the last choice made, i.e, visiting node v_k before node v_j. Let $\mathcal{P}'_{sj} = \mathcal{P}_{si} \cup \{v_j\}$ be an alternative partial path to node v_j that does not visit node v_k. According to Feillet et al. (2004), if $\mathcal{P}'_{sj} \subseteq \mathcal{P}_{sj}$, $q(\mathcal{P}'_{sj}) \leq q(\mathcal{P}_{sj})$, $r(\mathcal{P}'_{sj}) \leq r(\mathcal{P}_{sj})$, $t(\mathcal{P}'_{sj}) \leq t(\mathcal{P}_{sj})$, and at least one of the four previous conditions holds strictly (e.g., $\mathcal{P}'_{sj} \subset \mathcal{P}_{sj}$ or $r(\mathcal{P}'_{sj}) < r(\mathcal{P}_{sj})$) it is said that path $\mathcal{P}_{sj}$ is dominated by path $\mathcal{P}'_{sj}$. Note that by construction, the alternative path $\mathcal{P}'_{sj}$ always satisfies $\mathcal{P}'_{sj} \subset \mathcal{P}_{sj}$ and $q(\mathcal{P}'_{sj}) \leq q(\mathcal{P}_{sj})$, so the rollback pruning strategy only needs to check if $r(\mathcal{P}'_{sj}) \leq r(\mathcal{P}_{sj})$ and $t(\mathcal{P}'_{sj}) \leq t(\mathcal{P}_{sj})$ to prune path $\mathcal{P}_{sj}$. This check does not require any kind of storage (e.g., labels), but just simple arithmetic calculations for $r(\mathcal{P}'_{sj})$ and $t(\mathcal{P}'_{sj})$. Figure 1 presents a graphical representation of paths $\mathcal{P}_{sj}$ and $\mathcal{P}'_{sj}$.

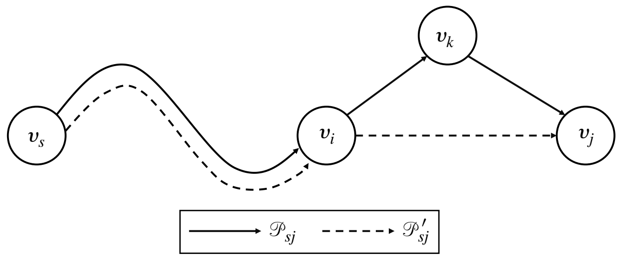

Figure 1 **Graphical Representation of Paths $\mathcal{P}_{sj}$ and $\mathcal{P}'_{sj}$ Used in the Rollback Pruning Strategy**

5. Computational Experiments

To assess the performance of the pulse algorithm, we embedded it inside a column generation scheme that solves the linear relaxation (root node) of the VRPTW as done by Feillet et al. (2004). First, we compared the pulse algorithm against the labeling state-of-the-art algorithm (GENR) by Baldacci, Mingozzi, and Roberti (2011a). Second, we compared the lower bounds obtained by our column generation scheme against Desaulniers, Lessard, and Hadjar (2008) and Baldacci, Mingozzi, and Roberti (2011a) in terms of the quality of the bounds and the time required to compute them. Finally, we conducted additional experiments to assess the performance of the proposed pruning strategies within the pulse algorithm.

In our column generation implementation, we used a conventional set covering formulation. To speed up the first iterations of the column generation–when there are plenty of promising columns–we followed a common practice that relies on heuristics (Desaulniers, Lessard, and Hadjar 2008, Chabrier 2006, Feillet, Gendreau, and Rousseau 2007). We coded a tabu search (TS) (Glover and Laguna 1997, Gendreau and Potvin 2010) along the lines of the one proposed by Desaulniers, Lessard, and Hadjar (2008). To diversify the search, Desaulniers, Lessard, and Hadjar (2008) start their TS from different initial solutions associated with the basic variables of the current restricted master problem. In addition, we execute $R_{\max}$ restarts from solutions randomly selected from the column pool, not necessarily those linked to the basic variables. Our TS starts with an initial solution and during a limited number of iterations $I_{\max}$ it

Table 1 **Comparing the Pulse Algorithm Inside a Column Generation Scheme Against Baldacci, Mingozzi, and Roberti (2011a) on the Solomon 100-Series Benchmark Problems**

	Baldacci, Mingozzi, and Roberti (2011a)		Column generation with pulse and TS			
Instance	Time (s)	LB_3	Time (s)	Scaled time (s)	Lower bound	Bound improvement (%)
r101.100	6.0	1,630.4	**1.6**	**2.3**	1,631.2	**0.05**
r102.100	7.0	1,466.2	**2.5**	**3.5**	1,466.6	**0.03**
r103.100	6.0	1,203.2	**3.9**	**5.5**	1,206.8	**0.30**
r104.100	12.0	955.7	**7.0**	**9.8**	956.9	**0.12**
r105.100	6.0	1,345.0	**2.9**	**4.1**	1,346.1	**0.08**
r106.100	7.0	1,225.7	**4.6**	**6.5**	1,226.9	**0.10**
r107.100	7.0	1,052.3	5.5	7.6	1,053.3	**0.09**
r108.100	10.0	913.0	7.4	10.4	913.5	**0.05**
r109.100	9.0	1,133.5	**4.2**	**5.9**	1,134.3	**0.07**
r110.100	9.0	1,054.9	**6.0**	**8.4**	1,055.6	**0.06**
r111.100	10.0	1,033.9	**5.6**	**7.9**	1,034.7	**0.08**
r112.100	11.0	926.2	**7.5**	**10.5**	926.7	**0.06**
rc101.100	5.0	1,581.1	**1.9**	**2.6**	1,584.1	**0.19**
rc102.100	8.0	1,405.5	**3.3**	**4.6**	1,406.3	**0.05**
rc103.100	8.0	1,224.5	**4.5**	**6.3**	1,225.5	**0.08**
rc104.100	35.0	1,100.4	**6.6**	**9.3**	1,101.8	**0.13**
rc105.100	7.0	1,471.7	**3.3**	**4.6**	1,471.9	**0.02**
rc106.100	8.0	1,318.5	**3.4**	**4.7**	1,318.8	**0.02**
rc107.100	8.0	1,182.7	**4.1**	**5.7**	1,183.4	**0.06**
rc108.100	11.0	1,073.0	**4.9**	**6.9**	1,073.4	**0.04**
c101.100	n/a	n/a	1.3	1.8	827.3	n/a
c102.100	8.0	827.3	**2.3**	**3.2**	827.3	0.00
c103.100	11.0	826.3	**6.0**	**8.4**	826.3	0.00
c104.100	31.0	822.7	30.8	43.2	822.9	**0.02**
c105.100	8.0	827.3	**1.5**	**2.1**	827.3	0.00
c106.100	10.0	827.3	**1.8**	**2.5**	827.3	0.00
c107.100	8.0	827.3	**2.0**	**2.8**	827.3	0.00
c108.100	8.0	827.3	**2.5**	**3.5**	827.3	0.00
c109.100	10.0	827.3	**4.1**	**5.7**	827.3	0.00

explores neighborhoods based on insertion and deletion operators.

We coded our algorithm in Java, using Eclipse SDK version 4.3.0, and executed the experiments on a laptop computer with an Intel Core i7-3537U CPU (2 cores) running at 2 GHz with 512 MB of RAM allocated to the memory heap size of the Java Virtual Machine on Windows 8. It is worth highlighting that we implemented a multithread version of the pulse algorithm following the ideas outlined in Lozano and Medaglia (2013), where a fixed number of threads are triggered at node v_s exploring different outgoing arcs of v_s in parallel. The code for the pulse algorithm is publicly available on request. To solve the master problem we used Gurobi 5.0.1 as the linear optimizer.

We tested our approach on the well-known Solomon's benchmark problems for the VRPTW. These problems are organized in three categories: the r-instances where customers are randomly distributed; the c-instances where customers are located in clusters, and the rc-instances where some customers are randomly located and others are clustered. All problems have 100 customers and are divided into two series depending on the tightness of the time windows.

5.1. Embedding the Pulse Algorithm in a Column Generation Scheme for the VRPTW

The approaches by Desaulniers, Lessard, and Hadjar (2008) and Baldacci, Mingozzi, and Roberti (2011a) both use the ESPPRC as a subproblem. Whereas Desaulniers, Lessard, and Hadjar (2008) relied on a standard column generation method based on simplex; Baldacci, Mingozzi, and Roberti (2011a) proposed a dual ascent heuristic to solve the master problem. Desaulniers, Lessard, and Hadjar (2008) executed their experiments on a Linux PC with a Dual Core AMD Opteron processor running at 2.6 GHz and Baldacci, Mingozzi, and Roberti (2011a) used an IBM Intel Xeon X7350 server running at 2.93 GHz with 16 GB of RAM. Aside from the operating system and programming language differences, according to the LINPACK benchmark (Dongarra 2013), our laptop computer is approximately 1.4 times faster than the computer used by Baldacci, Mingozzi, and Roberti (2011a) and about 2.8 times faster than the one used by Desaulniers, Lessard, and Hadjar (2008).

We conducted a few preliminary runs to fine-tune the parameters of the pulse algorithm. After these runs, we fixed $\Delta = 10$ and stopped the bounding scheme with $\underline{t} = 0.2\bar{t}$. For the TS heuristic, we

Table 2 **Comparing the Pulse Algorithm Inside a Column Generation Scheme Against Baldacci, Mingozzi, and Roberti (2011a) on the Solomon 200-Series Benchmark Problems**

	Baldacci, Mingozzi, and Roberti (2011a)		Column generation with pulse and TS			
Instance	Time (s)	LB_3	Time (s)	Scaled time (s)	Lower bound	Bound improvement (%)
r201.100	17.0	1,140.3	**9.3**	**13.0**	1,140.3	0.00
r202.100	22.0	1,021.2	18.0	25.2	1,022.2	**0.10**
r203.100	165.0	865.8	**72.4**	**101.4**	866.9	**0.13**
r204.100	194.0	724.2	**133.2**	**186.4**	724.9	**0.10**
r205.100	31.0	938.0	25.3	35.4	938.9	**0.10**
r206.100	76.0	866.3	**44.7**	**62.6**	866.9	**0.07**
r207.100	368.0	789.9	**127.5**	**178.6**	790.7	**0.10**
r208.100	2,405.0	690.3	**266.3**	**372.9**	692.0	**0.24**
r209.100	59.0	840.6	42.3	59.2	841.4	**0.10**
r210.100	57.0	888.2	**34.4**	**48.2**	889.4	**0.14**
r211.100	219.0	734.1	**76.8**	**107.6**	734.7	**0.08**
rc201.100	12.0	1,255.4	**6.2**	**8.6**	1,255.9	**0.04**
rc202.100	13.0	1,086.2	**7.1**	**9.9**	1,088.1	**0.17**
rc203.100	22.0	919.5	34.7	48.5	922.5	**0.33**
rc204.100	455.0	778.4	**322.7**	**451.7**	779.7	**0.17**
rc205.100	14.0	1,145.8	**7.7**	**10.8**	1,147.6	**0.16**
rc206.100	16.0	1,037.7	19.2	26.9	1,038.6	**0.09**
rc207.100	62.0	945.8	**42.8**	**59.9**	947.3	**0.16**
rc208.100	168.0	765.8	441.9	618.6	766.7	**0.12**
c201.100	n/a	n/a	2.4	3.4	589.1	n/a
c202.100	n/a	n/a	165.7	231.9	589.1	n/a
c203.100	n/a	n/a	173.6	243.0	588.7	n/a
c204.100	182.0	588.1	323.3	452.6	588.1	0.00
c205.100	n/a	n/a	4.8	6.7	586.4	n/a
c206.100	n/a	n/a	4.6	6.5	586.0	n/a
c207.100	n/a	n/a	8.2	11.4	585.8	n/a
c208.100	n/a	n/a	7.8	10.9	585.8	n/a

fixed $I_{\max} = 30$, $R_{\max} = 200$, and saved the best 5,000 columns generated at each call of the TS heuristic.

Tables 1 and 2 compare the pulse algorithm against Baldacci, Mingozzi, and Roberti (2011a). They defined four different bounding procedures based on different relaxations of the routes. The bound obtained with the ESPPRC as a subproblem is called LB_3 and is the one that we compare against. Column 1 shows the name of the instance; columns 2 and 3 show the total computing time in seconds and the bound LB_3 reported by Baldacci, Mingozzi, and Roberti (2011a); columns 4 and 5 present the total computing time for our approach and the scaled time in seconds (in bold if it is better than the benchmark); column 6 shows the lower bound obtained using our approach; and column 7 presents the bound improvement in percentage (in bold if our bound is tighter than the benchmark). Values not reported by Baldacci, Mingozzi, and Roberti (2011a) are shown in the tables as n/a.

For the Solomon 100-series our approach was faster than Baldacci, Mingozzi, and Roberti (2011a) in 25 of the 28 reported instances. The lower bounds are tighter in the r-instances and the rc-instances (20 instances) and are the same for the c-instances (eight instances). Besides, about 90% of the instances were solved in less than 10 seconds.

For the harder instances in the Solomon 200-series our approach was faster than Baldacci, Mingozzi, and Roberti (2011a) in 13 of the 20 reported instances. Once again, our bounds are tighter for the r-instances (in 10 of 11 instances) and the rc-instances (in eight of eight instances) and are the same for the c-instances (one reported instance). Given the fast performance of the pulse algorithm, our approach was able to find tighter bounds than LB_3 in less computation time. Roughly 70% of these harder problems were solved within one minute.

Desaulniers, Lessard, and Hadjar (2008) selected a subset of 12 instances that are among the most difficult ones. They reported the time to solve the linear relaxation at the root node and the bound obtained. Table 3 shows the comparison of our approach against Desaulniers, Lessard, and Hadjar (2008).

On the subset of hard Solomon instances, our approach was faster than Desaulniers, Lessard, and Hadjar (2008) in 10 of the 12 instances. The lower bounds obtained are exactly the same, because both approaches rely on an ESPPRC subproblem under a traditional simplex-based column generation scheme.

Table 3 **Comparing the Pulse Algorithm Inside a Column Generation Scheme Against Desaulniers, Lessard, and Hadjar (2008) on a Subset of Hard Solomon Benchmark Problems**

	Desaulniers, Lessard, and Hadjar (2008)		Column generation with pulse and TS		
Instance	Time (s)	Lower bound	Time (s)	Scaled time (s)	Lower bound
rc104.100	36	1,101.8	**6.6**	**18.6**	1,101.8
rc108.100	26	1,073.4	**4.9**	**13.7**	1,073.4
r108.100	26	913.5	**7.4**	**20.8**	913.5
r112.100	30	926.7	**7.5**	**21.0**	926.7
rc203.100	95	922.5	34.7	97.1	922.5
rc206.100	68	1,038.6	**19.2**	**53.8**	1,038.6
rc207.100	148	947.3	**42.8**	**119.9**	947.3
r203.100	172	866.9	72.4	202.8	866.9
r205.100	97	938.9	**25.3**	**70.8**	938.9
r206.100	210	866.9	**44.7**	**125.2**	866.9
r209.100	181	841.4	**42.3**	**118.4**	841.4
r210.100	269	889.4	**34.4**	**96.3**	889.4

5.2. Measuring the Performance of the Pulse Algorithm on the ESPPRC

We compared the average time it took the pulse algorithm to solve the ESPPRC subproblems inside the column generation scheme, against the average time used by GENR to solve the ESPPRC subproblems inside the H^3 bounding procedure by Baldacci, Mingozzi, and Roberti (2011a). Table 4 shows the results of this experiment. Column 1 presents the name of the instance; column 2 shows the GENR average time in seconds and column 3 shows the pulse average time in seconds (in bold if it is better than the benchmark); and finally, columns 4 and 5 present the best and worst running times for the pulse algorithm. It is important to note that we did not code the GENR algorithm, but Baldacci, Mingozzi, and Roberti (2011a) generously shared their results with us.

It is worth highlighting that both algorithms solved most of the instances within fractions of a second. In 20 of 29 instances, the pulse algorithm was faster than GENR; and they tied in seven. The best/worst running times for the pulse algorithm are very close to the average, indicating the robustness of the proposed algorithm.

5.3. Assessing the Performance of the Pruning Strategies

We performed an additional experiment to assess the contribution of the proposed pruning strategies inside the pulse algorithm embedded into the column generation scheme. Table 5 shows the results of this experiment over the set of difficult instances defined by Desaulniers, Lessard, and Hadjar (2008). Column 1 presents the name of the instance. Column 2 shows the average total time in seconds for solving a single ESPPRC subproblem. Columns 3 and 4 present the average time spent inside the bounding step and in the pulse function execution. Column 5 shows the average number of paths that reached the end node. Note that if the pruning strategies are strong, they are likely to prune paths efficiently and few should reach the end node. Finally, columns 6 to 8 present the average number of paths pruned by each strategy. The last line presents the geometric average for each measure. Note that the geometric average is a fairer comparison for the strategies since large/small values

Table 4 **Comparing the Pulse Algorithm Against Baldacci, Mingozzi, and Roberti (2011a) on ESPPRC Instances**

	GENR time (s)	Pulse time (s)		
	Avg.	Avg.	Min.	Max.
r101.100	< 0.1	0.02	0.01	0.05
r102.100	0.1	**0.02**	0.01	0.05
r103.100	0.1	**0.03**	0.01	0.06
r104.100	0.3	**0.06**	0.05	0.09
r105.100	< 0.1	0.03	0.01	0.05
r106.100	0.1	**0.03**	0.01	0.08
r107.100	0.1	**0.04**	0.03	0.09
r108.100	0.3	**0.07**	0.04	0.10
r109.100	0.1	**0.02**	0.01	0.06
r110.100	0.1	**0.04**	0.03	0.12
r111.100	0.1	**0.04**	0.03	0.08
r112.100	0.2	**0.07**	0.06	0.11
rc101.100	0.1	**0.02**	0.01	0.05
rc102.100	0.2	**0.05**	0.02	0.12
rc103.100	0.2	**0.05**	0.03	0.10
rc104.100	0.4	**0.35**	0.31	0.44
rc105.100	0.1	**0.02**	0.01	0.05
rc106.100	0.2	**0.03**	0.01	0.06
rc107.100	0.2	**0.07**	0.03	0.14
rc108.100	0.2	**0.19**	0.16	0.23
c101.100	< 0.1	0.03	0.01	0.06
c102.100	< 0.1	0.05	0.03	0.10
c103.100	0.2	0.29	0.19	0.37
c104.100	0.5	1.12	0.67	1.51
c105.100	< 0.1	0.04	0.01	0.05
c106.100	< 0.1	0.02	0.01	0.05
c107.100	< 0.1	0.02	0.01	0.06
c108.100	0.2	**0.04**	0.01	0.06
c109.100	0.3	**0.07**	0.04	0.12

Table 5 Assessing the Strength of the Pruning Strategies on a Subset of Hard Solomon Benchmark Problems

					Paths pruned by (in thousands)		
Instance	Total time (s)	Bound time (s)	Pulse time (s)	Complete paths (in thousands)	Infeasibility	Bounds	Rollback
rc104.100	0.35	0.09	0.26	249	10,270	4,830	17
rc108.100	0.19	0.08	0.11	108	5,165	1,504	5
r108.100	0.07	0.04	0.03	3	82	152	2
r112.100	0.07	0.06	0.01	3	99	113	1
rc203.100	0.7	0.68	0.06	18	288	898	214
rc206.100	0.14	0.13	0.01	1	8	39	13
rc207.100	0.86	0.84	0.03	5	109	237	184
r203.100	2.16	2.11	0.05	16	142	751	857
r205.100	0.13	0.12	0.01	1	9	50	8
r206.100	0.69	0.66	0.03	8	89	445	160
r209.100	0.31	0.29	0.01	2	22	78	19
r210.100	0.27	0.26	0.02	2	19	128	50
Geometric avg.	0.30	0.22	0.03	6	114	273	25

on few instances do not have a great impact on the measure as often happens with the arithmetic average (Bixby 2002).

The geometric averages from Table 5 show that most of the time is spent in the bounding stage of the algorithm. However, the bounds pruning strategy is also the strongest, followed by infeasibility and rollback pruning. Given the combinatorial hardness of the problem, it is worth noting that just a few thousand paths reach the end node, suggesting that the pruning strategies are effectively pruning a vast number of solutions. Consequently, the instances where large amounts of paths reach the end node are also the ones that require more time to solve.

6. Conclusions and Future Work

Accelerating the solution of the ESPPRC is a critical aspect in current branch-and-price methodologies for multiple variants of the VRP. We present a new exact algorithm for the ESPPRC that works well when embedded inside a column generation scheme that solves the linear relaxation (root node) of the VRPTW. Comparing our approach against the algorithms by Baldacci, Mingozzi, and Roberti (2011a) and Desaulniers, Lessard, and Hadjar (2008) over the Solomon's benchmark problems, we found equal or tighter bounds in less time for 38 of 48 instances and for 10 of 12 instances, respectively. Moreover, in a head-to-head comparison against the state-of-the-art algorithm GENR by Baldacci, Mingozzi, and Roberti (2011a), our algorithm solves the ESPPRC subproblems faster on 20 of 29 instances. Nevertheless, both GENR and the proposed pulse algorithm are able to solve most of the problems in just a fraction of a second. After assessing the strength of the proposed pruning strategies, we found that the bounds strategy is the main one responsible for the fast performance of our algorithm.

Beyond the solution of the ESPPRC, the proposed algorithm can be seen as a general framework that can efficiently handle different shortest path problems. From this point of view, the framework could be extended to include additional constraints to tackle multiple variants of the VRP. The general idea is that constraints are handled with additional pruning strategies to effectively prune the solution space.

Future work includes embedding our approach inside a branch-and-price algorithm for the VRPTW. Work currently underway concentrates on extending the pulse intuition to solve other complex VRP subproblems like those arising in the dial-a-ride problem, the orienteering problem with time windows, and rich versions of the shortest path—like the one with replenishment arcs.

Acknowledgments
The authors would like to thank Professor Roberto Roberti at Università di Bologna for his generosity in sharing the GENR average running times for the Solomon 100-series instances. The authors would also like to thank the anonymous referees for their insightful comments.

References

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

Baldacci R, Mingozzi A, Roberti R (2011b) New state-space relaxations for solving the traveling salesman problem with time windows. *INFORMS J. Comput.* 24(3):356–371.

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

Baldacci R, Bartoline E, Mingozzi A, Roberti R (2010) An exact solution framework for a broad class of vehicle routing problems. *Comput. Management Sci.* 7(3):229–268.

Bixby RE (2002) Solving real-world linear programs: A decade and more of progress. *Oper. Res.* 50(1):3–15.

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

Chabrier A (2006) Vehicle routing problem with elementary shortest path based column generation. *Comput. Oper. Res.* 33(10):2972–2990.

Desaulniers G, Lessard F, Hadjar A (2008) Tabu search, partial elementarity, and generalized k-path inequalities for the vehicle routing problem with time windows. *Transportation Sci.* 43(3):387–404.

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

Desrosiers J, Lübbecke ME (2005) A primer in column generation. Desaulniers G, Desrosiers J, Solomon MM, eds. *Column Generation* (Springer, New York), 1–32.

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

Dongarra JJ (2013) Performance of various computers using standard linear equations software. Technical Report CS-89-85, University of Tennessee, Knoxville.

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

Feillet D, Gendreau M, Rousseau L-M (2007) New refinements for the solution of vehicle routing problems with branch and price. *Inform. Systems Oper. Res.* 45(4):239–256.

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.

Gendreau M, Potvin J-Y (2010) Tabu search. Gendreau M, Potvin J-Y, eds. *Handbook of Metaheuristics*, International Series in Operations Research & Management Science, Vol. 146 (Springer, New York), 41–59.

Glover F, Laguna M (1997) *Tabu Search* (Kluwer, Norwell, MA).

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

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

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

Joksch H (1966) The shortest route problem with constraints. *J. Math. Anal. Appl.* 14(1):191–197.

Lozano L, Medaglia AL (2013) On an exact method for the constrained shortest path problem. *Comput. Oper. Res.* 40(1):378–384.

Petersen B, Pisinger D, Spoorendonk S (2008) Chvátal-Gomory rank-1 cuts used in a Dantzig-Wolfe decomposition of the vehicle routing problem with time windows. Golden B, Raghavan R, Wasil E, eds. *The Vehicle Routing Problem: Latest Advances and New Challenges* (Springer, New York), 397–419.

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

Rousseau L-M, Gendreau M, Pesant G, Focacci F (2004) Solving VRPTWs with constraint programming based column generation. *Ann. Oper. Res.* 130(1):199–216.

Toth P, Vigo D (2002) *The Vehicle Routing Problem*, SIAM Monographs on Discrete Mathematics and Applications (SIAM, Philadelphia).