Adaptive Large Neighborhood Search Algorithm for the Rural Postman Problem with Time Windows

Marcela Monroy-Licht
Département de Mathématiques et de Génie Industriel, Polytechnique Montréal, Canada
Centre Interuniversitaire de Recherche sur les Réseaux d'Entreprise, la Logistique et le Transport (CIRRELT), Montréal, Canada

Ciro Alberto Amaya
Departamento de Ingeniería Industrial, Universidad de Los Andes, Bogotá, Colombia
Grupo de investigación en producción y logística (PYLO), Bogotá, Colombia

André Langevin
Département de Mathématiques et de Génie Industriel, Polytechnique Montréal, Canada
Centre Interuniversitaire de Recherche sur les Réseaux d'Entreprise, la Logistique et le Transport (CIRRELT), Montréal, Canada

The *rural postman problem with time windows* is the problem of serving some required edges with one vehicle; the vehicle must visit these edges during established time windows. This article presents a competitive *adaptive large neighborhood search algorithm* to solve the problem. Computational experiments are performed on a large set of instances with up to 104 required edges. The results show that this approach is efficient, significantly reducing the computational time on large instances and achieving good solutions: the algorithm is able to solve to optimality 224 of 232 instances. **NETWORKS, Vol. 000(00), 000–000 2017**

Keywords: rural postman problem; time windows; adaptive large neighborhood search; metaheuristics

1. INTRODUCTION

Winter road maintenance includes all the operations that aim at the removal or reduction of snow and ice on roadways providing safe winter driving conditions and safe sidewalks for pedestrians. Spreading chemicals and abrasives, plowing roadways and sidewalks, loading snow into trucks, and ice control are some examples of operations to maintaining the safety and the mobility in cities and rural areas. The complex operations, the infrastructure constraints, especially in urban areas, and the dynamic nature of the operating conditions make the winter road maintenance a challenge and costly work for many governments and transportation agencies in North America. For example, in Ontario, the total expenditures in highway winter maintenance reached $171 million in the 2013 fiscal year [15]. The New Jersey Department of Transportation reported that it spent a record $138 million to keep state roadways clear of snow and ice for 2013 [29]. The Pennsylvania Department of Transportation, which had $189.2 million budgeted for the 2013–2014 winter, spent $284 million [29].

Winter maintenance authorities are constantly seeking for technology-based solutions such as advanced road weather information systems for monitoring localized weather and road surface conditions, automated vehicle location and Global Positioning System for tracking fleet operations and performance [7]. However, to obtain a great benefit from these technologies, the integration with algorithms that support the decisions and plans in operational, strategic and tactical level is necessary.

Perrier et al. [18–21] present a comprehensive review of models and algorithms developed for the variety of winter road maintenance operations. This work is divided into four surveys. The first one focuses on optimization models and solution algorithms for the design for spreading and plowing. The second one discusses system design problems for snow disposal operations. The last two address vehicle routing, depot location, and fleet sizing models for winter road maintenance.

Later, Perrier et al. [17] provide a survey of recent optimization models and solution methodologies for the routing of spreading operations. They present a detailed classification scheme for spreader routing models developed over the past 40 years. They emphasize that the new models

Received August 2014; accepted May 2017
Correspondence to: M. Monroy-Licht; e-mail: marcela.monroy@polymtl.ca
DOI 10.1002/net.21747
Published online 00 00 2017 in Wiley Online Library (wileyonlinelibrary.com).

demonstrate impressive capacities to include more issues of the real complexity, the use of more sophisticated hybrid solutions strategies and consideration of more comprehensive models that integrate vehicle routing with other strategic winter maintenance problems. However, they note that there is still a large gap between state-of-the-art models and actual implementations.

Campbell et al. [2] present the most recent survey on operational research methods on snow plow routing and a case study on implementation of route optimization for snow plowing. They report the works not covered in the survey of Perrier et al. [19]. This review documented the trend from a heuristic approach to mathematical programming-based approaches, as well as efforts to include more issues of the real complexity in snow plowing routing. They conclude that even when in the last decade there has been some impressive progress in snow plow routing research, similar progress has not occurred in implementing route optimization for winter road maintenance because it seems that models are generally still not comprehensive enough to consider all that needs to be included, and the mathematical programming-based models do not have the ease of use required by operating personnel.

Eglese et al. [5] provide detailed background on arc routing for the salt spreading. The survey shows that while early work has used simple constructive heuristics, more recently various metaheuristics algorithms have been developed for salt spreading applications. The authors note that there are few works of exact solution methods being used in real cases because a reasonable amount of computing time is required. Hence solution methods tend to rely on heuristic approaches.

Ice control is one of the key operations in winter road maintenance to ensure safety conditions for drivers and pedestrians. However, in arc routing problems it has not received as much attention as snow plowing or spreading chemicals for winter. In this article, we study the ice control conducted by the Ministry of Transportation in the region of "Estrie" in the Province of Quebec from mid-October to mid-December.

In particular, the problem is defined for a patrol, who daily must detect black ice on roads in a timely fashion in order to avoid pedestrian falls or automobile accidents. Black ice may appear at low temperatures close to 0 °C or if the air warms suddenly after a prolonged cold spell. Once the patrol has noticed a possible risk for safe mobility, it reports the incident to service center in charge of road signs and special works. In order to define the route for a day, the patrol matches the report of weather stations in different zones to the roads with a high likelihood of black ice appearing and decides which roads to visit and the suitable time to visit them.

The ice control problem defined is equivalent to solve the *rural postman problem with time windows* (RPPTW), where the time windows are determined by the meteorological conditions reported.

More formally, given a graph $G=(V,E)$, where V is a set of vertices and E is a set of edges, if $E_R \subseteq E$ is a subset of required edges, the problem consists of finding a minimum-cost tour that visits all the edges in E_R at least once, starting and finishing at the same origin vertex. In addition, every visit should be carried during a time interval defined by the time windows interval $[a_i, b_i]$ for $i \subseteq E_R$. The patrol should wait along the tour if it arrives to any request before the earliest possible time, a_i, and start the service only when the time windows opens.

The RPPTW is NP-hard, even when $E_R = E$, since it reduces to the *Chinese postman problem* (CPP) *with time windows*, which is NP-hard [4]. If $E_R \subset E$ and all the time windows are defined by the interval $[0,\infty]$, it reduces to the *rural postman problem* (RPP). The RPP has been shown to be NP-hard [11].

Other applications of the RPPTW may be found in areas related to arc routing problems such as trash collection where the collection must respect some schedules in large cities or such as street sweeping where the activity must be done during some specific time because of parking restrictions.

This work is an extension of our previous work [13] where an exact method is presented as solution method. The aim now is to develop an alternative solution method giving good, but not necessarily optimal, solutions. The focus is to propose an algorithm that solves the hardest instances quickly, so we have developed a metaheuristic approach. This is a contribution to the developing of optimization methods for time-sensitive arc routing problems useful in winter road maintenance.

The remainder of this article is organized as follows. Section 2 summarizes related work. Section 3 gives a formal description of the problem. The Section 4 describes the *adaptive large neighborhood search* implemented. Experimental results are presented in Section 5, and Section 6 provides concluding remarks.

2. LITERATURE REVIEW

The ice control problem is a time-sensitive arc routing problem. This is an important feature on winter road maintenance noted in several surveys; for instance, Eglese et al. [5] and Perrier et al. [17] state that the timing of operations is crucial to achieve the desired level of service in winter road maintenance. The literature regarding to time-sensitive arc routing problems in this context is still scarce. Table 1 summarises the works on time-sensitive arc routing problems in winter road maintenance.

From Table 1, it can be seen that most works that consider timing-sensitive arc routing problems have focused on the capacitated case and column generation is the approach most used as the solution method.

Eglese [6] first presents an application in routing for winter gritting. Multiple depot locations and limited vehicle capacities are considered, and roads with different priority imply that some roads must be treated within 2 h and other within 4 h of the start of gritting. This problem can be considered as *a capacitated arc routing problem with time windows* (CARPTW), where the time windows are rather wide. A two-phase heuristic method is proposed as solution method: first it solves an unconstrained CPP when road of category 1 are

TABLE 1. Synthesis of works in winter road maintenance considering time-sensitive.

Problem	Authors	Real application	Problem characteristics	Solution method
CARPTW*	Eglese [5]	Spreading operations	Multi-depots Wide time windows	Two-phase heuristic
RPP** with deadline classes	Letchford and Eglese [6]	Spreading operations	Deadline classes	Cutting-plane approach
CARPTW*	Golbaharan [8]	Plowing routing	Multi-depots Time windows	Column generation
CARPTW*	Razmara [23]	Plowing routing	Time windows	Column generation
CARP*** with time dependent service cost	Tagmouti et al. [32]	Spreading operations	Time-dependent service cost	Column generation
CARP*** with time dependent service cost	Tagmouti et al. [33]	Spreading operations	Time-dependent service cost	Variable neighborhood descent
Dynamic CARP*** with time dependent service cost	Tagmouti et al. [34]	Spreading operations	Dynamic version Time-dependent service cost	Variable neighborhood descent
RPP with time windows	Monroy-Licht et al. [14]	Ice control	Time windows	Cutting plane algorithm

CARPTW* Capacitated arc routing problem with time windows.
RPP** Rural postman problem.
CARP*** Capacitated arc routing problem.

considered. Then depot locations and routes are defined. The second phase attempts to improve the current solution using a simulated annealing algorithm.

Golbaharan [8] studies a *multi-depot* CARPTW in the context of snow removal. Every snow plow starts from a depot and returns to the same depot. The problem is formulated as a linear integer programming model; indeed as a constrained *set covering problem*. The objective function in this case minimizes the total cost of the routes and the penalty for using extra snow plows. A column generation method is implemented to solve the problem. The master problem includes the constraints on the number of snow plows available at each depot and on the required services. The subproblem contains the time window constraints and network flow constraints. The master problem is solved by the dual simplex method, and the subproblem for every depot is formulated as a *shortest path problem* (SPP) *with time windows*, and it is solved with a label-setting algorithm. Computational experiments are conducted on real-life instances involving 7 depots, 21 snow plows, 362 nodes, and 707 required edges.

Razmara [23] presents a real problem of the Swedish National Road Agency on snow removal routing for homogeneous snowplows; in this case every segment in the network must be plowed in its associated time windows and the routes must start from and end at the same depot. The case is formulated as a linear integer programming problem and solved using a Dantzig-Wolfe decomposition scheme similar to the work of Golbaharan [8], but in this work an integer solution to the master problem is found by a greedy algorithm or a variable reduction procedure.

Tagmouti et al. [32] study the directed *capacitated arc routing problem* (CARP) *with time-dependent service cost* inspired on winter gritting operations, where the timing of an intervention is crucial; if the intervention is too early or too late, the cost in material and time increases; therefore, the cost depends on the time of beginning of service, indeed, the cost is a piecewise linear function of time. The problem consists of finding a set of routes that serve all required arcs in the graph at least cost (sum of travel cost and service cost) with the constraint that vehicles are not allowed to wait along their route and must be back at the depot by a given deadline. A column generation algorithm is proposed to solve the problem where the master problem is a *set covering problem* and the subproblems are *time-dependent SPP with resource constraints*. The method is tested on a set of instances derived from benchmarks *vehicle routing problem* (VRP) *with time windows* solving to optimality problems with up to 40 required arcs.

Later, Tagmouti et al. [33] propose a *variable neighborhood descent* (VND) heuristic for solving the same problem. First, two initial solutions are constructed with insertion and saving heuristics. The VND is then applied to each solution to improve them. The structure of the neighborhoods manipulates an arc or sequence of arcs. The performance of the algorithm is tested on a set of instances adapted from the CARP. The algorithm behaved appropriately and shows to be fast and competitive when compared with the adaptive multi-star local search algorithm of Ibaraki et al. [9] which is designed for the VRP with soft time windows.

Tagmouti et al. [34] study the dynamic version of their previous work: the CARP *with time-dependent service cost*. In the dynamic case, the time interval where the service cost is minimal changes due to weather report updates, and therefore real-time modifications are required to the current routes. An adaptation of the VND of their previous work is presented as solution method. A starting solution is first computed with VND using service time cost functions based on an initial forecast. A simulated storm goes through the network and move along the x-axis. At different times, weather reports are received and update the storm speed. The VND is applied on a new static problem each time a weather report is received. In each static problem, the graph is updated taking into account the already visited arcs. The algorithm is tested on a set of 60 generated instances with weather reports received every 5 min. The VND shows to be fast and allows the system to quickly use the new solution.

There are only two works in winter road maintenance applications using the *time-sensitive rural postman problem* [12, 13].

The RPP *with deadline classes* is presented by Letchford and Eglese [12]. In this problem, the set of arcs are partitioned into small number of classes according to priority, with each class having its own deadline on service. Formally the set R of required edges is partitioned into $\{R^1, \ldots, R^p\}$ and services for each class k of edges $(k = 1, \ldots, p)$ must be completed by time T^k. The problem is formulated on an undirected graph as an integer linear programming model. The authors have proposed several classes of valid inequalities which exploit the structure of the problem. They use a dual cutting-plane method to solve the problem: an initial *lineal program* (LP) relaxation is solved and then each time that a violated inequality is identified, it is added to the LP, and the LP is solved using the dual simplex method. When no more violated inequalities can be found, branch-and-bound is invoked to obtain integrality. The algorithm is tested on a set of 10 instances: 5 problems from Corberán and Sanchis [3] were adapted and the value of p was set to 1 and 2 for each of the problems. The problems have between 22 and 67 required edges and between 3 and 6 connected components. The cutting plane algorithm shows a good performance as all the instances are solved to optimality.

Our preliminary work on the ice control operation [13] introduces the problem. Three formulations based on mixed integer linear programming are presented for the undirected and directed case, and a solution approach is proposed: a cutting plane algorithm including valid inequalities from the *traveling salesman problem* (TSP) *with time windows* and from the *precedence constrained* TSP. Several tests are done on two sets of instances. For the first set, 222 of out 225 instances with time windows structured by each required edge are solved to optimality. The instances in the second set are larger and with time windows structured by time slots: 5 of out 9 instances with up to 104 required edges are solved to optimality.

Works related to solving time-sensitive RPP in other applications, different to winter road maintenance, are also few. An application involving scheduling with time-dependent processing [1, 31] introduced the *time-dependent* RPP. In this case, the travel (or service) time of each arc depends on time, that is, the travel time of an arc depend on the time interval during which the arc is traversed, and the postman is not required to cover every arc in the network, but only a subset of arcs. A formal definition is presented by Tan et al. [36]. Both the travel time and the service time are time-dependent piecewise functions. A postman located at the depot vertex is required to service the arcs and to end the service tour at the initial location. The postman is allowed to wait along the tour and must start after a given time t_0. The *time-dependent* RPP consists of finding the postman tour servicing all of the required arcs with a minimum cost with respect to the time-dependent travel time and the service time.

Tan and Sun [35] and Tan et al. [36] present the version that considers only time-dependent travel times and propose an arc-path formulation and strong valid inequalities. The authors note that the constant travel time assumption in other timing sensitive arc routing problems never holds on the time dependent network. Thus, the transformation methods which use the shortest path algorithm have as subproblem the *time dependent* SPP, which has been proved to be NP-hard [16]. They propose an integer linear programming: an arc-path formulation for the problem with a constraint set divided into two parts. The first part defines the polytope of the arc-path alternation sequence and the second part is closely related to time-dependent travel time. The service time functions considered as piecewise functions are linearized. Based on the polyhedral results, a cutting plane algorithm is proposed as solution method. The algorithm is tested on two sets of randomly generated instances with up to 50 vertices and with up to 50 arcs; the travel time is treated as the step function with 3 and 4 intervals, and the percentage of required arcs ranges from 10% to 30%. The computational results show that for all 42 test instances the method solved instances up to 25 vertices and 50 arcs. The relative gap between the best feasible solution and the lower bound is 3.16%for all the instances on average.

There are three works that deal with the RPPTW. The first approach to the RPPTW is that of Nobert and Picard [14]. The authors define two types of required edges, E_1 and E_2. The edges in E_1 must be serviced during the morning and the edges in E_2 may be serviced at any time. A heuristic algorithm based on the solution of two rural path problems and on the computation of suitable penalties is presented, but no numerical results are given. Kang and Han [10] solve a relaxed version of the problem: late arrivals are penalized. The authors present a genetic algorithm to solve a bi-objective problem that minimizes the total travel cost and the total penalty, and they compare three crossover operators.

Recently, Sun et al. [30] present a work that solves the RPPTW for the directed case. They propose a unified *Timed Automata* (TA) model that solves also the *Chinese postman with time windows*. They show TA as a natural tool for posing and solving time varying postman problems. Computational results show that the TA model can solve small-sized instances optimally and it can obtain a better optimality gap than those obtained via a cutting plan algorithm.

In summary, the only work dealing with the Rural Postman Problem with Time Windows for the undirected case and shows computational results is our previous work [13]. The results showed that the computational time becomes excessive when the exact algorithm is used on the hardest instances and impractical to solve real larger cases. Therefore, we decide to explore a metaheuristic as a solution method. We have opted for an ALNS because it has performed well on time-window routing problems [25] and has achieved good results on complex arc routing problems [24, 27, 28]. The proposed ALNS addresses the undirected version of the RPPTW, but it could easily be extended to other versions of the problem.

3. PROBLEM DEFINITION

Let $G = (V,E)$ be an undirected graph, where V is the vertex set and E is the edge set, which is partitioned into two

subsets: $E_R \subseteq E$, the set of required edges and $E \backslash E_R$, the set of non-required edges. Each edge i in E is associated with a travel time T_i and a travel cost c_i. Additionally we denote by $[a_i\ b_i]$ the associated time window for each edge i in E_R, where a_i and b_i respectively represent the earliest and latest time where the postman is allowed to start the service of edge i. The RPPTW is the problem of finding a minimum-cost tour starting and finishing at the depot node, $v_0 \in V$, servicing each edge i in E_R exactly once, such that the starting time of servicing each edge i in E_R is in its associated time window.

Three formulations of the problem and an exact solution method are presented in [13]. The first one is a "direct" formulation, where for each required edge in E_R, $|E_R|$ copies must be included in the graph. However even if a set of constraints are included reducing the number of equivalent solutions, this formulation is still not practical. The other two formulations transform the original graph. The transformations use shortest path algorithms to produce the travel time on the edges in the transformed graph.

The limitations of the exact method proposed to solve this problem are naturally the size of the instances and the structure of the time windows. Two time windows structures were defined: (i) a different time window for each required edge and (ii) time slots, where subsets of required edges have the same time window. The results show that even if the instances are larger in terms of required edges, it seems more difficult to solve problems with a smaller number of required edges but with different time windows for each required edge. In order to provide a competitive solution method for real instances with hard time windows, where the cutting plane algorithm is not able to solve the problem, we have explored a heuristic solution method: an ALNS.

4. ADAPTIVE LARGE NEIGHBORHOOD SEARCH

This section describes the ALNS heuristic proposed to solve the RPPTW. ALNS was introduced by Pisinger and Ropke [22] and Ropke and Pisinger [26]. Basically, the algorithm constructs an initial solution and then attempts to improve it using competing heuristics. At each iteration, a heuristic is chosen to remove part of the current solution, and another one is selected to insert the removed part in a likely better position. The new solution is accepted if it satisfies a criterion defined by a search implemented at the master level. ALNS is based on two principles. First, it explores a large neighborhood of the solutions (as the algorithm has many possibilities for moving to other solutions, the neighborhood becomes very large). Second, the search is adaptive, this is it uses the best removal and insertion heuristics of the past iterations (at every iteration of the algorithm, the heuristics are selected based on their past performance).

4.1. Initial Solution

A required edge is represented by the pair of nodes (i,j) corresponding to its end nodes. This notation indicates the direction in which the edge is traversed. We represent a solution x as the ordered sequence $(i,i)_0$, $(u,v)_1$,...,$(w,t)_n$, $(i,i_{n+1}$ of the n required edges together with the edges placed at positions 0 and n+1, which represent the depot edge. The cost of the solution x is the sum of the traversal cost across the sequence, that is, the sum of the shortest paths between the final node of one required edge and the initial node of the next.

We run three simple heuristics based on sorting schemes to get an initial solution:

- Increasing release date (IRD): This heuristic sort the required edges according to increasing release dates (lower bounds of time windows) and checks whether the sequence is feasible. Algorithm 1 gives the constructive procedure.
- Increasing due dates (IDD): This heuristic sort the required edges according to increasing due dates (upper bounds of time windows) and checks whether this sequence is feasible. The constructive procedure is similar to Algorithm 1.
- Increasing center dates (ICD): This heuristic sort the required edges according to increasing centers of the time windows, $a_i + \frac{b_i - a_i}{2}$, and checks whether this sequence is feasible. The constructive procedure is similar to Algorithm 1.

The best solution delivered by the three construction heuristics is chosen. If no feasible solution is found by any of the heuristics, we allow infeasible solutions by modifying line 3 of Algorithm 1 to *While* $|E_R| \neq 0$ *do*. We keep the resulting solution as the initial solution for the improvement phase.

Algorithm 1 IRD

Input: Set of n required edges E_R
Output: x, an ordered sequence of required edges

1 $k = 0$
2 $x_0 \leftarrow (i,i)_k$
3 **While** $|E_R| \neq 0$ & x_0 is feasible **do**
4 $k = k+1$
5 Choose $(u,v) \in E_R$ with the minimum release date
6 **If** $cost((w,t)_{k-1,}(u,v)) \leq cost((w,t)_{k-1,}(v,u))$ **then**
7 $x_0 \leftarrow x_0 \cup (u,v)_k, E_R = E_R \setminus (u,v)$
8 **Else**
9 $x_0 \leftarrow x_0 \cup (v,u)_k, E_R = E_R \setminus (v,u)$
10 **End if**
11 **End while**
12 **If** $|x_0| = n+1$ **then**
13 $x \leftarrow x_0$
14 **Else**
15 $x \leftarrow \emptyset$
16 **End if**
17 **Return** x

4.2. Improvement Phase

Given an initial solution x, we apply the ALNS algorithm until a stopping criterion is reached. The algorithm outputs the best solution x^* encountered during the search.

At each iteration, the algorithm chooses a removal and an insertion heuristic based on their weights. The selected

removal heuristic deletes some required edges from the solution x and then, the insertion heuristic attempts to reinsert them in better positions. Seven removal and two insertion heuristics were implemented. Now, we review the main elements of the heuristic.

4.2.1. Weight and Score Adjustments. The weights of the heuristics are based on their historic performance. At each iteration, we assign scores that depend on the performance (see Ropke and Pisinger [25]); a high score indicates a successful heuristic. If the removal and insertion heuristics are able to find a new overall best solution their scores are increased by σ_1. If the heuristics improve the current solution but not the overall best solution their scores are increased by σ_2. If the solution is worse than the previous solution but accepted with a given probability, the heuristic scores are increased by σ_3. The scores for both the removal and insertion heuristics are updated by the same amount, σ_1, σ_2, or σ_3.

The improvement phase is divided into *seg* segments and *sseg* subsegments. At the beginning of each segment the weights are set to one. After each subsegment of *ite* iterations, the weights are updated according to the scores obtained. Let $\bar{w}_{h,l}$ be the weight of heuristic h used in subsegment l. The weight for the heuristic h to be used in subsegment l+1 is calculated as follows:

$$\bar{w}_{h,l+1} = r\frac{\pi_h}{\theta_h} + (1-r)\frac{\sum_{d=1}^{l-1}\bar{w}_{h,d}}{l-1} \tag{1}$$

π_h is the score of heuristic h obtained during the previous subsegment l; θh is the number of times that heuristic h was used in subsegment l; r is a factor, $0 \leq r \leq 1$, that controls the emphasis placed on the score obtained in the last subsegment with reference to the historical average weight.

Each removal/insertion heuristic h is chosen at each iteration by two separate roulette-wheel mechanisms. The probability of selecting heuristic h for subsegment l is

$$\frac{\bar{w}_{h,l}}{\sum_{h=1}^{o}\bar{w}_{h,l}} \tag{2}$$

where o is the total number of removal/insertion heuristics (i.e., $o = 7$ or 2).

4.2.2. Acceptance and Stopping Criteria. The acceptance criterion for a new solution and the stopping criterion for the entire search usually are defined by implementing a master level search. We have chosen a simulating annealing (SA) framework. Let $C(x)$ represent the cost of the current solution x. The new solution given by the ALNS at each iteration is represented by $\bar{x}$. The SA determines whether or not the new solution $\bar{x}$ should be accepted. A solution is accepted if $C(\bar{x})$ is less than $C(x)$, and it is accepted with probability $e^{\frac{-C(\bar{x})-C(x)}{T_l}}$ if $C(\bar{x})$ is greater than or equal to $C(x)$, where $T_l > 0$ is the temperature. The temperature has an initial value T_0 calculated from the initial solution so that a solution that is ω% worse than the current solution is accepted with 50% probability [26]. At each subsegment the temperature is decreased slightly: $T_{l+1} = T_l - \left(\frac{T_0}{100}\right)$, where T_{l+1} is the temperature of subsegment $l+1$. Regarding the stopping criteria, in our implementation we stopped the algorithm after 25,000 iterations as it was done in [25].

4.2.3. Diversification. Toward the end of the search we accept only good moves, and therefore it is harder for the heuristics to get high scores. If no solutions improve the current best solution in an entire subsegment, the algorithm perturbs the best-known solution by reversing the traversal direction of $f|E_R|$ required edges. The $f|E_R|$ required edges are randomly chosen.

4.2.4. Penalized Objective Function. Our ALNS also allows infeasible solutions (with regards to the time windows), which improves the overall search; the infeasibility is penalized with a high cost in the objective function.

4.2.5. Removal Heuristics. Our ALNS framework for the RPPTW uses seven different removal heuristics. These heuristics take as input a given solution x and output a random number $q = \lceil [p_1|E_R|, p_2|E_R|] \rceil$ of required edges, called *requests*, which are removed from the tour.

R1. Random removal: This simple heuristic selects q requests at random and removes them from the solution x. The purpose of this heuristic is to diversify the search.

R2. Series removal: This heuristic selects a series of q consecutive requests. It randomly selects a position k between $k = 1$ and $k = n - q + 1$, where n represents the number of required edges, and removes the next q requests starting at position k and finishing at position $k + q - 1$.

R3. Worst-removal: This heuristic chooses q requests that are very expensive, i.e., with a long distance to cover them. It seems reasonable to try to remove requests with a high cost and insert them elsewhere to obtain a better solution.

Given a request (i,j) placed at position k in solution s, we define the cost of the request as $cost\,((i,j)) = d\left((u,v)_{k-1},(i,j)_k\right) + d\left((i,j)_k,(w,t)_{k+1}\right)$, where d represents the distance function between two requests.

The worst-removal heuristic repeatedly chooses the request (i,j) with the largest $cost\,((i,j))$ until h $(h = 2q)$ requests have been selected. To obtain variability in the removal, the heuristic is randomized: q random requests are removed from the h selected requests.

R4. Space-related removal: This heuristic removes a set of q related requests that could be closer and hence provide a better solution. For the RPPTW, we define the relatedness $r_{\{i,j\}\{u,v\}}$ of two requests by the distance (the shortest path) between them.

The heuristic initially selects a request (i,j) at random and places it into the set Q of selected requests. Then it repeatedly calculates $r_{(i,j)(u,v)}$ for all (i,j) in Q and (u,v) not in Q, and chooses the request (u,v) that minimizes $r_{(i,j)(u,v)}$. The algorithm stops when q requests have been chosen. Ties are broken randomly to diversify the output.

R5. Time-related removal: This heuristic is a variant of the space-related removal heuristic. Here we try to remove requests that are close in space and have similar time windows. The motivation is to try to remove requests that are easy to interchange.

We measure the time-window relatedness $m_{(i,j)(u,v)}$ as the difference of the centers of the time windows. The center of request (i,j) is defined by $m_{(i,j)} = a_i + \frac{b_{(i,j)} - a_{(i,j)}}{2}$. The heuristic first selects h requests ($h = 2q$) as in the space-related removal heuristic. Then, it chooses the q requests in Q with the minimum $m_{(i,j)(u,v)} = a_{i,j} + \frac{b_{(i,j)} - a_{(i,j)}}{2} - a_{u,v} + \frac{b_{(u,v)} - a_{(u,v)}}{2}$, i.e., it iteratively chooses the pair (i,j),(u,v) that minimizes $m_{(i,j)(u,v)}$ and places this pair into the set of selected requests.

R6. Far-time-close-position removal: This heuristic removes requests that seem to be placed wrongly because they are close in the solution but they have a high time windows center difference.

The heuristic starts by selecting a request (i,j) at random. Then the algorithm identifies the h requests with the largest differences in their time-window centers with respect to the time-window center of request (i,j). These requests are placed into H. Then, the heuristic iteratively chooses the $q-1$ requests in H closest to (i,j) in the solution x. The set of requests to remove is completed by adding the request (i,j).

R7. Series-far-time removal: This heuristic is an extension of the series removal heuristic. The purpose of this heuristic is to choose requests that could be placed wrongly, that is, they are close in the solution but they have a large mean time-window difference.

The heuristic first selects a series of $h = 2q$ requests, as in the series removal heuristic, and places them into H. Then, it chooses the q requests with highest $m_{(i,j)(u,v)}$ for (i,j),(u,v) in H.

4.2.6. Insertion Heuristics. The insertion heuristics construct a solution $\bar{x}$ by inserting requests into the partial solution x' obtained when the q requests from the removal heuristics are removed from x. The removed requests are initially placed in a set U.

I1. Basic greedy insertion: This heuristic is a simple construction heuristic that repeatedly inserts a request (i,j) into the cheapest position considering both directions in which the request can be inserted.

We define $c_{(i,j),k} = d((u,v)_{k-1},(i,j)) + d((i,j),(w,t)_k) - d((u,v)_{k-1},(w,t)_k)$ as the insertion cost of request (i,j) at position k of solution x'. If (i,j) cannot be inserted at position k, then $c_{(i,j),k} = \infty$. Let $\Delta x'_{(i,j)}$ denote the change in the objective value of solution x' incurred by inserting request (i,j) into its best overall position, i.e., $\Delta x'_{(i,j)} = min_k c_{(i,j),k}$.

At each iteration, the algorithm chooses the request $(i,j) \in U$ with the minimum $\Delta x'_{(i,j)}$ and inserts (i,j) into its cheapest position. Then (i,j) is removed from U. This process continues until all the requests in U have been inserted into solution x', and we then set $\bar{x} = x'$.

This heuristic has an obvious drawback: it often postpones the placement of "hard" requests (requests with large $\Delta x'_{(i,j)}$) until the last iterations, when there are few feasible positions because of the time windows. The next heuristic tries to circumvent this problem.

I2. Regret insertion: The regret heuristic tries to improve on the basic greedy insertion heuristic by incorporating a kind of look-ahead information when selecting the request to insert. We define the regret value $r_{(i,j)}$ as the difference of the two lowest values $c_{(i,j),k}$ for each request $(i,j) \in U$. In other words, the regret value is the difference between the cost of inserting the request into its best position and that of its second-best position.

At each iteration, the regret heuristic chooses to insert the request $(i,j) \in U$ with the maximum $r_{(i,j)}$. The selected request is inserted into its cheapest position considering both directions in which the request can be inserted. The process stops when all requests in U have been inserted into solution x', and we then set $\bar{x} = x'$.

4.3. Pseudocode of the Algorithm

The general structure of the algorithm just described is now summarized in the following pseudo-code (Algorithm 2).

Algorithm 2 ALNS

Input: Initial solution: x
Parameters: *seg, sseg, ite*
Output: The best solution encountered: x^*

Initialization
  Set $x^* = x$
  Set scores of R1, R2, R3, R4, R5, R6, R7, I2 and I2 = 0
For $i = 1$:*seg* **do**
  Set weights of R1, R2, R3, R4, R5, R6, R7, I2 and I2 = 1
  Set $T_1 = T_0$
  For l = 1:*seg* **do**
    For k = 1: *ite* **do**
      Choose a removal heuristic R and an insertion heuristic I using roulette- wheel selection principle based on weights of the segment l.
      Generate a new solution $\bar{x}$ from x using R and I
      If $C(\bar{x}) \leq C(x)$ or $\bar{x}$ is accepted with SA criterion **then**
        Set $x = \bar{x}$
        If $C(x) < C(x^*)$ **then**
          Set $x^* = x$
        End If
        Update scores of R and I and number of times R and I were used
      End If
    End For
    Update the weights of R1, R2, R3, R4, R5, R6, R7, I2 and I2
    Update the temperature T_{l+1}
    If no solutions improve the current best solution **then**
      Select $f|E_R|$ required edges randomly and reverse their traversal direction
    End If
  End For
End For
Return x^*

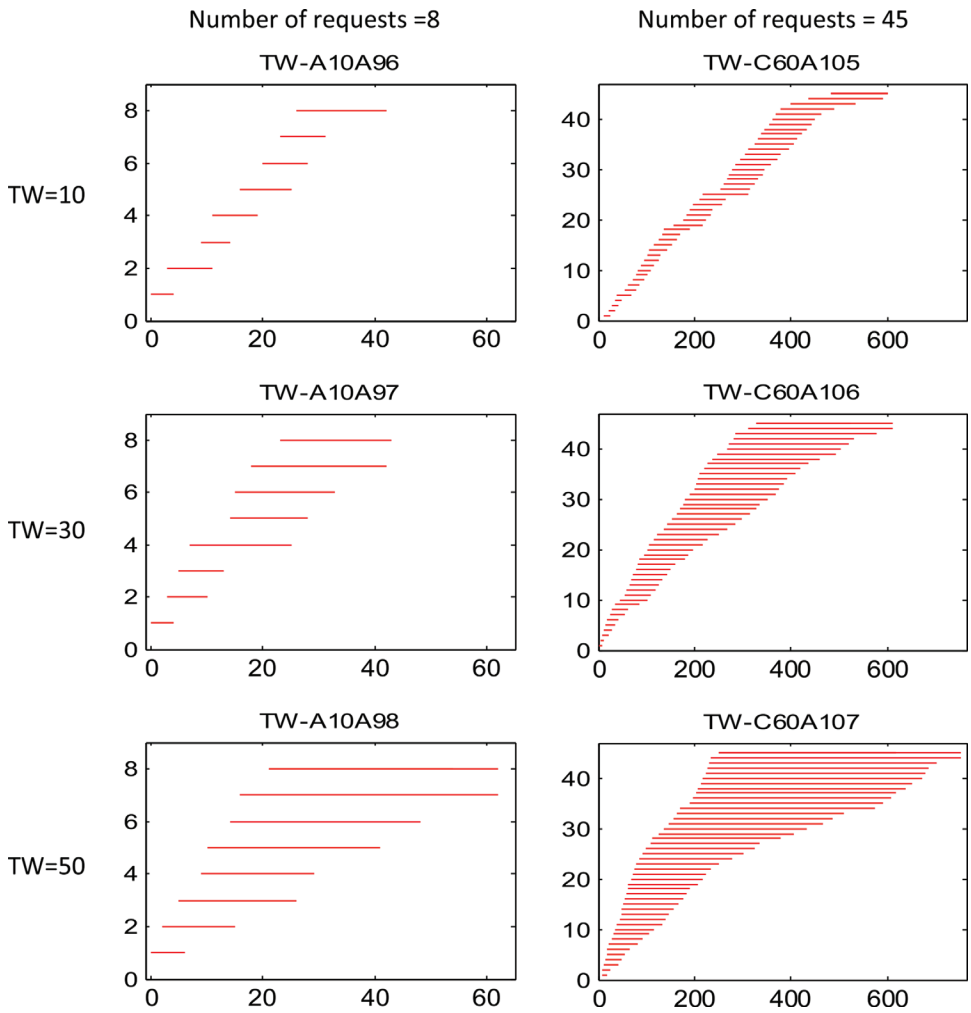

FIG. 1. Time windows structure. [Color figure can be viewed in the online issue, which is available at wileyonlinelibrary.com]

5. RESULTS

In this section, we describe our computational experiments. There are six major objectives for this section:

1. We present the set of instances and the optimal value, not previously reported, for some instances.
2. We discuss the parameter tuning based on statistical tests.
3. The performance and robustness of the overall algorithm depends on the choice of the removal and insertion procedures [25]. Therefore, the third objective is to compare the performance of the heuristics with an exact method.
4. We present a present a comparison with a heuristic solution.
5. We characterise the failure cases.
6. We also explore the performance of the algorithm on larger instances.

The tests were coded in Python and run on a 1.9-GHz AMD PC with 1983 MB of internal memory. We chose this test environment since it is the same as that used in [13].

5.1. *Instances*

We tested the algorithm on two sets of instances, both proposed by Monroy-Licht et al. [13]. These are the only known benchmark instances for the RPPTW.

The first group (*set1*) is a set of 225 instances with between 2 and 45 required edges, 2 and 12 connected components and 3 types of time windows: tight, labeled $TW = 10$; intermediate, $TW = 30$; and wide, $TW = 50$. The second set of instances (*set2*) is generated for a real network of major roads in the Estrie region. This set of 8 instances has between 74 and 104 required edges, 3 and 4 connected components and 3 widths of time windows (10, 30, and 50). Monroy-Licht et al. [13] presented 9 instances in *set2*, but we did not use instance *Inst-00* because it seems to have an error in the data.

In *set1*, each required edge has a randomly generated time window. The instances in *set2* have a special feature: the time windows are generated as four or five time slots, and so several required edges have the same time windows.

The structure of the time windows of both sets of instances is defined by partial intersections or empty intersections. In other words, there are no two interval time windows such that one is totally covered by the other. The Figure 1 presents

TABLE 2. Optimal solutions for benchmark instances.

Instance	*R*	*TW*	*O.F.*	Time
TW-B60A115	45	30	511.0	25779.5
TW-C40C152	35	50	289.0	23729.6
Inst-02	74	50	259.7	366.2
Inst-04	104	30	289.2	615.4
Inst-05	104	50	289.2	306.9

graphically the structure of the time windows. We selected 6 instances, 3 with 8 requests and 3 with 45 requests from *set1*. For each instance, we plot the time windows of the requests with a line. It can be seen that the intersections of the time windows are either partial or empty.

Monroy-Licht et al. [13] solved 227 instances to optimality. We ran the cutting plane algorithm proposed by the authors for the instances that could not be solved in less than 3 hours in [13]. With a time limit of 10 hours and a parameter tolerance gap of 0 we solved to optimality 5 new instances. Table 2 presents the optimal solution values (*O.F.*) and the time in seconds taken by the cutting plane algorithm. Column *R* indicates the number of required edges, and column *TW* indicates the type of time windows.

The cutting plane algorithm spends significantly more time on TW-B60A115 and TW-C40C152 than on the other instances. Since Inst-02, Inst-04, and Inst-05, which belong to *set2*, are larger than TW-B60A115 and TW-C40C152, we can see that it is easier to solve the larger instances because the structure of their time windows is different, as explained above.

5.2. *Tuning Set Parameters*

This section determines the parameters that need to be tuned. The weights of the ALNS are controlled by parameters σ_1, σ_2, σ_3 (performance scores of removal and insertion heuristics) and r (emphasis on the score obtained in recent iterations). To control the acceptance criteria we use the parameter ω. Finally, we must determine p_1 and p_2, the parameters that control how many requests we remove and insert, and f, which controls the number of required edges that can be reversed (change of the traversal direction).

Intuitively, p_2 and f should have a more significant effect on the quality of our algorithm. We focused on the calibration of these two parameters. For the other parameters, we decided to use the values used in similar work in the literature. For all the experiments, we used the parameter values determined in [22] and [26] for parameters r and ω. They were respectively set to 0.7 and 5. The parameters σ_1, σ_2, and σ_3 were set to 15, 8, and 2, respectively. The relative values of these parameters maintain the relationship determined in [22, 26]. p_1 was set to 0.1 as in [22].

The maximum number of requests that can be removed in a single iteration, $p_2|E_R|$, and the number of required edges that can be reversed (change of the traversal direction), $f|E_R|$, are controlled by p_2 and f, which are expressed as a percentage of the number of required edges.

We produced a fair parameter setting through empirical tests. Two values were chosen for p_2: 0.2 and 0.4. Parameter f was set to 0.02, 0.05, and 0.1. Then we applied a statistical F-test with two factors (p_2 and f); p_2 had 2 levels and f had 3 levels. The statistical test of factorial crossing gives us information about the effect of the six statistical treatments. Each statistical treatment was run 5 times.

For the tuning tests, we selected a random sample of 30 instances with different sizes and time windows widths from *set1* and *set2*; all of them with known optimal solutions. At the 95% level the test rejected the hypothesis that there is a difference between the performance of the parameter combinations, with p-values of 0.7093 and 0.6005 for the effect of f and p_2, and 0.677 for the effect of the interaction between f and p_2. Therefore, the ALNS results are sufficiently robust with respect to variations of the parameters, and no significant improvement in the solution can be expected by selecting a different parameter combination p_2, f. For the subsequent tests, the parameters p_2 and f were set to 0.2 and 0.1 respectively, as the preliminary tests indicated that these values provided the best performance.

One of the key factors in ALNS is to divide the entire search by segments with the aim of escaping local optimums. Here, after a process of trial and error, the number of segments *seg* and subsegments *sseg* were set to 10 and 10 respectively.

5.3. *Performance of Removal and Insertion Heuristics*

The objective of this section is to compare the individual performance of each removal and insertion heuristic to that of all the heuristics. We then explore the best combination of heuristics.

As before, we denote the removal heuristics by R1, R2, R3, R4, R5, R6, and R7 and the insertion heuristics by I1 and I2. We implemented different versions of the ALNS. Each was applied five times to each instance (and all subsequent experiments were also performed five times). Initially we applied VR1, VR2, VR3, VR4, VR5, VR6, VR7, VI1, and VI2. Versions VR1 to VR7 involve using one removal heuristic and both insertion heuristics I1 and I2. Versions VI1 and VI2 involve using all seven removal heuristics and one insertion heuristic. We refer to these as *single versions*. We also applied the full version (FV) of the ALNS, that is, with all removal and insertion heuristics.

Table 3 summarizes the performance of the different versions on *set1*. Detailed results for each instance and the full experimental results can be found at http://wwwprof.uniandes.edu.co/~pylo/inst/ALNSRPPTW/instances.html.

Table 3 shows that the most efficient removal versions are VR2, VR1, and VR4: they found the optimal solution for at least 92% of times. Version VR5 has the worst performance, finding the optimal solution for just 57% of times. All the single removal versions except VR5 outperform FV. Version VR7 performs well for instances with fewer than 35 required edges. The columns showing the single insertion versions indicate that version VI2 outperforms VI1 in most of the

TABLE 3. Performance of single versions of the ALNS on *set1*.

R	*TW*	*n.e.*	VR1	VR2	VR3	VR4	VR5	VR6	VR7	VI1	VI2	FV
≤10	10	215	**215**	**215**	**215**	**215**	181	**215**	**215**	**205**	204	204
	30	215	212	212	202	211	154	**214**	207	164	**188**	190
	50	215	**205**	**205**	204	**205**	163	200	203	160	**195**	194
12	10	45	**45**	**45**	**45**	**45**	32	**45**	**45**	44	**45**	**45**
	30	45	**45**	**45**	**45**	**45**	17	44	**45**	27	**41**	37
	50	45	42	44	37	**45**	11	34	35	14	**33**	33
16	10	15	**15**	**15**	**15**	**15**	**15**	**15**	**15**	**15**	**15**	**15**
	30	15	**15**	**15**	**15**	**15**	**15**	**15**	**15**	**15**	**15**	**15**
	50	15	14	**15**	10	14	5	**15**	**15**	5	**9**	11
21	10	30	**30**	**30**	29	**30**	14	25	**30**	24	**26**	29
	30	30	**30**	**30**	**30**	29	7	29	29	5	**17**	19
	50	30	**30**	28	27	28	1	23	**30**	0	**20**	20
27	10	20	**15**	**15**	**15**	**15**	5	**15**	**15**	3	**14**	13
	30	20	**20**	**20**	15	**20**	6	15	**20**	0	**16**	13
	50	20	10	**19**	6	15	0	2	7	0	**10**	11
35	10	30	**28**	27	23	25	5	18	25	8	**27**	24
	30	30	16	**17**	10	10	1	8	11	0	**10**	10
	50	30	24	**30**	15	26	5	16	22	5	**20**	18
45	10	20	**14**	6	10	9	1	7	8	0	**7**	6
	30	20	9	**11**	2	9	0	1	7	0	**2**	3
	50	15	4	**8**	2	6	0	0	1	0	**5**	2
Total		1120	1038	1052	972	1032	638	956	1000	694	919	912

The table compares the different single versions of ALNS and the full version. The first column shows the number of required edges; the second gives the width of the time windows. Column *n.e.* indicates the number of experiments performed for a data set of size *R* (number of required edges). The other columns indicate how many times the optimal solution was reached by the corresponding version of the ALNS. Bold entries indicate that the version solved a maximum number of problems to optimality (for single removal and single insertion versions respectively). The last row shows the number of times the optimal solution was found over all the experiments by each version.

TABLE 4. Gaps for single versions of the ALNS on *set1*.

Gap(%)	*TW*	VR1	VR2	VR3	VR4	VR5	VR6	VR7	VI1	VI2	FV
Average gap	10	**0.080**	0.154	**0.130**	0.145	1.154	0.165	**0.123**	0.635	**0.232**	0.249
	30	**0.124**	**0.075**	0.568	**0.172**	2.851	0.461	0.242	2.754	**0.931**	0.924
	50	**0.336**	**0.206**	0.494	**0.231**	4.201	1.062	0.675	4.763	**0.774**	0.922
Gap instances that did not reach the optimal value	10	2.319	2.633	**2.126**	2.584	3.546	**1.770**	**2.099**	3.133	**2.348**	2.398
	30	**1.659**	**1.121**	3.806	**1.793**	6.109	3.526	2.218	6.298	**4.059**	3.936
	50	**3.035**	3.637	**2.649**	**2.755**	8.403	4.912	4.379	9.476	**3.673**	4.212
Maximum gap	10	**3.753**	5.622	**3.753**	4.836	10.526	**3.753**	4.771	7.631	**7.609**	7.609
	30	**3.644**	**2.963**	13.889	**4.815**	21.875	8.713	6.061	19.149	**18.750**	14.063
	50	**8.333**	**8.858**	9.091	**8.333**	34.146	15.287	15.873	31.746	**18.182**	26.829

The bold entries indicate the three best gaps for the single removal versions and FV. The bold entries in columns VI1 and VI2 indicate the better gap for the two single insertion versions.

cases: VI2 was effective for 82% of times and version VI1 for 62%. The performance of FV was similar to that of VI2.

The gaps are shown in Table 4. The gap is averaged over all the experiments of a given time-window width. We also calculate the average gap for instances that did not reach the optimal value. The last three rows of the table indicate the maximum gap over all the experiments for each single version of the ALNS.

The average gap was less than 1.1% for all the single removal versions except VR5. VR1, VR2, and VR4 had the smallest average gaps; these gaps were always better than those of FV. Versions VR1, VR2, and VR4 had the smallest maximum gaps on the hardest instances ($TW = 50$). For the insertion versions, VI2 achieved better gaps than VI1 in all cases.

Tables 5 and 6 present the results of other tests. We applied the ALNS with the three best removal heuristics: R1, R2, and R4. Version VR124I2 includes I2 as the insertion heuristic, and VR124I12 includes both I1 and I2. Table 5 lists how many times these versions found the optimal solution for all the instances.

Version VR124I12 performs better than VR12I2. Note that although VR2 outperforms VR1, the algorithm obtains better results when both insertion heuristics are included. Overall,

TABLE 5. Performance of versions VR124I2 and VR124I12 on *set1*.

R	*TW*	*n.e.*	VR124I2	VR124I1I2
≤10	10	215	**215**	**215**
	30	215	210	**214**
	50	215	**205**	**205**
12	10	45	**45**	**45**
	30	45	**45**	**45**
	50	45	42	44
16	10	15	**15**	**15**
	30	15	**15**	**15**
	50	15	**15**	**15**
21	10	30	**30**	**30**
	30	30	**30**	**30**
	50	30	29	**30**
27	10	20	**15**	**15**
	30	20	**20**	**20**
	50	20	12	18
35	10	30	29	**30**
	30	30	13	**21**
	50	30	23	29
45	10	20	13	13
	30	20	14	**15**
	50	15	7	**10**
Total		1120	1042	1074

The table should be interpreted as Table 2. Bold entries indicate when the version solved a maximum number of problems to optimality over all versions of the ALNS (VR1 to VR7, FV, VI1, VI2, VR124I2, and VR124I12).

VR124I12 is the best version. It finds the optimal solution for 95.9% (1,074/1,120) of times.

Table 6 lists the gaps for VR124I2 and VR124I12. It shows that VR124I12 is quite stable: the average gap never exceeds 0.2% and it has the smallest average gap of all versions of the ALNS.

Finally, we compared the performance of the best version with that of the cutting plane algorithm [13]. The results are summarized in Table 7. VR124I12 fails to find the optimal solution for only 6 instances. Moreover, the computational time of the cutting plane algorithm increases significally on the hardest instances: ALNS solves the hardest problems in less than 5.5 min. on average while the cutting plane algorithm requires 34.3 min.

5.4. Comparison to a Heuristic Solution

We compare the performance of our solution method with other heuristic solution. Since there is not previous heuristic works dealing with this problem, we use a commercial local search solver. We compare the average gap of 5 runs of the version VR124I12 with the average gap of 5 runs of the local search solver for each of the instances in *set1*. The results are summarized in Table 8. We set the running time of the solver to the maximum running time of our method for each instance.

The local search solver was able to find feasible solutions only for instances with up to 12 requests. Our method shows a much better performance: we were able to find feasible solutions for all of the instances considered in Table 8; conversely, when it is possible to compare the gap of ALNS and local search, ALNS gets always better gaps, indeed, in most of the cases it got the optimal solution.

TABLE 6. Gaps for VR124I2 and VR124I12 on *set1*.

Gap (%)	*TW*	VR124I2	VR124I1I2
Average	10	0.098	**0.078**
	30	0.302	**0.043**
	50	0.387	**0.163**
Instances that did not reach the optimal value	10	2.816	2.447
	30	4.044	**1.071**
	50	3.867	3.183
Maximum	10	**3.753**	**3.753**
	30	13.889	**1.887**
	50	15.873	**8.333**

The interpretation is as for Table 3. Bold entries indicate when the corresponding version obtained the smallest gap over all versions (VR1 to VR7, FV, VI1, VI2, VR124I2, and VR124I12).

5.5. Failures

This section presents the characterization of the cases where the best version of our algorithm was not successful. In Table 9, we report all the cases where VR124I12 did not find the optimum value, this considering the five times we run VR124I12 for each instances in *set1*.

The values of average and maximum gap reported in Table 9 were computed for the failure cases only (column *num*). As expected, failures increase with the width of the time windows and with a higher number of required edges. Regarding the number of connected components, it seems that it does not have an evident effect.

5.6. Results for set2

Our final experiments were carried out on eight larger instances based on a real network [13]. We applied only VR124I12, because it had good results in the previous tests.

We applied the ALNS with different values of q, and we used different limits on the computational time. We set $q = 3$, 5, 10, 40, or a random number between $0.1|R|$ and $0.2|R|$. We set the time limit to 180, 600, 900, and 1,800 s. We applied the ALNS five times for each instance and each combination of q and the time limit. We stopped the algorithm when it reached 25,000 iterations or the time limit.

Because we are interested in an algorithm that works well quickly, we analyze the quality of the solutions in terms of the running times. We also analyze the effect of the parameter q, which has the largest impact on the solution time: when more requests can be removed in a single iteration the solution time is higher.

Table 10 presents the average gap for all the experiments. The average gap is not reported if the algorithm uses less time to reach 25,000 iterations than the time limit; we reallocated those experiments to a group based on the upper bound of

TABLE 7. Comparison of VR124I12 and cutting plane algorithm.

TW		10					30					50				
		VR124I12			Cutting plane		VR124I12			Cutting plane		VR124I12			Cutting plane	
R	*Ins*	*sol*	*Avg-t*	*Max-t*	*Avg-t*	*Max-t*	*Sol*	*Avg-t*	*Max-t*	*Avg-t*	*Max-t*	*sol*	*Avg-t*	*Max-t*	*Avg-t*	*Max-t*
≤10	43	43	7.35	11.50	**0.13**	**0.83**	43	7.65	12.61	**0.18**	**0.79**	41	7.80	12.23	**0.27**	**0.98**
12	9	9	20.22	22.04	**0.51**	**0.81**	9	22.01	24.74	**1.60**	**2.85**	9	22.98	25.24	**1.27**	**1.99**
16	3	3	31.86	34.25	**1.68**	**2.31**	3	36.03	38.69	**3.32**	**7.27**	3	36.19	39.77	**4.70**	**6.53**
21	6	6	58.49	64.60	**4.26**	**9.65**	6	63.64	69.84	**6.74**	**9.14**	6	68.05	75.19	**10.34**	**33.70**
27	4	3	87.52	96.44	**8.44**	**10.75**	4	95.59	103.92	**16.36**	**34.08**	4	**99.20**	**109.10**	1484.43	5494.11
35	6	6	145.36	175.01	**29.14**	**46.00**	5	151.51	179.98	**44.44**	**83.02**	6	**174.50**	**187.26**	6110.53	23729.59
45	4	3	257.88	287.62	**44.63**	**60.23**	3	**301.05**	**332.02**	6892.40	25779.46	4	**306.25**	**335.30**	2060.24	6030.17

The table gives the number of problems solved to optimality and the computational times. Column *Ins* shows the number of instances of size *R*. The data set is divided into three groups: $TW = 10$, $TW = 30$, and $TW = 50$. For each group we report how many problems were solved to optimality (*sol*) and the average (*Avg-t*) and maximum (*Max-t*) ALNS computational time. The columns for the cutting plane algorithm show the average and maximum computational times reported by Monroy-Licht et al. [13]. Bold entries indicate the faster method.

TABLE 8. VR124I12 and a local search solver.

TW		10			30			50		
		ALNS	*Local Search*		*ALNS*	*Local Search*		*ALNS*	*Local Search*	
R	*Ins*	*Gap*	*Feasible*	*Gap*	*Gap*	*Feasible*	*Gap*	*Gap*	*Feasible*	*Gap*
2	3	0	3	0	0	3	0	0	3	0
3	9	0	9	0	0	9	0.672	0	9	0
4	3	0	3	0	0	3	0	0	3	0
5	3	0	2	0	0	3	0	0	1	0
7	15	0	3	1.274	0	6	9.382	0.121	13	7.400
8	3	0	1	0	0	0	-	2.778	0	-
9	4	0	0	-	0.057	0	-	0	0	-
10	3	0	0	-	0	0	-	0	1	0
12	9	0	0	-	0	0	-	0	1	3.172
16	3	0	0	-	0	0	-	0	0	-
21	6	0	0	-	0	0	-	0	0	-
27	4	0.938	0	-	0	0	-	0.107	0	-
35	6	0	0	-	0.374	0	-	0.058	0	-
45	3	0.530	0	-	0.186	0	-	1.023	0	-

The columns are as follows. *R* gives the number of required edges and *Ins* the number of instances with optimum value known. For each width time windows the column *ALNS Gap* represents the average gap of the version VR124I12, *Feasible* the number of instances with a feasible solution found by the solver, and *Gap* shows the gap to the optimal solution. No values are reported as "-."

TABLE 9. Failure cases.

TW	*Instance*	*R*	*Connected*	*Num*	*Ave*	*Max*
10	TW-B60A114	45	10	2	0.367	0.367
	TW-B60A123	**45**	**11**	**5**	**1.973**	**2.321**
	TW-C60A102	**27**	**6**	**5**	**3.753**	**3.753**
30	**TW-A40D232**	**35**	**5**	**5**	**0.755**	**0.755**
	TW-B60A115	**45**	**8**	**5**	**0.744**	**1.370**
	TW-C40C151	35	5	3	1.852	1.852
	TW-C40D142	35	5	1	1.887	1.887
	TW-C60A100	9	4	1	1.136	1.136
50	**TW-A13C68**	**7**	**4**	**5**	**1.818**	**1.818**
	TW-A13C71	12	2	1	2.857	2.857
	TW-A60A134	45	7	2	0.216	0.216
	TW-B10A53	**8**	**3**	**5**	**8.333**	**8.333**
	TW-C40C152	35	4	1	1.730	1.730
	TW-C60A104	27	10	2	1.072	1.072
	TW-C60A107	45	9	3	0.853	1.772

The columns correspond to: the width of the time windows, the name of the instance, the number or required edges, the number of connected components, the number of failures, the average gap and the maximum gap. Bold entries indicate the cases where VR124I12 was not able to find the optimal solution for all the 5 runs.

their running times. For $q = 5$ the algorithm obtained the best average gap in 4 instances, using a maximum of 685 s. For $q = 10$ and a time limit of 1,800 s, we obtained the smallest gaps for 4 instances.

Table 11 summarizes the best results obtained by the algorithm in each instance. The first three columns present the features of each instance. The value of the best solution obtained by the algorithm is presented in column *ALNS*. The corresponding values of q and the time limit are given in the column *Best combination*. The average computational time in seconds is presented in column T. The number of times the algorithm obtained the optimal solution is given in column n. The last columns give the optimal value, the computational time in seconds for the cutting plane algorithm [13], and the gap. Note that in some cases the running time T is smaller than the time limit because the algorithm has reached the limit on the number of iterations. Although the running times of the exact method are in general better, the metaheuristic finds

TABLE 10. Average gap: Study of the effect of q and *time limit*.

Instance	Q	Time limit (sec.) 180	600	900	1800	Avg. gap - all time limits (%)
	Random	3.363	2.814	2.688	2.281	2.793
Inst-01	3	4.585	4.368			4.422
$\|R\| = 74$	5	3.627	2.922			3.098
$TW = 30$	10	3.591	3.148	3.174	2.969	3.220
	40	34.355	18.820	9.997	7.975	17.787
	Avg. gap - all q (%)	9.904	5.183	5.230	4.456	6.264
	Random	4.317	3.641	3.594	3.272	3.706
Inst-02	3	4.347	4.412			4.396
$\|R\| = 74$	5	3.838	3.630			3.682
$TW = 50$	10	4.190	3.695	3.585	3.219	3.672
	40	10.980	7.671	7.275	5.556	7.871
	Avg. gap - all q (%)	5.535	4.348	4.818	4.016	4.665
	Random	9.232	7.143	5.121	5.212	6.677
Inst-06	3	5.702	3.766			4.250
$\|R\| = 93$	5	5.942	2.854			3.626
$TW = 10$	10	9.633	5.461	3.806	3.516	5.604
	40	21.062	15.696	13.904	11.851	15.505
	Avg. gap - all q (%)	10.169	5.427	7.467	6.968	7.132
	Random	4.341	3.361	2.646	1.613	2.990
Inst-07	3	3.235	3.213			3.219
$\|R\| = 93$	5	2.853	1.905			2.142
$TW = 30$	10	3.809	1.891	1.399	0.528	1.907
	40	12.253	7.577	7.205	5.176	7.897
	Avg. gap - all q (%)	5.108	3.196	3.750	2.439	3.631
	Random	4.582	2.509	2.485	1.732	2.827
Inst-08	3	4.544	3.645			3.870
$\|R\| = 93$	5	3.122	2.467			2.631
$TW = 50$	10	4.253	2.596	1.726	0.851	2.357
	40	9.232	7.315	4.834	4.701	6.486
	Avg. gap - all q (%)	5.091	3.446	2.974	2.478	3.634
	Random	9.682	5.519	6.837	5.252	6.823
Inst-03	3	4.159	2.965			3.264
$\|R\| = 104$	5	2.548	1.472	1.470		1.740
$TW = 10$	10	6.126	3.909	3.141	2.461	3.910
	40	20.702	15.902	16.507	14.524	16.909
	Avg. gap - all q (%)	8.643	4.358	8.368	7.413	6.529
	Random	3.568	2.743	2.414	2.155	2.720
Inst-04	3	2.583	2.247			2.331
$\|R\| = 104$	5	1.421	0.831	0.884		0.983
$TW = 30$	10	2.595	1.861	1.593	0.984	1.758
	40	17.825	13.789	11.801	10.634	13.378
	Avg. gap - all q (%)	5.263	3.317	4.831	4.591	4.234
	Random	5.534	4.770	3.758	2.519	4.145
Inst-05	3	2.742	2.691			2.704
$\|R\| = 104$	5	1.753	1.266	0.968		1.258
$TW = 50$	10	4.170	2.676	1.932	1.646	2.606
	40	20.196	14.820	12.748	11.986	14.938
	Avg. gap - all q (%)	6.879	4.396	4.250	5.384	5.130
	Avg. gap - all instances (%)	**7.085**	**4.205**	**5.167**	**4.719**	**5.153**

The columns are as follows. Instance gives the name and description of each instance, q is the number of requests to remove and insert in each iteration, and the next four columns are the maximum running time for each time limit. The final column is the average gap for all experiments with the corresponding value of q. The table also reports for each instance the average gap over all experiments with a given time limit. The last row of the table gives the average gap over all instances with a given time limit.

TABLE 11. Best solutions: *set2*.

Instance	*R*	*TW*	*ALNS*	*Best combinationq, time limit*	*T*	*N*	*OF**	*t**	*Gap (%)*
Inst-01	74	30	259.7	random,600	600	1	259.7	**165.46**	0.00
				random, 1800	1654	1			
Inst-02	74	50	261.2	5,1800	456	0	259.7	**366.17**	0.59
Inst-06	93	10	301.0	5, 600	545	0	299.7	**193.41**	0.45
Inst-07	93	30	299.7	random, 1800	1800	1	299.7	**208.75**	0.00
				5, 600	499	1			
				10, 1800	1695	4			
Inst-08	93	50	299.7	random, 600	600	1	299.7	**137.76**	0.00
				random, 1800	1800	3			
				10, 1800	1678	1			
Inst-03	104	10	289.2	5, 180	**180**	1	289.2	202.29	0.00
				5, 900	582.50	2			
				5, 1800	555.66	3			
Inst-04	104	30	289.2	5, 600	585.8	5	289.2	615.35	0.00
				5, 900	614	1			
				5, 1800	**580.25**	4			
Inst-05	104	50	289.2	5, 600	600	2	289.2	**366.88**	0.00
				5, 900	641	3			
				5, 1800	632	2			

Bold entries indicate the faster method.

good solutions; it solves six instances to optimality, and the minimum gap for the other two is less than 0.6%.

5.7. *Summary of Computational Results*

We have applied different versions of the ALNS to compare the performance of seven removal and two insertion heuristics and a full version that includes all of them. The selection of the removal heuristic has an impact on the solution quality: the removal heuristics based on randomness (R1 and R2) and the one that considers distance requirements (R4) perform better than the others. I2, the insertion heuristic that prioritizes the most difficult insertions, outperforms I1, which prioritizes the cheapest insertions.

The most efficient version is VR124I12; it uses R1, R2, and R4 as removal heuristics and I1 and I2 as insertion heuristics. It was able to find 218 optimal solutions for *set1* and 6 for *set2*.

The results show that the ALNS performs well; for the hardest instances ($TW = 50$) the average gap of VR124I12 was 0.163%, and it required less than 336 s in the worst case; the exact method takes 20 times as long.

For larger instances, the number of requests that can be removed and inserted at each iteration has an important impact on the solution time. For $q = 5$ the average gap is 2.39% and the maximum running time is 685 s.

In the case of the instances based on the real network, the metaheuristic found 6 optimal solutions out of 8 problems and the gap reported for the other two problems was less than 0.6%. However, the running times of the exact method in this set of instances were on average better.

Although the instances in *set2* have more required edges than those in *set1*, the latter are harder to solve because of their time-window structure.

6. CONCLUSIONS

We have proposed an ALNS algorithm for the RPPTW. We have proposed and evaluated seven removal heuristics and two insertion heuristics. The best results were achieved when the algorithm used a combination of the best removal heuristics.

The ANLS is robust and performs well compared to exact methods. It found 224 optimal solutions for the 232 benchmark instances while significantly reducing the computational time on the hardest instances.

As future research the quality of the initial solution should be measured as it has an impact in the performance of the metaheuristic regarding to quality solutions and computational time: we allow infeasible initial solutions, and therefore the ALNS algorithm may take more time to converge to a good solution. In addition, the problems of time windows given by time slots may suggest using other heuristics that remove and insert requests with different rules to those proposed here. Finally, we plan to consider inclusion of other constraints in the RPPTW as for example the prohibition of U-turns, for safety reasons, and the extension to the case of multiple vehicles.

ACKNOWLEDGMENTS

The authors thank two anonymous referees for valuable comments and suggestions to improve the paper.

REFERENCES

[1] B. Alidaee and N.K. Womer, Scheduling with time dependent processing times: Review and extensions, J Oper Res Soc 50 (1999), 711–720.

[2] J.F. Campbell, A. Langevin, and N. Perrier, "Advances in vehicle routing for snow plowing," Arc routing: Problems, methods, and applications, Á. Corberán and G. Laporte (Editors), SIAM, 2014, pp. 321–350.

[3] A. Corberán and J.M. Sanchis, A polyhedral approach to the rural postman problem, Eur J Oper Res, 79 (1994), 95–114.

[4] M. Dror, Arc routing: Theory, solutions and applications, Kluwer Academic Publishers, Boston, 2000.

[5] R. Eglese, B. Golden, and E. Wasil, "Route optimization for meter reading and salt spreading," Arc routing: problems, methods, and applications, Á. Corberán and G. Laporte (Editors), SIAM, Philadelphia, 2014, pp. 303–320. (MOS/SIAM Series on Optimization; 20).

[6] R.W. Eglese, Routeing winter gritting vehicles, Discrete Appl Math 48 (1994), 231–244.

[7] L. Fu, M. Trudel, and V. Kim, Optimizing winter road maintenance operations under real-time information, Eur J Oper Res 196 (2009), 332–341.

[8] N. Golbaharan, An application of optimization to the snow removal problem: A column generation approach, Division of Optimization, Department of Mathematics, Linköpings university, 2001.

[9] T. Ibaraki, S. Imahori, M. Kubo, T. Masuda, T. Uno, and M. Yagiura, Effective local search algorithms for routing and scheduling problems with general time-window constraints, Transp Sci 39 (2005), 206–232.

[10] M.-J. Kang and C.-G. Han, "Comparison of crossover operators for rural postman problem with Time Windows," Soft computing in engineering design and manufacturing, P.K. Chawdhry, R. Roy, and R.K. Pant (Editors), Springer, London, 1998, pp. 259–267.

[11] J.K. Lenstra and A.H.G.R. Kan, On general routing problems, Networks 6 (1976), 273–280.

[12] A.N. Letchford and R.W. Eglese, The rural postman problem with deadline classes, Eur J Oper Res 105 (1998), 390–400.

[13] M. Monroy-Licht, C.A. Amaya, and A. Langevin, The Rural Postman Problem with Time Windows, Networks, 64 (2014), 169–180.

[14] Y. Nobert and J.C. Picard, A heuristic algorithm for the Rural Postman Problem with Time Windows. Paper presented at the ORSA/TIMS, Detroit, 1994.

[15] Office of the Auditor General of Ontario, Winter highway maintenance Special report, April 2015, Queen's Printer for Ontario, Toronto, 2015, p. 44.

[16] A. Orda and R. Rom, Shortest-path and minimum-delay algorithms in networks with time-dependent edge-length, J ACM 37 (1990), 607–625.

[17] N. Perrier, J.F. Campbell, M. Gendreau, and A. Langevin, "Vehicle routing models and algorithms for winter road spreading operations," Hybrid algorithms for service, computing and manufacturing systems: Routing and scheduling solutions, R.M.-T. Jairo, A. J. Angel, H. Luisa Huaccho, F. Javier, and L. R.-V. Gloria (Editors), IGI Global, Hershey, PA, 2012, pp. 15–45.

[18] N. Perrier, A. Langevin, and J.F. Campbell, A survey of models and algorithms for winter road maintenance. Part I: system design for spreading and plowing, Comput Oper Res 33 (2006), 209–238.

[19] N. Perrier, A. Langevin, and J.F. Campbell, A survey of models and algorithms for winter road maintenance. Part II: system design for snow disposal, Comput Oper Res, 33 (2006), 239–262.

[20] N. Perrier, A. Langevin, and J.F. Campbell, A survey of models and algorithms for winter road maintenance. Part III: Vehicle routing and depot location for spreading, Comput Oper Res 34 (2007), 211–257.

[21] N. Perrier, A. Langevin, and J.F. Campbell, A survey of models and algorithms for winter road maintenance. Part IV: Vehicle routing and fleet sizing for plowing and snow disposal, Comput Oper Res, 34 (2007), 258–294.

[22] D. Pisinger and S. Ropke, A general heuristic for vehicle routing problems, Comput Oper Res 34 (2007), 2403–2435.

[23] G. Razmara, Snow removal routing problems: Theory and applications. (PhD.), Linköping University, Linköping, Sweden, 2004.

[24] J.-P. Riquelme-Rodríguez, A. Langevin, and M. Gamache, Adaptive large neighborhood search for the periodic capacitated arc routing problem with inventory constraints, Networks 64 (2014), 125–139.

[25] S. Ropke and D. Pisinger, An adaptive large neighborhood search heuristic for the pickup and delivery problem with Time Windows, Transp Sci 40 (2006), 455–472.

[26] S. Ropke and D. Pisinger, A unified heuristic for a large class of Vehicle Routing Problems with Backhauls, Eur J Oper Res 171 (2006), 750–775.

[27] M.A. Salazar-Aguilar, A. Langevin, and G. Laporte, Synchronized arc routing for snow plowing operations, Comput Oper Res 39 (2012), 1432–1440.

[28] M.A. Salazar-Aguilar, A. Langevin, and G. Laporte, The synchronized arc and node routing problem: Application to road marking, Comput Oper Res 40 (2013), 1708–1715.

[29] S. Slone, High costs of winter road maintenance, 2013-14, Capitol research, 4 (2014), Available at: http://knowledgecenter.csg.org/kc/system/files/CR_WinterMaintenanceCosts.pdf website. Last accessed date May 15, 2015

[30] J. Sun, Y. Meng, G. Tan, and J. Sun, Solving the Time Varying Postman Problems with Timed Automata, Optim Methods Softw 30 (2015), 804–824.

[31] P.S. Sundararaghavan and A.S. Kunnathur, Single machine scheduling with start time dependent processing times: Some solvable cases, Eur J Oper Res 78 (1994), 394–403.

[32] M. Tagmouti, M. Gendreau, and J.-Y. Potvin, Arc routing problems with time-dependent service costs, Eur J Oper Res 181 (2007), 30–39.

[33] M. Tagmouti, M. Gendreau, and J.-Y. Potvin, A variable neighborhood descent heuristic for arc routing problems with time-dependent service costs, Comput Ind Eng 59 (2010), 954–963.

[34] M. Tagmouti, M. Gendreau and J.-Y. Potvin, A dynamic capacitated arc routing problem with time-dependent service costs, Transp Res Part C Emerg Technol 19 (2011), 20–28.

[35] G. Tan and J. Sun, "An integer programming approach for the rural postman problem with time dependent travel times," Computing and combinatorics Vol. 6842, B. Fu and D.-Z. Du (Editors), Springer, Berlin Heidelberg, 2011, pp. 414–431.

[36] G. Tan, J. Sun, and G. Hou, The time-dependent rural postman problem: polyhedral results, Optim Methods Softw 28 (2013), 855–870.